PROJECT	IRONSTONE
REPORT TITLE	Analysis of DLSM Protocol
AUTHORS	Petr Matousek (Brno University of Technology)
REVIEWERS	
ABSTRACT	DLMS (Device Language Message Specification) / COSEM (Companion Specification for Energy Metering) describes an interface model and communication protocols for data exchange with metering equipment. DLMS/COSEM specification includes description of exchanged objects, identification of objects (addressing), specification of messages and transporting methods. This document described the basic features of DLMS/COSEM and specifies format of the packets and how messages are exchanged.
STATUS	DRAFT

Table of Contents

1 Introduction

DLMS (Device Language Message Specification, originally Distribution Line Message Specification, IEC 62056-5-3)[1] is an application layer specification designed to support messaging to and from (energy) distribution devices. Applications like remote meter reading, remote control and value added services for metering any kind of energy, like electricity, water, gas, or heat are supported. DLMS specification is used to describe interface classes for various objects available (voltage, current) with their attributes.

DLMS specification is developed and maintained by the DMLS User Association (DLMS UA)[1]. For electricity metering, IEC TC13 WG14 has established the IEC 62056 series of standards for electricity metering data exchange[2].

IEC 62056 is a set of standards for electricity metering, data exchange for meter reading, tariff and load control established by International Electrotechnical Commission (IEC). This series includes the following standards:

- IEC 62056-21: Direct local data exchange
- IEC 62056-42: Physical layer services and procedures for connection-oriented asynchronous data exchange
- IEC 62056-46: Data link layer using HDLC protocol
- IEC 62056-47: COSEM transport layers for IPv4 networks
- IEC 62056-51: Application layer protocols
- IEC 62056-53: COSEM Application layer
- IEC 62056-61: Object identification system (OBIS)
- IEC 62056-62: Interface classes

IEC 62056 standards are focused on electricity metering while DLMS/COSEM is more general and applied to any energy metering. Communication standards differs, e.g., IEC 62056-21 is ASCII based communication while DLMS is a binary protocol.

This document provides overview of COSEM modeling of metering devices, addressing and DLMS communication. The focus of this study is on the format of DLMS/COSEM communication for the purpose of security monitoring.

[1] See http://dlms.com/index2.php (last access in June 2017).

[2] See http://www.dlms.com/documentation/dlmscosemspecification/iecstandardsforelectricitymetering.html (last accessed in June 2017)

2 Companion Specification for Energy Metering (COSEM)

COSEM [1] is an interface model of communicating energy metering equipment that provides a view of the functionality available through the communication interface. It provides semantics for metering application. COSEM model uses an object-oriented approach. An instance of a COSEM interface class is called *COSEM interface object*. The set of objects instantiated in the logical devices of a physical device model the functionality of the metering equipment as it is seen through its communicating interfaces.

The COSEM model represents the meter as a server used by client applications that retrieve data from, provide control information to, and instigate known actions within the meter via controlled access to the attributes and specific methods of objects making up the server interface. The client may be supporting the business processes of utilities, customers, meter operators, or meter manufacturers.

2.1 Physical and Logical Devices

COSEM models metering equipment as *physical devices* (physical metering equipment), containing one or more *logical devices*. Each logical device contains a number of COSEM objects, modelling the functionality of the logical device. Each logical device supports one or more application associations with clients. Each logical device is uniquely identified by its logical device name.

The COSEM server is structured into three hierarchical levels: physical device, logical device, and accessible COSEM objects, see the following Figure 1.

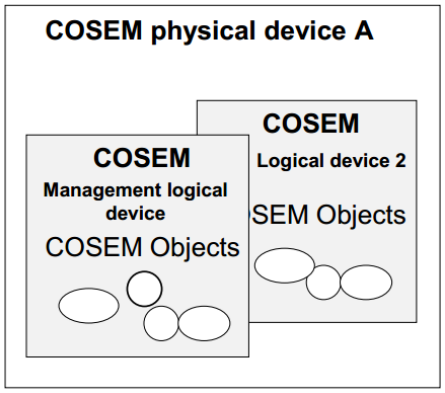


Figure 1: The COSEM server model

- A physical device hosts one or several logical devices. A logical device models a specific functionality of the physical device. Each physical device shall contain a “Management logical device”. For example, in a multi-energy meter, on logical device could be an electricity meter, another, a gas meter, etc., see Figure 2.

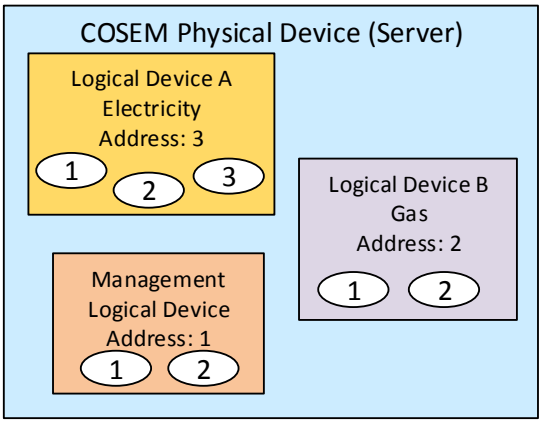


Figure 2: A physical device hosting three logical devices: Management (1), gas (2), and electricity (3)

- A logical device is a container for COSEM objects. A COSEM object is simply a structured piece of information with attributes and methods. All objects that share the same structure are of the same COSEM class. There are many COSEM classes [2,13], see Appendix A.
- Each logical device can be identified by its unique *logical device name* (LDN). This name can be retrieved from an instance of IC *SAP assignment* (*class_id=17*) or from a COSEM object *COSEM logical device name* of the IC *Data*, see Table 1.

COSEM logical device name object	IC	OBIS code					
		A	B	C	D	E	F
COSEM logical device name	1, Data [a]	0	0	42	0	0	255
[a] If the IC "Data" is not available, "Register" (with scaler = 0, unit = 255) may be used.							

Table 1: COSEM logical device name

- The LDN is defined as an octet-string of up to 16 octets. The first three octets carry the manufacturer identifier that is administered by the DMLS User Association[3], see Table 2. The manufacturer shall ensure that LDN, starting with the three octets identifying the manufacturer and followed by up to 13 octets, is unique.

Manufacturers Identification Characters	
Flag	**Company**
AAA	Aventies GmbH, Linzerstraße 25, 53577 Neustadt/Wied, Germany
ABB	ABB AB, P.O. Box 1005, SE-61129 Nyköping, Nyköping,Sweden
ABN	ABN Braun AG, Platenstraße 59, 90441 Nürnberg, Germany
ABR	ABB s.r.o., Videnska 117, Brno, Czech Republic
ACA	Acean, Zi de la Liane, BP 439, 62206 Boulogne Sur Mer Cedex, FRANCE
ACB	AcBel Polytech Inc., No. 159, Sec. 3, Danjin Rd., Tamsui Dist.,New Taipei, Taiwan (R.O.C.)
ACC	Accurate (Pvt) Ltd, Office # 2, first floor, Ross Residentia, 234 Dhana Singhwala, 1 Campus Road Canal Bank Lhr, Lahore, Pakistan
ACE	Actaris, France. (Electricity) C/O Itron S.A.S 52, Rue Camille Desmoulin, 92130 Issy-les-Moulineaux, FRANCE
ACG	Actaris, France. (Gas) C/O Itron S.A.S 52, Rue Camille Desmoulin, 92130 Issy-les-Moulineaux, FRANCE
ACH	Acantho S.p.A. Via Molino Rosso 8 40026 Imola Italy
ACL	Aclara Meters UK Ltd, Lothbury House, Newmarket Road, Cambridge, CB5 8PB, UK
...	

Table 2: Example of Manufacturers Flags registered by DLMS User Association

[3] See https://www.dlms.com/organization/flagmanufacturesids/index.html [last accessed in Jan 2018]

2.2 COSEM Interface Classes and Objects

COSEM interface classes (ICs) follow the object-oriented approach where an object is a collection of attributes and methods. Attributes represent the characteristics of an object. Each attribute contains meaning, data type and value. The value of an attribute may affect the behavior of an object. Methods allow performing operations on attributes, see Figure 3.

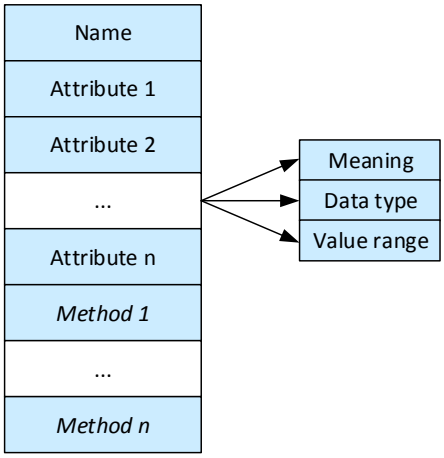


Figure 3: Attributes and methods constitute an object

- The first attribute of any object is the logical name which is a part of the identification of the object. An object may offer a number of methods to either examine or modify the values of the attributes.
- Objects that share common characteristics are generalized as an IC and identified with a *class_id*. Instantiations of ICs are called COSEM interface objects.

Relation between the class template, the specific interface class, and objects is in Figure 4.

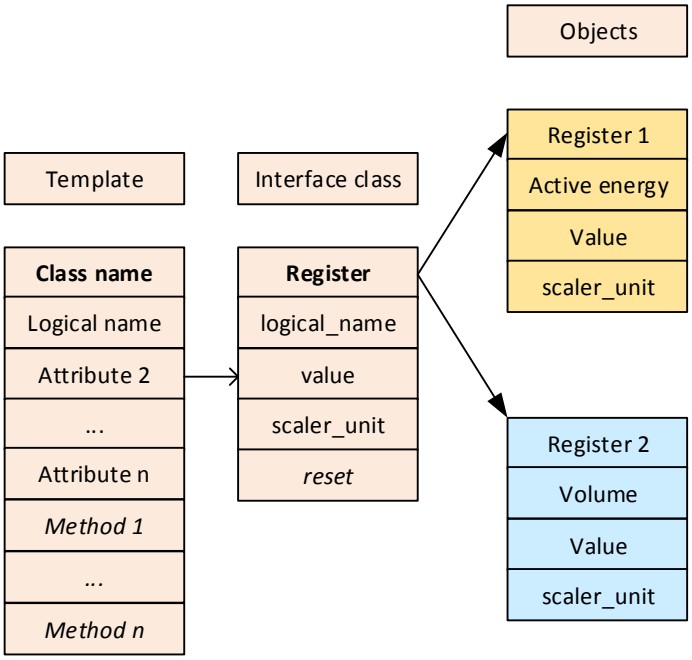


Figure 4: Interface class and objects

Figure 5 illustrates the COSEM object model on an example. The IC "Register" with *class_id=3* models the behavior of a generic register (containing measured or static information). The contents of the register are identified by the attribute *logical_name*. The *logical_name* contains an OBIS identifier (see Section 3). The actual content of the register is carried by its *value* attribute.

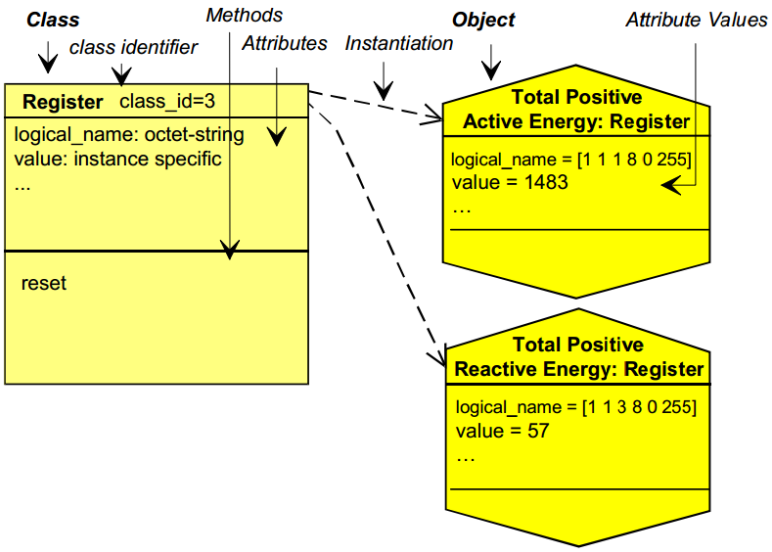


Figure 5: COSEM object model

Defining a specific meter means defining several specific objects. In the example above, the meter contains two registers, e.g., two specific instances of the interface class "Register".

Standard defines about 70 interface classes [3]. The interface class is defined by its name, class_id and version. It contains attributes and methods. Each attribute has its name, data type, minimum, maximum, and default value of the attribute, and the short name, see Figure 6. The first attribute of each interface class is the logical_name. Other attributes and methods depend on the definition of the specific interface class. An overview of standard interface classes is in given Appendix A.

Class name		**Cardinality**	**class_id, Version**		
Attribute(s)		***Data Type***	***Min.***	***Max.***	***Def***
1. logical_name	(static)	octet-string			
2.	(..)	.....			
3.	(..)	.....			
Specific Method(s) (if required)		***m/o***			
1.		.....			
2.		.....			

Figure 6: Class description template

Looking at example in Figure 5, class *Register* (*class_id=3*) has three attributes and one method:

- Attribute 1 is *logical_name* with data type *OCTET STRING* and short name *x*
- Attribute 2 is value with dynamic data type *CHOICE* and short name *x + 0x08*
- Attribute 3 is *scaler_unit* with data type *scal_unit_type* and short name *x + 0x10*
- Method 1 is reset (data) with short name *x + 0x28* .

2.3 Accessing the object via application association

In order to access COSEM objects in the server, an application association (AA) must be first established with a client. This identifies the partners and characterizes the context within which the association applications will communicate. The context includes:

- the application context
- the authentication context
- the xDLMS context

The information is contained in a special COSEM object *Association.* There are two types of the Association object depending on the name referencing: logical names (LN) or short names (SN).

Each logical device contains at least one object of class *Association LN* (class_id=15) or *Association SN* (class_id=12), see Figure 7. This object has an attribute no. 2 called *object_list* that contains the list of all objects available in the logical device. This helps us to know which objects exists in the given logical device. The association object has the predefined logical name 0.0.40.0.0.255. Therefore, we can find out what objects are available in a logical device just by reading its object list.

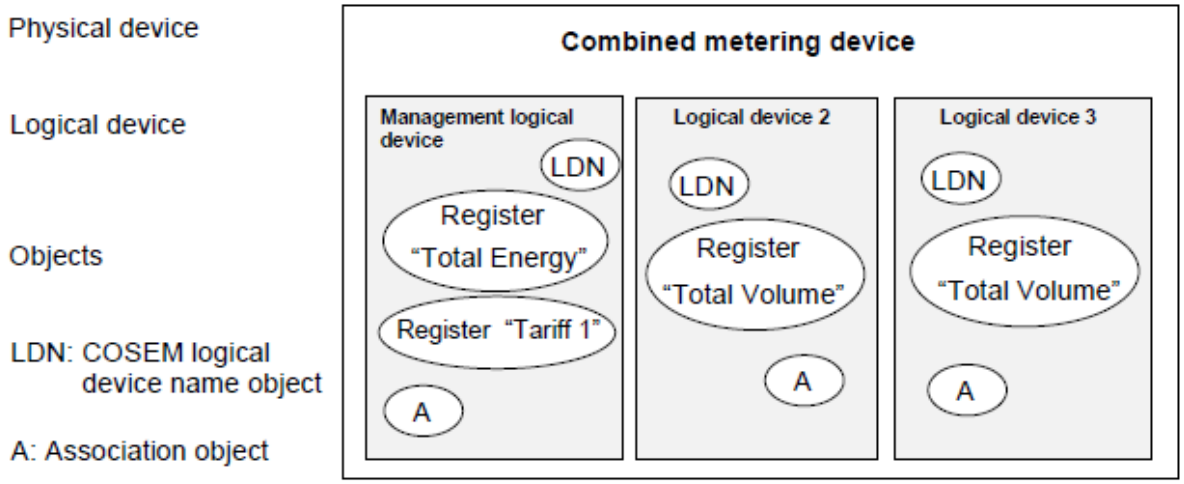


Figure 7: Combined metering device with association objects

The *Association LN* interface class (*class_id* = 15) contains eight attributes and four methods. The attributes and methods are used to establish connection with COSEM logical device within a COSEM context. The *association LN* attributes are:

1. *logical_name*
2. *object_list*
3. *associated_partner_id*
4. *application_context_name*
5. *xDLMS_context_info*
6. *authentication_mechanism_name*
7. *LLS secret*
8. *association_status*

The *Association SN* interface class (*class_id* = 12) has only two attributes:

1. *logical_name*
2. *object_list*

An instance of *Association LN* or *Association SN* interface class is initiated during association phase using DLMS AARQ and AARE message, see Section 4.4.

Using mandatory elements (COSEM objects and attributes) we can get the necessary information about the content of a physical device:

- Each physical device has a *management logical device* at address 1.
- A management logical device hosts a list of all available logical devices in the physical device. The list is the second attribute of the object class of SAP assignment with the predefined name 0.0.41.0.0.255. Each list item consists of the name and the address of a logical device.
- Each logical device hosts a list of all its available objects. The list is the second attributed of the object of class Association with the predefined name 0.0.40.0.0.255. Each list item consists of the logical name and the class of an object.

2.4 Logical Name (LN) and Short Name (SN)

Meter objects are read through the interfaces. Each object has always *Logical Name (LN)*. Manufacturers are using same logical names, so any data collector can read data from different meters with one data collection application. *Logical name* is the first attribute of any COSEM interface object. Together with the version of the interface class, the logical name defines the meaning of the object.

The logical name consists of a string of six values defined according to a system called OBIS (Object Identification System), e.g., 1.1.1.8.0.255, see Section 3. OBIS allows to uniquely identify each of the many data items used in the energy metering equipment.

Attributes and methods of COSEM objects can be accessed using *referencing*. There are two different ways how to reference COSEM objects:

- *Logical Name (LN) referencing* is a way how to access attributes and methods of COSEM interface objects using the identifier of the COSEM interface class and the COSEM object instance to which these attributes and methods belong.

  The reference for an attribute is: *class_id*, value of the *logical_name* attribute, *attribute_index*

  The reference for a method is: *class_id*, value of the *logical_name* attribute, *method_index*.

  Attribute and method indexes are specified in the definition of each IC. They are positive numbers starting with one. Proprietary attributes may be added (these use negative numbers).

- *Short name (SN)* referencing is a way how to access attributes and methods of COSEM interface objects after the first mapping them to short names. This kind of referencing is intended for use in simple devices. In this case, each attribute and method of a COSEM object is identified with a 13-bit integer. The syntax is the same as the syntax of a DLMS named variable. When SN referencing is used, each attribute and method of object instances has to be mapped to short names.

  The *base_name x* of each object instance is the DLMS name variable the logical name attribute is mapped to. It is selected in the implementation phase. The IC definition specifies the offset for the other attributes and for the methods.

  Short Name (SN) is an identifier of a COSEM interface object attribute or method, using the syntax of a DLMS named variable. Short names are assigned during the process of mapping during the design phase of a DLMS/COSEM metering equipment. For example, the class *Register* (id=3) has three attributes (*logical_name, value, scaler_unit*) and one method (*reset*). The short names for these elements are: base_name is *x* refers to the *logical_name.* The short name of the attribute *value* is *x + 0x08* (offset 0x08), the SN of *scaler_unit* is *x + 0x10*, the SN of *reset* method is *x + 0x28*.

3 COSEM Object Identification System (OBIS)

The OBIS defines the identification codes (ID-codes) for commonly used data items in metering equipment. It provides a unique identifier for all data within the metering equipment. The ID codes defined by OBIS are used for the identification of:

- logical names of the various instances of the ICs or objects,
- data transmitted through communication lines,
- data displayed on the metering equipment.

OBIS codes identify data items used in energy metering equipment, in a hierarchical structure using six value groups A to F, see Table 4. Further description of the values is in Appendix B.

Value group	Use of the value group
A	Identifies the media (energy type) to which the metering is related. Non-media related information is handled as abstract data.
B	Generally, identifies the measurement channel number, i.e. the number of the input of a metering equipment having several inputs for the measurement of energy of the same or different types (for example in data concentrators, registration units). Data from different sources can thus be identified. It may also identify the communication channel, and in some cases it may identify other elements. The definitions for this value group are independent from the value
C	Identifies abstract or physical data items related to the information source concerned, for example current, voltage, power, volume, temperature. The definitions depend on the value in the value group A. Further processing, classification and storage methods are defined by value groups D, E and F. For abstract data, value groups D to F provide
D	Identifies types, or the result of the processing of physical quantities identified by values in value groups A and C, according to various specific algorithms. The algorithms can deliver energy and demand quantities as well as other
E	Identifies further processing or classification of quantities identified by values in value groups A to D.
F	Identifies historical values of data, identified by values in value groups A to E, according to different billing periods. Where this is not relevant, this value group can be used for further classification.

Table 4: OBIS code structure and use of value groups

Meaning and range of values is as follows:

- group A: Medium or energy type (0-15)
- group B: Channel (0-64)
- group C: Quantity (0-255)
- group D: Processing of physical quantities (0-255)
- group E: Classification, e.g., tarification (0-255)
- group F: Historical values (0-126)

Example of an OBIS code is depicted at Figure 8. The code 1.1.1.8.2.255 represents a simple meter of electricity (A=1) on channel 1 (B=1), where the measured quantity is $\sum L_i$ Active power+ (C=1) using time integral 1 (D=8), rate is 2 (E=2) and the current value is 255 (F).

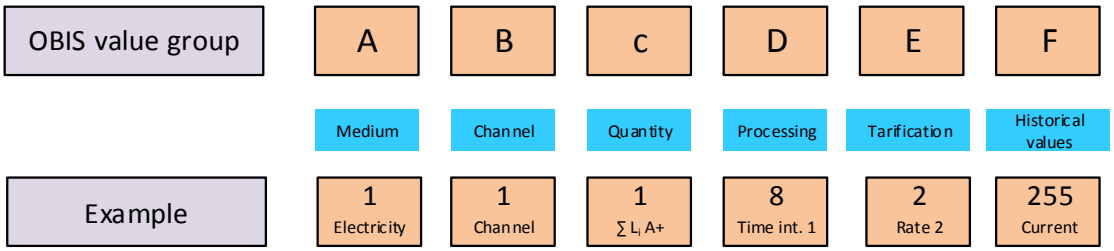


Figure 8: OBIS data identification system

If any of these value groups contain a value in the manufacturer specific range, then the whole OBIS code shall be considered as manufacturer specific.

A list of standard OBIS codes and COSEM objects are regularly maintained by DMLS UA and it is freely available at DLMS portal[4].

Table 5 shows OBIS codes of useful objects of class A that can be found on common DLMS devices:

OBIS code	Description
0.0.42.0.0.255	COSEM logical device name
0.0.44.0.0.255	URL for firmware upgrade
0.0.1.0.0.255	Clock
0.0.40.0.0.255	LN associations
0.0.40.0.1.255	LN associations public client
0.0.41.0.0.255	SAP assignment
0.0.13.0.0.255	Activity calendar
0.0.25.0.0.255	TCP setup
0.0.25.1.0.255	IPv4 setup
0.0.25.2.0.255	MAC setup
0.0.25.3.0.255	PPP setup
0.0.25.4.0.255	GPRS modem setup
0.0.25.6.0.255	GSM diagnostic
0.0.22.0.0.255	HDLC setup
0.0.96.1.0.255	Manufacturing number
0.0.99.12.0.255	Connection profile
0.0.99.98.0.255	Event Log
0.1.2.0.0.255	Modem configuration
0.0.0.1.1.255	Number of available billing periods objects.

Table 5: OBIS codes of common objects

[4] See http://www.dlms.com/documentation/listofstandardobiscodesandmaintenanceproces/index.html [last accessed in January 2018]

4 DLMS/COSEM Communications Framework

DLMS/COSEM (IEC 62056-53, IEC 62056-62) is a standard specification using COSEM for interface modeling equipment and DLMS for data exchange of such metering equipment. It comprises the object model, the application layer protocol and the communication profiles to transport the messages.

In ISO OSI model DLMS communicates over L4-L5 (transport and session layer), COSEM forms presentation layer (L6), see Table 6.

Layer	Function	DLMS/COSEM
Application	Network process to application	Application
Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data	COSEM
Session	Interhost communication, managing sessions between applications	DLMS
Transport	End-to-end connections, reliability and flow control	DLMS
Network	Path determination and logical addressing	DLMS
Data link	Physical addressing	HDLC, IEC 62056-47
Physical	Media, signal and binary transmission	Serial media, cable, radio

Table 6: DLMS and ISO OSI model

The DLMS/COSEM specification specifies a data model and communication protocols for data exchange with metering equipment. It follows a three-step approach as illustrated in Figure 9 [2]:

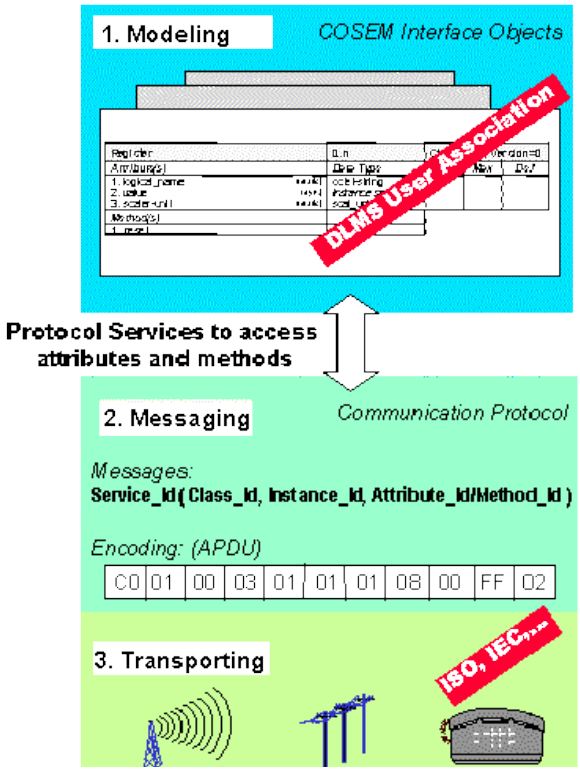


Figure 9: DLMS/COSEM model

1. *Modelling* (Data model) covers the data model of metering equipment as well as rules for identification. The data model provides a view of the functionality of the meter, as it is available at its interfaces. It uses generic building blocks to model this functionality. The model does not cover internal, implementation-specific issues.
It specifies COSEM internal classes (ICs), the object identification system (OBIS), and the use of interface objects for modelling various functions of the metering equipment.
2. *Messaging* covers the communication services and protocols for mapping the elements of the data model to application protocol data units (APDU).
3. *Transporting* covers the services and protocols for the transportation of the messages through the communication channel.

In DLMS/COSEM, clients always use data communication services with logical names referencing. The server may use either services with logical name referencing or with short name referencing.

For LN referencing, DLSM/COSEM defines following client/server services:

- GET service – retrieves attributes of COSEM interface objects
- SET service – modifies attributes of COSEM interface objects
- ACTION service – invokes methods of COSEM interface objects
- EventNotification service – using this service, the server is able to send an unsolicited notification of the occurrence of an event to the client.

For SN referencing, DLMS/COSEM defines following services:

- Read
- Write
- Unconfirmed Write operations

4.1 Communication Profiles

Data exchange between data collection systems and metering equipment using the COSEM interface object model is based on the client/server paradigm. Metering equipment plays the role of the server. The data collection application and the metering application are modelled as one or more application processes (APs). Therefore, in this environment communication takes place always between a client and a server AP: the client AP requests services and the server AP provides them. A client AP may be able to exchange data with a single or with multiple server APs at the same time. A server AP may be able to exchange data with one or more client APs at the same time.

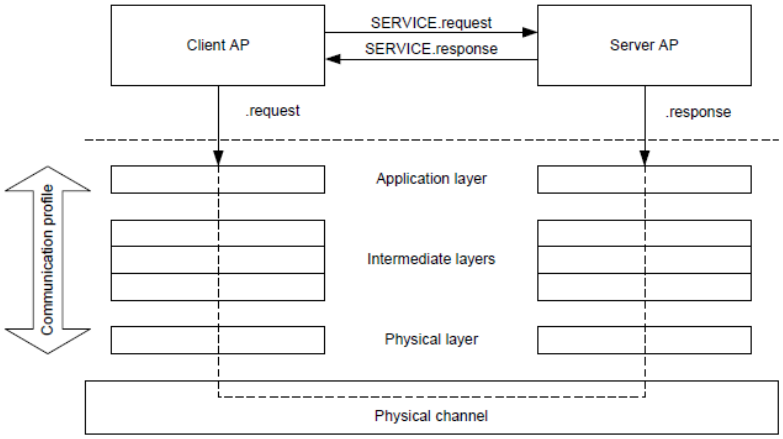


Figure 10: Client/server relationship and protocols

A given set of protocol layers with the COSEM Application Layer (AL) on top constitutes a DLMS/COSEM communication profile. Each profile is characterized by the protocol layers included, their parameters, and by the type of the Application Control Service Element (ACSE) – connection-oriented or connectionless – included in the AL. A single device may support more than one communication profiles, to allow data exchange using various communication media.

DLMS specifies two communication profiles [5]:

- HDLC based profile
  - The 3-layer, connection-oriented HDLC based profile includes three layers: the physical layer (serial connection), HDLC layer and the application layer. It supports data exchange via a local optical or electrical port according to IEC 62056-21, leased lines and the PSTN or the GSM telephone network.
  - The client HDLC address (also called MAC address) is a byte value, e.g., 16 for public clients. The server MAC address is divided into two parts: the upper part is the logical device address, and the lower part is the physical device address. In some cases (e.g., point-to-point topology), the lower part can be omitted. The length of the server address is 1 byte (just an upper address), 2 bytes (1 byte for the upper address and 1 byte for the lower address), or 4 bytes (2 bytes for the upper address and 2 bytes for the lower address).
  - HDLC communication and encapsulation is in Appendix H.
- TCP-UDP/IP based profile
  - This profile supports data exchange via the Internet over various physical media, like Ethernet, ISDN, GPRS, PSTN, or GSM using PPP etc. In these profiles, the COSEM AL is supported by the COSEM transport layers(s), comprising a wrapper and the Internet TCP or UDP protocol, see Figure 11.

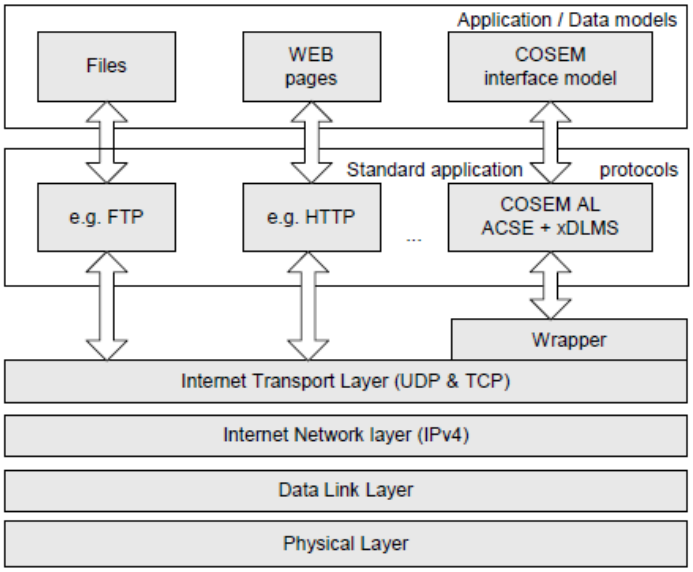


Figure 11: COSEM over IPv4

  - The main function of the COSEM wrapper is to adapt the OSI-style services set provided by COSEM transport layer to UDP/TCP function calls.

For DLMS/COSEM, IANA registers port numbers 4059/TCP and 4059/UDP.

The data exchange uses the client-server model. The client sends requests and the server answers with responses. The request can be “read the object 1.0.1.8.0.255” and the answer is “1789.8 kWh”. Before being able to send requests, the client has first to establish a connection with the other side.

4.2 COSEM Application Layer

The COSEM application layer contains three mandatory components both on the client and the server side:

- the Application Control Service Element (ACSE) [10], see Appendix E
  - The task of ASCE is to establish, maintain, and release application associations.
  - Encoding the ACSE AARQ and AARE APDUs are in BER.
- the extended DLMS Application Service Element, xDLMS_ASE
  - The task of the xDLMS_ASE is to provide data transfer services between COSEM application processes. It is based on DLMS standard IEC 61334-4-41. It has been extended for DLMS/COSEM.
  - xDLMS data transfer services are related to attributes and methods of COSEM interface objects [2]. For accessing these attributes and methods, client/server type services are used: the client requests services and the server provides them. There is also an unsolicited non-client/server type service. This service is requested by the server, upon an occurrence of an event, to inform the client of the value of one or more attributes, as though they had been requested by the client. It is an unconfirmed service.
  - Encoding the xDLMS APDUs carrying the data transfer services in A-XDR [4].
- the Control Function (CF)
  - The CF element specifies how the ASO service invoke the appropriate service primitives of the ACSE, the xDLMS_ASE and the services of the supporting layer.

4.3 Data Transfer Services

COSEM defines two distinct service sets – one for logical name (LN) referencing and one for short name (SN) referencing:

- COSEM client/server type data transfer services for *LN referencing* are the following:
  - *GET service*: it is used to read the value of one or more attributes of COSEM objects.
  - *SET service*: it is used to write the value of one or more attributes of COSEM objects.
  - *ACTION service*: it is used to invoke one or more methods of COSEM objects. Invoking methods may imply sending service parameters and returning data.
  - *EventNotification service*: this is a non-client/server type service.
- COSEM client/server type data transfer services for *SN referencing* are the following:
  - *Read service*: it is used to read the value of one or more attributes or to invoke one or more methods of COSEM objects. It is a confirmed service.
  - *Write service*: It is used to write the value of one or more attributes or to invoke one or more methods of COSEM objects. It is a confirmed service.
  - *UnconfirmedWrite service*: it is used to write the value of one or more attributes or invoke one or more methods of COSEM objects. It is an unconfirmed service.
  - *InformationReport service*: this is a non-client/server type service.

COSEM APDU Message identifiers are as follows:

- 192 – GetRequest
- 193 – SetRequest

- 194 – EventNotificationRequest
- 195 – ActionRequest
- 196 – GetResponse
- 197 – SetResponse
- 198 – ActionResponse

The format of DLMS/COSEM APDUs is described in [9] and in Appendix D.

4.4 Application Layer PDUs

COSEM application layer includes the ASCE and xDLMS ASE.

4.4.1 The Association Control Service Element (ACSE) services

The ACSE provides following services to establish and release application associations (AAs):

- AARQ (A-Associate Request)
- AARE (A-Associate Response)
- RLRQ (A-Release Request)
- RLRE (A-Release Response)

The format of ACSE APDUs is described by standard OSI/IEC 8650-1 or ITU X.227 [10], see also Appendix E. The ACSE APDUs are encoded in BER. The *user-information* parameter of the ACSE APDU shall carry the xDMLS InitiateRequest / InitiateResponse / confirmedServiceError APDU as appropriate, encoded in A-XDR, and then encoding the resulting OCTET STRING in BER, see Fig 12.

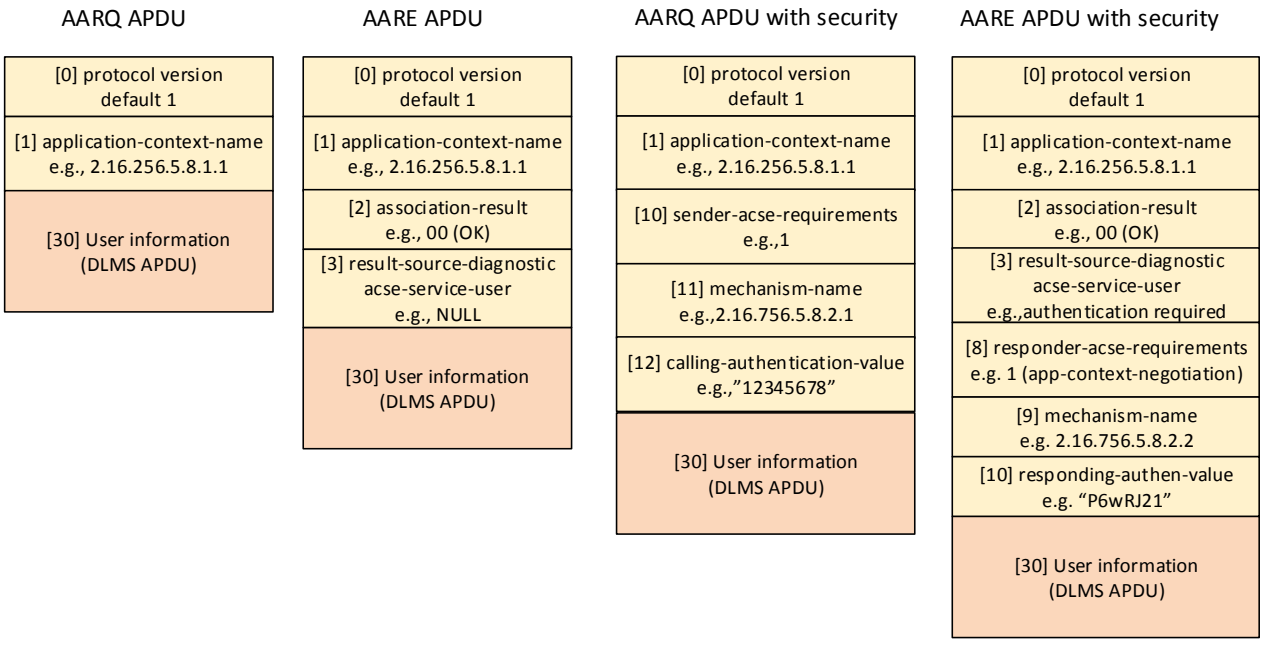


Figure 12: Example of AARQ and AARE APDUs

APDUs in Figure 12 have variable structure depending on the fields that are transmitted. Formally, the format is described by ASN.1 notation as follows. For the full description, see Appendix E.

```
AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE {     --- APPLICATION 0 = 60H = 96
        protocol-version                [0]     IMPLICIT BIT STRING { version 1(0)} DEFAULT {version 1},
        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-ide        [5]     AE-invocation-identifier OPTIONAL,
        calling-AP-title                [6]     AP-title OPTIONAL,
        calling-AE-quantifier           [7]     AE-qualifier OPTIONAL,
        calling-AP-invocation-id        [8]     AP-invocation-identifier OPTIONAL,
        calling-AE-invocation-id        [9]     AE-invocation-identifier OPTIONAL,
        sender-acse-requirements        [10]    IMPLICIT ACSE-requirements OPTIONAL,
        mechanism-name                  [11]    IMPLICIT Mechanism-name OPTIONAL,
        calling-authentication-value    [12]    EXPLICIT Authentication-value OPTIONAL,
        application-context-name-list   [13]    IMPLICIT Application-context-name-list OPTIONAL,
        p-context-definition-list       [14]    Syntactic-context-list OPTIONAL,
        called-asoi-tag                 [15]    IMPLICIT ASOI-tag OPTIONAL,
        calling-asoi-tag                [16]    IMPLICIT ASOI-tag OPTIONAL,
        implementation-information      [29]    IMPLICIT    Implementation-data OPTIONAL,
        user-information                [30]    IMPLICIT    Association-information OPTIONAL
}
```

4.4.2 The xDLMS Assocation Service Element (ASE)

To access attributes and methods of COSEM objects, the services of the xDLMS ASE are used. It provides services to transport date between COSEM APs.

xDLMS (extended DLMS) includes some extension to the DLMS standard IEC 61334-4-41 [7]. These extensions define added functionality. They are made in such a way, that there is no conflict with the existing DLMS standard.

The extensions comprise the following functions [5]:

- additional services- GET, SET, ACTION, and EventNotification;
- additional data types;
- new DLMS version number – the number of the first version of the xDLMS ASE is 6;
- new conformance block – enables optimized DLMS/COSEM server implementations with extended functionality. This block has application data type 31 (Conformance, BIT STRING SIZE (24)), instead of application data type 30 (Conformance, BIT STRING SIZE (16)).
- clarification of the meaning of the PDU size – the Proposed Max PDU Size means now the Client Max Receive PDU Size, the Negotiated Max PDU Size means now the Server Max Receive PDU Size.

IEC 61334-4-41 defines DLMS APDU [7] as follows, see also Appendix C.

```
DLMSpdu ::= CHOICE {
        confirmedServiceRequest          [0]    RequestToConfirmedService,
        initiateRequest                  [1]    RequestToInitiate,
        getStatusRequest                 [2]    RequestToGetStatus,
        getNameListRequest               [3]    RequestToGetNameList,
        getVariableAttributeRequest      [4]    RequestToGetVariableAttribute,
        readRequest                      [5]    RequestToRead,
        writeRequest                     [6]    RequestToWrite,
        confirmedServiceResponse         [7]    ResponseToConfirmedService,
        initiateResponse                 [8]    ResponseToInitiate,
        getStatusResponse                [9]    ResponseToGetStatus,
        getNameListResponse              [10]   ResponseToGetNameList,
        getVariableAttributeResponse     [11]   ResponseToGetVariableAttribute,
        readResponse                     [12]   ResponseToRead,
        writeResponse                    [13]   ResponseToWrite,
        confirmedServiceError            [14]   ErrorConfirmedService,
        unconfirmedServiceRequest        [15]   RequestToUnconfirmedService,
        abortRequest                     [16]   RequestToAbort,
        unconfirmedWriteRequest          [17]   RequestToUnconfirmedWrite,
        unsolicitedServiceRequest        [18]   RequestToUnsolicitedService,
        informationReportRequest         [19]   RequestToInformationReport,
        …
}
```

Examples of encoded DLMS/COSEM PDUs are in Section 5.

4.4.3 Session Establishment

Session establishment and tear-down uses ACSE defined by ITU-T X.227 [10], see Appendix D. Figure 13 shows ACSE messages (blue) and DLMS/COSEM APDUs (black).

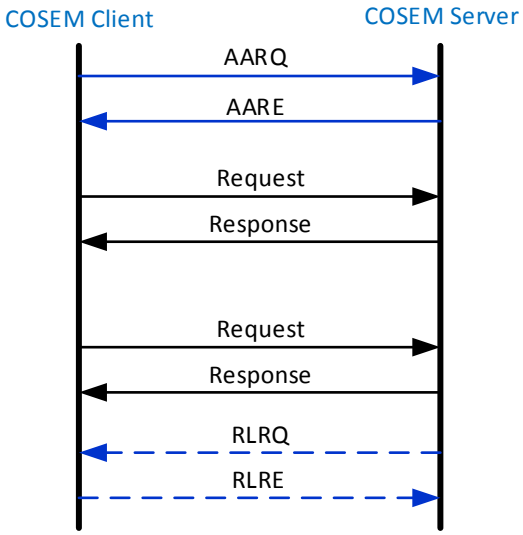


Figure 13: COSEM messages

The ACSE protocol establishes and tears down an Application Association (AA) between a client Application Process (AP) and a server Application Process (AP) for a given Application Context Name. This name identifies whether an encrypted or plaintext application context is being requested and whether short name or long names are used for addressing [11]:

- Long Name Referencing, Without Ciphering (OID = 2.16.756.5.8.1.1)
- Short Name Referencing, Without Ciphering (OID = 2.16.756.5.8.1.2)
- Long Name Referencing, With Ciphering (OID = 2.16.756.5.8.1.3)
- Short Name Referencing, With Ciphering (OID = 2.16.756.5.8.1.4)

4.4.4 Request and Response Messages

After establishing association, protocol DLMS/COSEM can read data, write data or invoke actions. DLMS/COSEM APDU format is also variable and encoded using A-XDR rules. Example of DLMS *Get-Request* and *Get-Response* APDUs is in Figure 14.

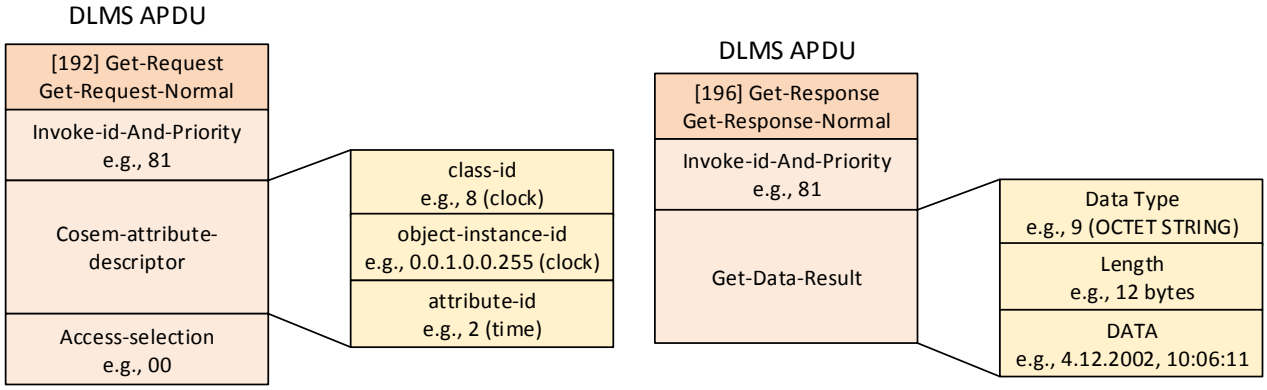


Figure 14: Example of DLMS APDUs

4.5 The Addressing

In the DLMS/COSEM communication, each side of the connection has an address. By definition, the client address is a byte value. The value of the client address determines also the real nature of the client. The standard states that a client with address 16 is a public client. There can be other kind of clients: a data collection system, a manufacturer, a consumer, etc.

The address is composed of the address of the physical device and the address of the logical device.

4.6 The Authentication and the Access Rights

Authentication is a process of establishing the true identity of the communicating partners before requesting and providing data communication services. There are three levels of authentication security defined:

1. Lowest Level Security – neither the client nor the server is identified
2. Low Level Security (LLS) – establishes the true identity of the client by verifying a password. LLS is used when the communication channel provides adequate security to avoid eavesdropping and message replay.
3. High Level Security (HLS) – establishes the true identity of both the client and the server. HLS authentication is typically used when the communication channel offers no intrinsic security and precautions have to be taken against eavesdroppers and against message replay.

The authentication method is indicated by the a*uthentication-mechanism-name* attribute represented by OID in AARQ message. The standard specifies the following OID values [11]:

- 2.16.756.5.8.2.0 – Lowest Level Security
- 2.16.756.5.8.2.1 – Low Level Security (LLS)
- 2.16.756.5.8.2.2 – High Level Security (HLS) – Vendor Proprietary
- 2.16.756.5.8.2.3 – High Level Security (HLS) – MD5
- 2.16.756.5.8.2.4 – High Level Security (HLS) – SHA1
- 2.16.756.5.8.2.5 – High Level Security (HLS) - GMAC

There are several mechanisms how to control the access to the COSEM objects. The simplest one is based on the client address. Based on the client address, only the objects that are allowed to be read or written can be accessed. Usually, non-public clients have more privileges. Nevertheless, a public client is always authorized to access the logical management device.

5 Examples of encoded DLMS/COSEM PDUs

The following examples are taken from [5], [7] and [9]. They demonstrate how COSEM and DLMS APDUs are described using ASN.1 notation and encoded using BER (Appendix F) and A-XDR encoding [4] for transmission. Proper understanding of the encoding is essential for implementation of the parser.

Example 1: Encoding xDLMS-Initiate.request PDU in xDLMS (Extended DLMS)

This type of DLMS APDU is usually carried in AARQ ACSE. The format of the APDU is as follows:

```
xDLMS-Initiate.request :: = SEQUENCE{
        dedicated-key                   OCTET STRING OPTIONAL,
        response-allowed                BOOLEAN DEFAULT TRUE,
        proposed-quality-of-service     [0] IMPLICIT Integer8 OPTIONAL,
        proposed-dlms-version-number    Unsigned8,
        proposed-conformance            Conformance,
        client-max-received-pdu-size    Unsigned16 (different interpretation for xDLMS)
}
```

Knowing APDU format we are able to encode and decode APDUs of this type. Suppose the client with the following values: no ciphering used, response-allowed=TRUE, no proposed-quality-of-service, dlms version set to 6, the proposed-conformance is 00 7E 1F for LN and 1C 03 20 for SN and the client PDU is 1200 (0x04B0).

The A-XDR encoding of of such APDU is 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0 for LN referencing is as follows:

- 01 – explicit tag of the APDU SEQUENCE InitiateRequest
- 00 – the dedicated-key (not present => FALSE (00); present => TRUE (01) followed by the OCTET STRING)
- 00 – the response-allowed (00=FALSE, FF=TRUE)
- 00 – the proposed-quality-of-service (not present => FALSE (00))
- 06 – the proposed-dlms-version-number (unsigned8, value 6). Version 6 means xDLMS with longer Conformance block
- 5F 1F 04 00 00 7E 1F – the conformance block
    - As specified in IEC 61334-6, Annex C, the proposed-conformance element of InitiateRequest PDU is encoded by BER, e.g, represented by TLV format (type-length-value) [8], see Appendix F.
    - Type 5E 1F is 0101 1111 0001 1111 binary. The first byte (5F) is called an identifier octet with the following meaning: 01 (application class), 0 (primitive form) and 11111 (31) means application tag no. 31. This is defined by xDLMS as Conformance data type [5] represented by the standard data type BIT STRING of size 24, e.g., three

bytes. This differs to DLMS where Conformance data has application tag 30 and BIT STRING SIZE (16) [7]. The second byte (1F) is not used[5].
    - 04 gives the length of the value, e.g., four bytes.
    - The first byte (00) of BIT STRING represents the number of bits left unused (see Appendix F). The value zero means there are not unused bits in the following octets and all the bits are important.
    - The value of the bit string is 00 7E 1F (in hex) for LN referencing.
- 04 B0 – the proposed-max-pdu-size (Unsigned16, value 0x4B0)

A-XDR encoding of the APDU for SN referencing is 01 00 00 00 06 5F 1F 04 00 1C 03 02 04 B0.

Example 2: Encoding InitiateRequest PDU

Suppose DLMS InitiateRequest PDU (not xDLMS) with the following values: no dedicated key, response-allowed FALSE, proposed QoS set to 4, DLMS version 1, maximum PDU size set to 134 (0x86) and conformance value set to 0x1C00.

```
RequestToInitiate :: = SEQUENCE {
        dedicated-key                   OCTET STRING OPTIONAL,
        response-allowed                BOOLEAN DEFAULT TRUE,
        proposed-quality-of-service     [0] IMPLICIT Integer8 OPTIONAL,
        proposed-dlms-version-number    Unsigned8,
        proposed-conformance            Conformance,
        proposed-max-pdu-size           Unsigned16
}
```

This PDU will be encoded by A-XDR as 01 00 00 01 04 01 5E 03 00 10 C3 00 86 as follows:

- 01 – explicit tag of the APDU SEQUENCE InitiateRequest
- 00 – the dedicated-key (FALSE, not present)
- 00 – the response-allowed FALSE
- 00 – the proposed-quality-of-service (TRUE, is present)
- 04 – the value of QoS (Integer8)
- 01 – the proposed-dlms-version (value 1)
- 5E 03 00 10 C3 – the conformance block
    - 5E (0101 1110) is the octet identifier. 01 means application class, primitive form (0) where application tag 11110 (30) means Conformance data type BIT STRING (SIZE(16)).
    - the length of 03 bytes
    - 00 10 C3 is BIT STRING with the number of unused bits 00. The value of the bit string is 10 C3 (two bytes).
- 00 86 – max PDU size (134 bytes in decimal)

[5] For compliance with existing implementations, encoding of the Application 31 tag on one byte (5F) instead of two bytes (5F 1F) is accepted when the 3-layer, connection-oriented, HDLS based profile is used.

Example 3: Encoding xDLMS-Initiate.response APDU (extended DLMS)

This type of APDU is usually carried in AARE ACSE. The format of this APDU is as follows:

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

Suppose the server with the following values: negotiated-quality-of-service not present, version number set to 6, max received PDU 500 (x001F4). The negotiated conformance is 0x00 50 1F for LN referencing and 0x1C 03 20 for SN referencing.

A-XDR encoding of the APDU is 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 for LN:

- 08 – explicit tag of the APDU SEQUENCE InitiateResponse.
- 00 – the negotiated-quality-of-service (FALSE, not present)
- 06 – dlms version (Unsigned8). This means xDLMS.
- 5F 1F 04 00 00 50 1F – a BER encoded part. Application tag no. 31 means Conformance data type [], e.g. BIT STRING (SIZE(24). The length of the value is 04 bytes. The number of unused bits is 0 (00) and the value is 00 50 1F for LN referencing.
- 01 F4 – the server-max-receive-pdu-size (unsigned16), e.g., 500 (decimal)
- 00 07 - the vaa-name component (objectname, integer16), value 0x0007 for LN and 0xFA00 for SN referencing

A-XDR encoding of the APDU for SN is 08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00.

Example 4: Encoding InitiateResponse APDU

Another example of DLMS APDU InitiateResponse has the values as in Example 2 and the vaa-name set to 0x0037.

```
ResponseToInitiate ::= SEQUENCE {
        negotiated-quality-of-service   IMPLICIT Integer8 OPTIONAL,
        negotiated-dlms-version-number          Unsigned8,
        negotiated-conformance                  Conformance,
        server-max-receive-pdu-size     Unsigned16,
        vaa-name                        ObjectName
}
```

A-XDR encoding of the APDU is 08 01 04 01 5E 03 00 1C 00 00 86 00 37.

- 08 – explicit tag of the APDU SEQUENCE InitiateResponse.
- 01 – the negotiated-quality-of-service (TRUE, is present)
- 04 – the value of QoS (Unsigned8)
- 01 – dlms version (Unsigned8)

- 5E 03 00 1C 00 – a BER encoded part.
  - 5E is the application tag (30) referring to Conformance data type which is BIT STRING SIZE (24).
  - 03 is the length of the value
  - 00 is the number of unused bits (no unused bits).
  - 1C 00 is the conformance value.
- 00 86 – max PDU size (Unsigned16), e.g., 134 in decimal.
- 00 37 – the vaa-name (objectname). Object name is Integer16.

Example 5: Encoding the AARQ APDU without the ACSE security

For ACSE use, suppose that the client would like to establish the application association with the following context: default protocol version, application context name 2.16.756.5.8.1.1 (joint-iso-ccitt(2).counter(16).country-name(756).identified-organization(5).DLMS-UA(8).application-context(1).context_id(1)) for LN referencing or 2.16.756.5.8.1.2 for SN referencing, no authentication used and no implementation-information included.

The format of AARQ APDU is as follows (see Appendix D):

```
AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE {          --- APPLICATION 0 = 60H = 96
        protocol-version              [0]    IMPLICIT BITSTRING {version 1(0)} DEFAULT {version 1},
        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-ide      [5]    AE-invocation-identifier OPTIONAL,
        calling-AP-title              [6]    AP-title OPTIONAL,
        calling-AE-quantifier         [7]    AE-qualifier OPTIONAL,
        calling-AP-invocation-id      [8]    AP-invocation-identifier OPTIONAL,
        calling-AE-invocation-id      [9]    AE-invocation-identifier OPTIONAL,
        sender-acse-requirements      [10]   IMPLICIT ACSE-requirements OPTIONAL,
        mechanism-name                [11]   IMPLICIT Mechanism-name OPTIONAL,
        calling-authentication-value  [12]   EXPLICIT Authentication-value OPTIONAL,
        implementation-information    [29]   IMPLICIT  Implementation-data OPTIONAL,
        user-information              [30]   IMPLICIT  Association-information OPTIONAL
}
```

The AARQ APDU is encoded using BER encoding as follows: 60 1D A1 09 06 07 60 85 74 05 08 01 01 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0 for LN referencing.

- 60 – the first byte (0110 0000) means application class (01) and constructed form (1) with application tag 0 which means AARQ APDU. The constructed type used here is the SEQUENCE.
- 1D represents the length of the message, e.g. 29 bytes.
- A1 (1010 0001) means context-specific class (10), constructed form (1), with context-specific tag 1 which means the *application-context-name* (see AARQ-apdu).
- 09 is the length of the value, e.g, 9 bytes.

- The value 06 07 60 85 74 05 08 01 01 is also a TLV triplet (see constructed type above).
    - Type 06 (0000 0110) denotes the universal class (00) and primitive form (0). The tag 6 (00110) means the *object identifier* (OID).
    - 07 is the length of the OID value, e.g, 7 bytes.
    - The value 60 85 74 05 08 01 01 is decoded as follows (see Appendix F):
        - Byte 60 (96 in decimal) decodes two bytes (in decimal): 2 and 16.
        - In decimal format, the value is 2.16.756.5.8.1.1 because bytes 85 (1000 0101) and 74 (0111 0100) represent one integer, e.g., 0010 1111 0100 which is 756 in decimal.
- The value BE is the start of a new TLV triplet.
    - BE (1011 1110) denotes the context-specific class (10) with constructed form (1). The application tag 11110 (30) refers to the *user-information* (see above). This field is of type Association-information which is OCTET STRING (see Appendix D).
    - The length of the value is 10 in hexadecimal which is 16 bytes.
    - Since it is a constructed type (1), the value embeds another TLV triplet:
        - Type 04 (0000 0100) means universal class (00), primitive form (0) with data tag 4 (00100) which means OCTET STRING (see Appendix D). However, for COSEM, this field contains the A-XDR encoded DLMS-Initiate-request/response PDUs.
        - The length of data is 0E, e.g., 14 bytes.
        - The value is 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0. This is DLMS-Initiate-request PDU. Decoding of this PDU is at Example 1.

Example 6: Encoding the AARQ APDU using low level authentication

This example demonstrates the encoding the AARQ APDU with authentication. In this example, the *sender-acse-requirements* option indicates that the ACSE functional unit is selected. The *mechanism-name* is set to default-COSEM-low-lever-security-mechanism with OID 2.16.756.5.8.2.1. The *calling-authentication-value* contains GraphicString with the password "12345678". The packet is an extension of APDU from example 5.

The BER-encoded APDU is as follows: 60 36 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 31 32 33 34 35 36 37 38 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0.

- 60 denotes AARQ APDU, constructed application type which is SEQUENCE.
- 36 represents the length, e.g., 54 bytes.
    - A1 is an embedded TLV triplet, with context-specific class, primitive form and tag 1 which means *application-context-name* with data of length 9 bytes. The value 06 07 60 85 74 05 08 01 01 is described in Example 5 – OID with value 2.16.756.5.8.1.1.
    - 8A (1000 1010) starts a new TLV triplet. The context-specific class with primitive form and tag 10 (01010) indicates the *sender-acse-requirements.* According to the standard (see Appendix D), the ACSE-requirements is BIT STRING.
        - The length of value is 2 bytes.

        - Value 07 80 includes number of unused bits (07) and the bit string 1000 0000 (0x80) where the last 7 bits are unused. Thus, the BIT STRING value is 1 which indicates the requirement of the authentication functional unit.
    - 8B (1000 1011) starts a new TLV triplet with context-specific class, primitive form and context-specific tag 11 which is the *mechanism-name*. The standard says that the *mechanism-name* is OBJECT IDENTIFIER (see Appendix D).
        - The length of the value is 7 bytes.
        - The value 60 85 74 05 08 02 01 (OBJECT IDENTIFIER) is BER-encoded value 2.16.756.5.8.2.1 (see Appendix F).
    - AC (1010 1100) is a new TLV triplet with context-specific class (10), constructed form (1) and context-specific tag 12 which corresponds to the *calling-authentication-value.* The data type of this value is CHOICE, see Appendix D.
        - The length is 0A which is 10 bytes.
        - The value 80 08 31 32 33 34 35 36 37 38 is an embedded TLV value.
            - 80 (1000 0000) means the context-specific class, primitive form and data tag 0. This corresponds to the *charstring* which is GraphicString.
            - The length of the value is 8 bytes.
            - The value 31 32 33 34 35 36 37 38 is a sequence of ASCII characters which is a string “12345678”.
    - The last sequence BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0 is the *user-information* containing *DLMS-Initiate-request* PDU, see Example 5.

Example 7: Encoding the AARQ APDU using high-level authentication

Similarly to the previous example, the APDU contains the same *mechanism-name*. The *calling-authentication-value* assumes that no client-to-server challenge is requested.

The BER-encoded APDU is as follows: 60 2E A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 02 AC 02 80 00 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0.

- 60 2E denotes AARQ APDU with length 46 bytes. It is a constructed form (SEQUENCE).
- The value A1 09 06 07 60 85 74 05 08 01 01 contains the *application-context-name* of the length 9 bytes and with value 2.16.756.5.8.1.1, see Example 6.
- Sequence 8A 02 07 80 encodes the *sender-acse-requirements* with value 1, see Example 6.
- Sequence 8B 07 60 85 74 05 08 02 02 encodes the *mechanism-name* (context-specific tag 11) where the value with length 7 is OID with value 2.16.756.5.8.2.2 which means vendor proprietary high-level security.
- AC (1010 1100) starts a new TLV triplet which denotes context-specific class with constructed form and context-specific tag 12 which means the *calling-authentication-value*, see Appendix D.
    - The length of the value is 2 bytes.
    - Authentication value is of data type CHOICE. Value 0x80 (1000 0000) denotes *charstring* with the length 0 bytes which means, there is no password present.
- The remaining data BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0 is *DLSM-Initiate-request* PDU, see Example 5.

Example 8: Encoding the AARE APDU not using security or using low level security

This example describes APDU send by the server that accepts the proposed AA within the following application context: protocol-version is the default ACSE version, application-context-name is 2.16.256.5.8.1.1 for LN referencing, no additional authentication is used and no implementation-information is included.

The format of APDU is as follows, see Appendix D:

```
AARE-apdu ::= [APPLICATION 1] IMPLICIT SEQUENCE {          --- APPLICATION 1 = 61H = 97
        protocol-version                [0]     IMPLICIT BIT STRING {version 1(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,
        responder-acse-requirements     [8]     IMPLICIT ACSE-requirement OPTIONAL,
        mechanism-name                  [9]     IMPLICIT Mechanism-name OPTIONAL,
        responding-authentication-value [10]    EXPLICIT Authentication-value OPTIONAL,
        application-context-name-list   [11]    IMPLICIT Application-context-name-list OPTIONAL,
        implementation-information      [29]    IMPLICIT Implementation-data OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}
```

The BER-encoded APDU is 61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07.

- Type 61 (0110 0001) describes application-specific class (01), constructed form (1) and application tag (01) corresponds to AARE APDU, see Appendix D.
- The length of the value is 41 bytes (29 in hexadecimal).
    - The value is an embedded BER-encoded triplet with type A1 (1010 0001) which means context-specific class (10), constructed form (1) and context-specific tag 1 which is the *application-context-name*, see above.
    - The length is 9 bytes and value is another TLV triplet.
        - The type 06 (0000 0110) is universal class (0), primitive form (0) and data type 6 which means OBJECT IDENTIFIER.
        - The length of the OID is 7 bytes and value 60 85 74 05 08 01 01 is 2.16.756.5.8.1.1
    - Another TLV triplet starts with A2 (1010 0010) which is also context-specific class with constructed form (1) and context-specific tag 2 which means the *result.* According to the standard, the *Association-result* is INTEGER.
        - The length is 3 bytes and value 02 01 00 is a TLV triplet of type 02 (universal class, primitive form, data type INTEGER).
            - The length is 1 byte and value is 00 which means *accepted*, see Appendix D.

- The following TLV triplet is A3 05 A1 03 02 01 00 contains type A3 (1010 0011) corresponds to the context-specific class (1), constructed form (1) and the data type is *result-source-diagnostic* (context-specific tag 3).
  - The length of the embedded triplet is 5 bytes. The type A1 (1010 0001) is of context-specific class (10), constructed form (1) and context-specific tag 1 which means the *acse-service-user* (CHOICE 1).
    - The length is 3 bytes and the value 02 01 00 which is another TLV triplet.
    - This triplet has type 02 which means universal class, primitive form and data type INTEGER (02). The length of the value is 1 byte and the value is 0. This value means no diagnostic provided (value NULL).
- The last TLV sequence BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 is of type BE (1011 1110) which denotes context-specific class (10), constructed form (1) and data type of tag 30 which denotes the *user-information* field.
  - The length of the field is 16 bytes (10 in hexadecimal). Its value is another TLV triplet of type 04 (0000 1110) OCTET STRING with the length 14 bytes (E in hexadecimal).
    - The sequence 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 is a A-XDR encoded xDLMS-Initiated.response APDU, see Example 3.

Example 9: Encoding the AARE APDU using high level security

This example is similar to Example 8 with the following differences: three additional fields are added (*responder-acse-requirements, mechanism-name*, and *responding-authentication-value*). In addition, the code in *result-source-diagnostic* is set to *authentication-required* to reflect the additional authentication step needed.

The BER-encoded APDU is sequence 61 41 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 0E 88 02 07 80 89 07 60 85 74 05 08 02 02 AA 09 80 07 50 36 77 52 4A 32 31 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 for LN referencing:

- Type 61 (0110 0001) denotes the application-specific class (10), constructed form (1) and data type with application-specific tag 1 which is AARE APDU.
  - The length of the value is 65 bytes (41 in hexadecimal).
  - The value an embedded TLV triplet with type A1 (*application-context-name)*, length 9 bytes and a value that represent OID 2.16.756.5.8.1.1, see Example 8.
  - The next TLV is of type A2 and length 3 bytes. The value represents the *association-result* with embedded TLV that mans *accepted,* see Example 8.
  - The following TLV triplet of type A3 (*result-source-diagnostic*) is of length 5 bytes. It is in the constructed form, thus, the value is another TLV triplet.
    - This TLV has type A1 which denotes the *acse-service-user* (tag 1). The length is 3 bytes and value 02 01 0E encodes INTEGER (02) of 1 byte and value 14 (E) which means *authentication-required*.

- The next TLV is of type 88 (1000 1000) which is the context-specific class (10), and the primitive form. The tag 8 denotes the *responder-acse-requirements* which is BITSTRING.
  - The length of the value is 2 bytes. The value 07 80 means that there are 7 unused bits in the last octet, e.g., the value is 1 which means *application-context-negotiation.*
- The next TLV has type 89 (1000 1001) which denotes *mechanism name* (tag 09) which is OBJECT IDENTIFIER. The value is 7 bytes long and is 60 85 74 05 08 02 02 which means 2.16.756.5.8.2.2 (default-COSEM-high-level-security-mechanism-name).
- The next TLV has type AA (1010 1010) which is the context-specific class (10), constructed form (1) and context-specific tag 10 which means the *responding-authentication-value.* Its length is 9 bytes.
  - The value is a triplet with type 80 (1000 0000) which denotes the context-specific class (10), primitive form (0) and data type 0 (charstring) which is *GraphicString.*
    - The length of the string is 7 bytes.
    - The value is 50 36 77 52 4A 32 31 is in ASCII code "P6wRJ21".
- The last sequence BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 is decoded in Example 8, see above.

Example 10: Encoding the AARE APDU case of failure 1

This example shows the construction of an AARE APDU, when the server is not able to accept the proposed association because of the received *application-context-name* does not fit the application context which is supported by the server. In this case, the *result* field of the AARE PDU shall contain the *rejected-permanent* value, the *result-source-diagnostic* field with *application-context-name-not-supported* value. Suppose that the server is able to support the proposed xDLMS context, i.e., the *user-information* field shall contain a correctly constructed xDLMS-Initiate.response PDU.

The BER-encoded APDU is 61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 01 A3 05 A1 03 02 01 02 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07.

- Type 61 indicates AARE APDU of length 41 bytes (0x29).
  - The value is another TLV structure with type A1 which means the *application-context-name.* Its length is 9 bytes and the value represents OID 2.16.756.5.8.1.1.
  - The next TLV is of type A2 (1010 0010) which means the constructed form and data type with tag 2, i.d., *association result.* The length is 3 bytes.
    - The value is an embedded TLV structure with type 02 which encodes 1-byte INTEGER value 1 that means *rejected-permanent.*
  - The next TLV is of type A3 which represents *result-source-diagnostic* value with length 5 bytes. It is the constructed type, so the value is another TLV.
    - The type of embedded TLV is A1 which is *acse-service-user* type with length 3 bytes and value 02 01 02.

- The value is 1-byte INTEGER with value 2 which means *application-context-name-not-supported.*

- The last sequence is BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 which is decoded in Example 8, see above.

Example 11: Encoding the AARE APDU case of failure 2

This example shows the construction of an AARE APDU when the serve is not able to accept the proposed AA because the proposed xDLMS context cannot be supported by the server. The proposed COSEM application context could be accepted.

In this case, the *result* field of the AARE APDU shall contain the *rejected-permanent* value, the *result-source-diagnostic* field has the *no-reason-given* value, and the *user-information* field shall contain a correctly constructed DLMS-ConfirmedServiceError PDU.

The DLMS-ConfirmedServiceError has application tag 14 and format as follows:

```
confirmedServiceError ::= CHOICE {
        --- tag 0 is reserved
        initiateError                   [1]     ServiceError,
        getStatus                       [2]     ServiceError,
        getNameList                     [3]     ServiceError,
        getVariableAttribute            [4]     ServiceError,
        read                            [5]     ServiceError,
        write                           [6]     ServiceError,
        getDataSetAttribute             [7]     ServiceError,
        getTIAttribute                  [8]     ServiceError,
        changeScope                     [9]     ServiceError,
        start                           [10]    ServiceError,
        stop                            [11]    ServiceError,
        resume                          [12]    ServiceError,
        makeUsable                      [13]    ServiceError,
        initiateLoad                    [14]    ServiceError,
        loadSegment                     [15]    ServiceError,
        terminateLoad                   [16]    ServiceError,
        initiateUpLoad                  [17]    ServiceError,
        upLoadSegment                   [18]    ServiceError,
        terminateUpLoad                 [19]    ServiceError,
}
ServiceError ::= CHOICE {
        initiate    [6]         IMPLICIT ENUMERATED {
                    other                           (0),
                    DLMS-version-too-low            (1),
                    incompatible-conformance        (2),
                    PDU-size-too-short              (3),
                    refused-by-the-VDE-Handler      (4)
        }
}
```

The APDU is 61 1F A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 01 A3 05 A1 03 02 01 01 BE 06 04 04 0E 01 06 01.

- Type 61 means AARE APDU with length 31 bytes (0x1F).
  - The value is an embedded TLV structure with type A1 (*application-context-name)* and the length 9 bytes.
    - The value represents OID 2.16.756.5.8.1.1.
  - The next TLV is of type A2 (*association-result)* and length of 3 bytes.
    - The embedded TLV is 1-byte INTEGER with value 1 which means *rejected-permanent.*
  - The next TLV has type A3 (*result-source-diagnostic)* and length of 5 bytes.
    - The value A1 03 02 01 01 is another TLV.
      - Type A1 means *acse-service-user* with length 3 bytes.
        - The embedded TLV is 1-byte INTEGER with value 1 which means *no-reason-given.*
  - The last TLV is sequence BE 06 04 04 0E 01 06 01.
    - BE (1011 1110) is the context-specific class (10) with constructed form (1) and context-specific tag is 30 which means *user-information.* The length of the value is 6 bytes. The value is another TLV structure.
      - The type of the TLV is 04 which means OCTET STRING, the length is 4 bytes and value 0E 01 06 01 represents an A-XDR-encoded DLMS message.
        - The type of DLMS message is 14 (0x0E) which means *confirmedServiceError.*
        - Value 1 is the tag for selecting ConfirmedServiceError which is *initiateError*.
        - Value 6 is the tag of ServiceError CHOICE which means *initiate.*
        - Value 1 encodes the enumerated reason of failure with the meaning *DLMS-version-too-low.*

Example 12: Reading the clock[6]

This example shows how to read the clock using DLMS. Data exchanges is composed of two pairs of packets: the *Get-Request* APDU requesting logical name of the clock of the meter, the *Get-Response* APDU on this request, the *Get-Request* asking for the time value of the meter, and the *Get-Response* with the requested date and time. All DLMS APDUs are encoded by A-XDR encoding, see Appendix G. The APDU format is as follows, see also Appendix D:

[6] The following examples are taken from http://www.dlms.com/faqanswers/questionsonthedlmscosemspecification/areexamplesforinformationexchangeavailable.php (Feb 2018)

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

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

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

Cosem-Class-Id ::= Unsigned16

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

Cosem-Object-Attribute-Id ::= Integer8

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

The first DLMS APDU (request) is C0 01 81 00 08 00 00 01 00 00 FF 01 00.

- Type C0 (192 in decimal) means the *get-request*. It is CHOICE data structure.
  - 01 means Get-Request-Normal which is a sequence.
    - 81 (1000 0001) is the *Invoke-id-and-priority* where 100 (4 in decimal) is an invoke ID, 00 are reserved, 0 (service class) means confirmed and 1 (priority) means high priority.
    - 00 08 00 00 01 00 00 FF 01 00 is Cosem-Attribute-Descriptor (SEQUENCE) where
      - 0008 is a Cosem Interface Class ID which means the clock, see App. A.
      - 00 00 01 00 00 FF (object instance ID) is a logical name in the form of OBIS code, e.g. 0.0.1.0.0.255 which is the clock object.
      - 01 means to ask for the attribute no.1 which is the logical name
      - 00 denotes an unused *access-selection* item.

The answer on this request is C4 01 81 00 09 06 00 00 01 00 00 FF.

- Type C4 (196 in decimal) means the *get-response.*
  - 01 means *Get-Response-Normal* which is SEQUENCE where
    - 81 (1000 0001) is the *Invoke-id-and-priority* (invoke id = 4, confirmed service, high priority)
    - 00 means the *Get-Data-Result*
    - 09 means OCTET STRING
    - 06 means the length
    - The value is 00 00 01 00 00 FF which is OBIS name 0.0.1.0.0.255.

The next APDU requests the second attribute of the clock which is time (OCTET STRING). *date_time* is octet-string(12) with the following interpretation [3]:

```
date_time octet-string {        -- 0xFF value means not specified (in any item)
        year highbyte           -- interpreted as long
        year lowbyte
        month
        day of month
        day of week
        hour
        minute
        second
        hundredths of second
        deviation highbyte      -- interpreted as long in minutes of local time of UTC
        deviation lowbyte
        clock status            -- 0x00 means ok, 0xFF means not specified, others see [3]
}
```

The APDU with request is as follows C0 01 81 00 08 00 00 01 00 00 FF 02 00:

- Type C0 (192 in decimal) means the *Get-Request*
  - 01 means the *get-request-normal*
  - 81 is the *Invoke-id-and-priority* (invoke id = 4, confirmed service, high priority)
  - 00 08 00 00 01 00 00 FF is *Cosem-Attribute-Descriptor* (SEQUENCE) where
    - 00 08 is the COSEM interface class no. 8 which is the clock
    - 00 00 01 00 00 FF is the *Object-Instance-Id*, which is OBIS code 0.0.1.0.0.255 (the clock)
  - 02 means the *attribute-id*, which means the 2nd attribute of the clock. That is *time* [3].
  - 00 denotes an unused *access-selection* item.

The response is given using *get-response* APDU with the following contents: C4 01 81 00 09 0C 07 D2 0C 04 03 0A 06 0B FF 00 78 00

- Type C4 (196 in decimal) means *Get-Response.*
  - 01 means the *Get-Response-Normal* which is SEQUENCE where
    - 81 (1000 0001) is the *Invoke-id-and-priority* (invoke id = 4, confirmed service, high priority)
    - 00 09 0C 07 D2 0C 04 03 0A 06 0B FF 00 78 00 is the *Get-Data-Result* block which is CHOICE where
      - 00 means *Data* item
      - 09 means OCTET STRING
      - 0C gives the length of the string, e.g., 12 bytes
      - Interpretation is the *date_time* structure is as follows:
        - 07 D2 is 2002 in decimal
        - 0C means the 12th month, i.e. December
        - 04 means the 4th day
        - 03 is the third day of the week, i.e., Wednesday
        - 0A denotes an hour, i.e, 10
        - 06 gives a number of minutes
        - 0B gives a number of seconds, i.e., 11
        - FF gives hundredths of seconds, i.e., 255 (not specified)
        - 00 78 is deviation, i.e., 120 minutes
        - 00 is the clock status and means OK
      - The date is Wen, 4th December 2002, 10:06:11

Example 13: Setting the clock

This example sets the clock to the value given in Example 12. It is composed of two packets: *Set-Request* and *Set-Response*. In this example, following APDU structures are used:

```
SET-Request ::= CHOICE {
        set-request-normal                          [1]   Set-Request-Normal,
        set-request-with-first-datablock            [2]   Set-Request-With-First-Datablock,
        set-request-with-datablock                  [3]   Set-Request-With-Datablock,
        set-request-with-list                       [4]   Set-Request-With-List,
        set-request-with-list-and-first-datablock   [5]   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-Response ::= CHOICE {
        set-response-normal                 [1]   Set-Response-Normal,
        set-response-datablock              [2]   Set-Response-Datablock,
        set-response-last-datablock         [3]   Set-Response-Last-Datablock,
        set-response-last-datablock-with-list [4] Set-Response-Last-Datablock-With-List,
        set-response-with-list              [5]   Set-Response-With-List
}
Set-Response-Normal ::= SEQUENCE {
        invoke-id-and-priority          Invoke-Id-And-Priority,
        result                          Data-Access-Result
}
```

The *Set-Request* is encoded as follows: C1 01 81 00 08 00 00 01 00 00 FF 02 00 09 0C 07 D2 0C 04 03 0A 06 0B FF 00 78 00

- Type C1 (193 in decimal) means the *Set-Request.*
  - 01 means the *set-request-normal* which is SEQUENCE where
    - 81 is the *Invoke-Id-And-Priority*
    - 00 08 00 00 01 00 00 FF 02 is the *Cosem-Attribute-Description* which is SEQUENCE where
      - 00 08 is the *class-id* which means the clock
      - 00 00 01 00 00 FF is the *OBIS code* of the object, i.e., 0.0.1.0.0.255 (clock)
      - 02 means attribute no. 2 of the object, i.e., time
    - 00 means the *access-selection* (not used)
    - 09 0C 07 D2 0C 04 03 0A 06 0B FF 00 78 00 is data to be set
      - 09 means OCTET STRING
      - 0C is the length, i.e., 12 bytes
      - D2 0C 04 03 0A 06 0B FF 00 78 00 gives the *time-date* value, i.e., 2002 Dec 4, Wednesday, 10:06:11 (see Example 12)

The response to the *Set-Request* is as follows: C5 01 81 00

- Type C5 (197 in decimal) means the *Set-Response* PDU
  - 01 is the *Set-Response-Normal*
  - 81 is the *Invoke-Id-And-Priority*
  - 00 is the *Data-Access-Result* which means success, see Appendix D.

Example 14: RequestToRead and *ResponseToRead* Messages

This examples shows how SEQUENCE OF data structure is encoded in A-XDR. The format of this APDU is given in Appendix C:

```
RequestToRead       ::=     SEQUENCE of Variable-Access-Specification

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

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

The A-XDR encoding sequence is as follows: 05 01 02 00 10

- Type 05 is an explicit tag denoting the *ReadRequest* message which is SEQUENCE OF.
  - 01 is the length, e.g., the number of items. This messages has only one item of Variable-Access-Specification.
    - 02 indicates CHOICE value which is the *variable-name*. Data type *ObjectName* is Integer16.
    - The value is 00 10 is the name of the variable which is 2 in decimal.

The answer to this request is sequence 0C 01 00 02 02 11 02 01 02 12 01 3E 12 02 CB.

- Type 0C (12 in decimal) denotes the *ReadResponse* message which is SEQUENCE OF CHOICE.
  - 01 is the length of the SEQUENCE OF which is 1 item of CHOICE.
  - 00 is the CHOICE value corresponding to *data* item of type Data.
    - Type 02 gives the CHOICE item of *Data* which is SEQUENCE of Data
    - 02 gives the length of the SEQUENCE which is two items.

- 11 (17 in decimal) is the type of the first item of type *Data* which is Unsigned 8.
  - The value of the first data item is 2
- 01 is the next *Data* item which is of type array, e.g., SEQUENCE of Data
  - 02 indicates the number of *Data* items in the array
    - 12 (18 in decimal) is the type of the first item which is Unsigned 16.
      - The value is 01 3E which is 318 in decimal.
    - 12 is the type of the next *Data* item which is also Unsigned16.
      - The value is 02 CB which is 715 in decimal.

Example 15: DLMSDirector.pcap – GetRequest Messages

This PCAP file was generated by GuruX DLMS Director tool[7]. The communication is transmitted over TCP, port 4061. It uses HDLC encapsulation on top of TCP/IP ☹, see Appendix H. This example demonstrates how to encoded the four starting PDUs.

Packet no.6: 7e a0 19 03 21 b2 67 5c e6 e6 00 c0 01 c1 00 0f 00 00 28 00 00 ff 01 00 f9 79 7e

- 7e a0 19 03 21 b2 67 5c is MAC header of the HDLC frame
  - e6 e6 00 is LLC header
    - c0 01 c1 00 0f 00 00 28 00 00 ff 01 00 is DLMS/COSEM APDU
      - Type C0 is the *Get-Request*
      - 01 means the *get-request-normal*
      - c1 is the *invoke-id-and-priority*
      - 00 0f is the COSEM interface class ID, 15 means the association LN
      - 00 00 28 00 00 ff is the OBIS code 0.0.40.0.0.255 of the object which means the *LN association*
      - 01 is the first attribute of the OBIS object
      - 00 means the *access-selection* (not used)
- f9 79 is the FCS of the HDLC frame
- 7e is the closing flag of the HDLC frame

Packet no. 7: 7e a0 18 21 03 da 92 09 e6 e7 00 c4 01 c1 00 09 06 00 00 28 00 00 ff b6 6e 7e

- HDLC header is 7e a0 18 21 03 da 92 09
  - LLC header is e6 e7 00
  - LLC data is c4 01 c1 00 09 06 00 00 28 00 00 ff
    - Type C4 means the *Get-Response*
    - 01 means the *Get-Response-Normal*
    - c1 is the *invoke-id-and-priority*
    - 00 means *Data*
    - 09 is OCTET STRING

[7] See http://www.gurux.fi/Download [Feb 2018]

        - 06 is the legth
        - 00 00 28 00 00 ff is the value, i.e. the OBIS code of the object 0.0.40.0.0.255
- HDLC footer is b6 6e 7e

Packet no. 25: 7e a0 19 03 21 d4 57 5a e6 e6 00 c0 01 c1 00 08 00 00 01 00 00 ff 02 00 60 1a 7e
- HDLC header is 7e a0 19 03 21 d4 57 5a
    - LLC header is e6 e6 00
    - LLC data is c0 01 c1 00 08 00 00 01 00 00 ff 02 00
        - c0 is the *Get-Request*
        - 01 means the *get-request-normal*
        - c1 is the *invoke-id-and-priority*
        - 00 08 is the COSEM class ID which means the *clock*
        - 00 00 01 00 00 ff is the OBIS code of the object, i.e., 0.0.1.0.0.255 (clock)
        - 02 means the second attribute, which is the time
        - 00 is the *access-selection*
- HDLC closing sequence is 60 1a 7e

Packet no. 26: 7e a0 1e 21 03 fc 3c 06 e6 e7 00 c4 01 c1 00 09 0c 07 e2 02 07 03 0b 2a 25 00 ff c4 00 45 d8 7e
- 7e a0 1e 21 03 fc 3c 06 is MAC header of HDLC PDU
    - e6 e7 00 is LLC header
        - c4 01 c1 00 09 0c 07 e2 02 07 03 0b 2a 25 00 ff c4 00 is DLMS message
            - Type C4 (196 in decimal) is the *Get-Response* message
            - 01 means the *get-response-normal*
            - c1 (1100 0001) is the *invoke-id-and-priority* where the invoke ID is 6 (110), unconfirmed service class (0), high priority (1)
            - 00 means *Data* of the *Get-Data-Result* CHOICE structure
                - 09 is OCTET STRING
                - 0c is the length, i.e., 12 bytes
                - The value is 07 e2 02 07 03 0b 2a 25 00 ff c4 00 and can be interpreted as the time: year 2018, month 2, day 7, Wend, 11:42:37, deviation 65.476 minutes, status OK.
- 45 d8 is FCS of the HDLC frame
- 7e is the closing flag of the HDLC frame

Example 15: DLMSDirector.pcap – G-Block data

This example shows COSEM data that are too big to be encapsulated in one IP packet, thus the application data is transmitted using several packets.

Packet no. 244: 7e a0 19 03 21 5c 17 52 e6 e6 00 c0 01 c1 00 0f 00 00 28 00 00 ff 02 00 91 53 7e
- 7e a0 19 03 21 5c 17 52 is a HDLC header
    - e6 e6 00 is a LLC header
    - c0 01 c1 00 0f 00 00 28 00 00 ff 02 00 are DLMS data

        - c0 denotes the *Get-Request*
        - 01 is the *get-request-normal*
        - c1 is the *invoke-id-and-priority*
        - 00 0f is a class id is the *association LN*
        - 00 00 28 00 00 ff is OBIS code of the requested object, i.e., 0.0.40.0.0.255 (LN Association)
        - 02 is the second attribute which is *object list*
        - 00 is the *access-selection* (not used)
- 91 53 7e is a HDLC closing sequence

Packet no. 245: 7e a8 7e 21 03 74 b8 cd e6 e7 00 c4 02 c1 00 00 00 00 01 00 82 03 f1 01 16 02 04 12 00 0f 11 02 09 06 00 00 28 00 00 ff 02 02 01 0b 02 03 0f 01 16 01 00 02 03 0f 02 16 01 00 02 03 0f 03 16 01 00 02 03 0f 04 16 01 00 02 03 0f 05 16 01 00 02 03 0f 06 16 01 00 02 03 0f 07 16 01 00 02 03 0f 08 16 01 00 02 03 0f 09 16 01 00 02 03 0f 0a 16 01 00 02 03 0f 0b 16 01 00 01 06 02 02 f9 7a 7e

- 7e a8 7e 21 03 74 b8 cd is a HDLC header
    - e6 e7 00 is a LLC header
    - c4 is the *Get-Response*
    - 02 is the *get-response-with-datablock*
    - c1 is the *invoke-id-and-priority*
    - Then follows DataBlock-G structure which is SEQUENCE
        - 00 is *last-block* indicator, 00 means (FALSE)
        - 00 00 00 01 is *block-number* (Unsigned32)
        - 00 means OCTET STRING
        - 82 (1000 0010) is the first byte of the length which says that the length has two bytes. The value of the length comprises sequence 03 f1 that is 1009 bytes.
- f9 7a 7e is a HDLC closing sequence.

Packet no. 247: 7e a8 7e 21 03 96 a4 09 0f 01 16 00 02 02 0f 02 16 00 02 02 0f 03 16 00 02 02 0f 04 16 00 02 02 0f 05 16 00 02 02 0f 06 16 00 02 04 12 00 17 11 00 09 06 00 00 16 00 00 ff 02 02 01 09 02 03 0f 01 16 01 00 02 03 0f 02 16 01 00 02 03 0f 03 16 01 00 02 03 0f 04 16 01 00 02 03 0f 05 16 01 00 02 03 0f 06 16 01 00 02 03 0f 07 16 01 00 02 03 0f 08 16 01 00 02 03 0f 09 16 01 00 01 6f f3 7e

- 7e a8 7e 21 03 96 a4 09 is a HDLC header which is composed of the opening flag (0x7e), format field (0xa8 e), destination address (0x21), source address (0x03), control field (0x96) and header checksum (0xa409).
    - Format Field 0xa87e is 1010 1000 01111 1110 where 1010 (10 in decimal) is the HDLC frame format number 3, the fifths bit 1 is segmentation, and the remaining 11 bits gives the length of the frame (without the opening and closing flags). The length 126 bytes (0x7e in hexadecimal).
    - Segmentation means that this frame is a continuation of the previous HDLC frame, i.d., it is the same frame format (i-Frame) and the same payload. It also means that there is no LLC header and data should be interpreted when reassembled by HDLC layer.
- Data starts with the sequence 0f 01 … and continues until 00 01 (117 bytes).

- 6f f3 7e is a HDLC closing sequence.

In this case we can DLMS data fragmentation on several layers:

1. Data can be fragmented on the application layer using G-Block data structure. This structure divides data into several G-blocks that are sent in the following DLMS packets. Each G-Block has its sequence number and the last block bit. When received, all G-Block data are decapsulated from TCP/IP and HDLC encapsulation and reassembled into one data block by the application layer.
2. Data can be segmented on the HDLC layer. HDLC supports segmentation which means that the payload which exceeds the limits of HDLC information field is transmitted via several HDLC I-frames. Each I-frame has its own sequence number stored in the control field and is acknowledged by S-frames. In addition, each HDLC frame that is a subject of segmentation has Segmentation bit in the Frame forma field set to 1 except the last frame. This helps reassemble HDLC-segmented frames on the receiver side.

References

1. DLMS User Association: COSEM. Glossary of Terms, Technical Report, DLSM UA 1002: 2003. Available at http://dlms.com/documents/White_book_1.pdf [June 2017].
2. DLMS User Association: Excerpt from DLMS UA 1001-1, Blue Book: COSEM. Identification System and Interface Classes, Technical Report, DLSM UA 1000-1:2010, Ed. 10. Available at http://dlms.com/documents/archive/Excerpt_BB10.pdf [June 2017].
3. DLMS User Association: Excerpt from DLMS UA 1000-1, Blue Book: COSEM interface classes and OBIS identification System, Technical Report, DLMS UA 1000-1, Edition 12.0, 2014. Available at http://dlms.com/documents/Excerpt_BB12.pdf [June 2017].
4. Distribution automation using distribution line carrier systems - Part 6: A-XDR encoding rule, IEC 61334-6:2000, 2006, Edition 1.
5. DLMS User Association: Excerpts from DLMS/COSEM. Architectures and Protocols. Technical Report, DLMS UA 1000-2:2007, Sixth Edition. Available at http://dlms.com/documents/archive/Excerpt_GB6.pdf [Jan 2018].
6. Electricity metering - Data exchange for meter reading, tariff and load control - Part 51: Application layer protocols, Technical Report IEC TS 62056-51:1998.
7. Distribution automation using distribution line carrier systems - Part 4: Data communication protocols - Section 41: Application protocol - Distribution line message specification, IEC 61334-4-41:1996, 1996.
8. Olivier Dubuisson: ASN.1 Communication Between Heterogeneous Systems, Morgan Kaufmann, 2000. Available at www.oss.com/asn1/resources/books-whitepapers-pubs/dubuisson-asn1-book.PDF [Jan 2018].
9. Electricity metering - Data exchange for meter reading, tariff and load control - Part 53: COSEM application layer, IEC 62056-53:2006.
10. ITU-T Recommendation X.227: Data Networks and Open System Communications: Connection-oriented Protocol for the Association Control Service Element: Protocol Specification, ITU-T X.227, 1995.
11. Loren Wieth: DLMS/COSEM Protocol Security Evaluation, Technische Universiteit Eidhoven, 2014.
12. DLMS User Association: Excerpts from Companion Specification for Energy Metering. COSEM. Identification Systems and Interface Objects, Technical Report DLMS UA 1000-1:2001, Forth Edition. Available at http://dlms.com/documents/archive/Excerpt_BB4.pdf [Feb 2018].
13. Electricity metering - Data exchange for meter reading, tariff and load control - Part 62: Interface classes, Technical Report IEC TS 62056-62:2006.

Appendix A: Standard COSEM Interface Classes and Attributes Data Types

COSEM standard [13] and DLMS UA recommendations [2,3] defines standard COSEM classes, their attributes and data types. This part gives an overview of the standard COSEM classes and the data types used for selected attributes. For the full list of attributes, see the standards above.

Standard COSEM Interface Classes

Interface Class Name	class_id	Interface Class Name	class_id
Data	1	IEC 8802-2 LLC Type 1 setup	57
Register	3	IEC 8802-2 LLC Type 2 setup	58
Extended register	4	IEC 8802-2 LLC Type 3 setup	59
Demand register	5	Register table	61
Register activation	6	Compact data	62
Profile generic	7	Status mapping	63
Clock	8	Security setup	64
Script table	9	Parameter monitor	65
Schedule	10	Sensor manager	67
Special days table	11	Arbitrator	68
Association SN	12	Disconnect control	70
Association LN	15	Limiter	71
SAP Assignment	17	M-Bus client	72
Image transfer	18	Wireless Mode Q channel	73
IEC local port setup	19	DLMS/COSEM server M-Bus port setup	74
Activity calendar	20	M-Bus diagnostic	77
Register monitor	21	61334-4-32 LLC SSCS setup	80
Single action schedule	22	PRIME NB OFDM PLC Physical layer counters	81
IEC HDLC setup	23	PRIME NB OFDM PLC MAC setup	82
IEC twisted pair setup	24	PRIME NB OFDM PLC MAC functional parameters	83
M-Bus slave port setup	25	PRIME NB OFDM PLC MAC counters	84
Utility tables	26	PRIME NB OFDM PLC MAC network administration data	85
Modem configuration	27	PRIME NB OFDM PLC Application identification	86
Auto answer	28	G3-PLC MAC layer counters	90
Auto connect	29	G3 NB OFDM PLC MAC layer counters	
Data protection	30	G3-PLC MAC setup	91
Push setup	40	G3 NB OFDM PLC MAC setup	
TCP-UDP setup	41	G3-PLC 6LoWPAN adaptation layer setup	92
IPv4 setup	42	G3 NB OFDM PLC 6LoWPAN adaptation layer setup	
MAC address setup	43	ZigBee® SAS startup	101
PPP setup	44	ZigBee® SAS join	102
GPRS modem setup	45	ZigBee® SAS APS fragmentation	103
SMTP setup	46	ZigBee® network control	104
GSM diagnostic	47	ZigBee® tunnel setup	105
IPv6 setup	48	Account	111
S-FSK PHY&MAC setup	50	Credit	112
S-FSK Active Initiator	51	Charge	113
S-FSK MAC sync timeouts	52	Token gateway	115
S-FSK MAC counters	53		
IEC 613344-32 LLC setup	55		
S-FSK Reporting system list	56		

Attributes Data Types

The following table contains data types usable for attributes of COSEM objects [2, page 18].

Type description	Tag	Definition
null-data	[0]	
array	[1]	complex data type
structure	[2]	complex data type
boolean	[3]	TRUE or FALSE
bit-string	[4]	An ordered sequence of boolean values
double-long	[5]	Integer32
double-long-unsigned	[6]	Unsigned32
octet-string	[9]	An ordered sequence of octets (8 bit bytes)
visible-string	[10]	An ordered sequence of ASCII characters
UTF8-string	[12]	An ordered sequence of characters encoded as UTF-8
bcd	[13]	binary coded decimal
integer	[15]	Integer8
long	[16]	Integer16
unsigned	[17]	Unsigned8
long-unsigned	[18]	Unsigned16
compact array	[19]	complex data type
long64	[20]	Integer64
long64-unsigned	[21]	Unsigned64
enum	[22]	0..255
float32	[23]	OCTET STRING (SIZE(4))
float64	[24]	OCTET STRING (SIZE(8))
date_time	[25]	OCTET STRING (SIZE(12))
date	[26]	OCTET STRING (SIZE(5))
time	[27]	OCTET STRING (SIZE(4))

Floating-point numbers shall be represented as a fixed length octet-strings, containing 4 bytes (float32) of the single format or 8 bytes (float64) of the double format floating-point number in the following format: the sign bit (1 bit), the exponent (8 or 11 bits), the fraction (23 or 52 bits) according to the standard IEC 60559.

```
-- data types used in clock interface class, attribute no.2 (time)
date        OCTET STRING (SIZE(5)){
            year highbyte,
            year lowbyte,
            month,
            day of month,
            day of week
}
time        OCTET STRING (SIZE(4)){
            hour,
            minute,
            second,
            hundredths
}
```

```
date_time  OCTET STRING (SIZE(12)) {
           year highbyte,
           year lowbyte,
           month,
           day of month,
           day of week,
           hour,
           minute,
           second,
           hundredths of second,
           deviation highbyte,
           deviation lowbyte,
           clock status
}

-- data types used in Association LN  interface class, attribute no. 2 (object_list)
object_list_type ::= array object_list_element

object_list_element ::= structure {
           class_id:          long-unsigned;      -- unsigned16
           version:           unsigned;           -- unsigned8
           logical_name:      octet-string;
           access_rights:     access_right
}

access_right ::= structure {
           attribute_access:  attribute_access_descriptor;
           method_access:     method_access_descriptor
}

attribute_access_descriptor ::= array attribute_access_item

attribute_access_item ::= structure {
           attribute_id:      integer;            -- integer8
           access_mode :      enum {
                                   no_access          (0)
                                   read_only          (1)
                                   write_only         (2)
                                   read_and_write     (3)
                              };
           access_selectors ::= CHOICE {
                                   null-data;
                                   array of integer8
                              }
}

method_access_descriptor ::= array method_access_item

method_access_item ::= structure {
           method_id:         integer;
           access_mode:       boolean
}
```

Appendix B: Object Identification System (OBIS)

The OBIS defines the identification codes (ID-codes) for commonly used data items in metering equipment using six value groups A to F, see [12].

- *Value group A (Medium)* specifies the media (energy type) to which the metering is related, e.g., abstract data, electricity-, gas-, water-related data. Non-media related information is handled as abstract data. The range for value group A is 0 to 15.

Value group A codes	
Code	Definition
0	Abstract objects
1	Electricity-related objects
4	Heat cost allocator related objects
5	Cooling related objects
6	Heat-related objects
7	Gas-related objects
8	Cold water-related objects
9	Hot water-related objects
F	Other media
All other	Reserved

Examples of abstract objects are in the following table.

Object	A	B	C	D	E	F
Billing period counter	0	b	0	1	0	VZ or 255
Time stamp of the most recent billing period	0	b	0	1	5	VZ or 255
Date of last firmware activation	0	b	96	2	13	
State of output control signals	0	b	96	3	2	
Battery voltage	0	b	96	6	3	
Power failure monitoring in all three phases	0	0	96	7	1	
Number of connections	0	b	96	12	1	
Currently active tariff	0	b	96	14	0..15	

- *Value group B (Channel)* defines the measurement channel number, i.e.., the number of the input of a metering equipment having several inputs for the measurement of energy of the same or different types (e.g., in data concentrators, registration units). Data from different sources can thus be identified. The definition for this value group are independent from the value group A. The range for value group B is 1 to 255.

Value group B codes	
Code	Definition
0	No channel specified
1..64	Channel 1..64
65..127	Reserved
128..254	Manufacturer specific codes
255	Reserved

- *Value group C (Quantity)* defines the abstract or physical data items related to the information source concerned, e.g., current, voltage, power, volume, temperature. The definitions depend on the value of the Value group A. Measurement, tariff processing, classification and data storage methods of these quantities are defined by value groups D, E and F. The range for value group C is 0 to 255.

Value group C codes	
Code	Abstract objects (A=0)
0..89	Context specific identifiers
93	Consortia specific identifiers
94	Country specific identifiers
96	General service entries
97	General error registers
98	General list objects
99	Abstract data profiles
127	Inactive objects
128..254	Manufacturer specific codes
All other	Reserved

Value group C codes	
Code	Electricity (A=1)
0	General purpose objects
1	$\sum L_i$ Active power+
2	$\sum L_i$ Active power-
3	$\sum L_i$ Reactive power+
4	$\sum L_i$ Reactive power-
11	Current: any phase
12	Voltage: any phase
14	Supply frequency
…	…

Value group C codes	
Code	Heat Cost Allocators (HCA) (A=4)
0	General purpose objects
1	Unrated integral
2	Rated integral
3	Radiator surface temperature
4	Heating medium temperature
5	Flow (forward) temperature
6	Return temperature
7	Room temperature
…	…

Value group C codes	
Code	Heat / cooling (A=5 or A=6)
0	General purpose objects
1	Energy
2	Accounted volume
3	Accounted mass
4	Flow volume
5	Flow mass
6	Return volume
7	Return mass
…	…

Value group C codes	
Code	Gas (A=7)
0	General purpose objects
1	Forward undisturbed meter volume
2	Forward disturbed meter volume
3	Forward absolute meter volume
4	Reverse undisturbed meter volume
5	Reverse disturbed meter volume
6	Reverse absolute meter volume
…	…

Value group C codes	
Code	Water (A=8 or A=9)
0	General purpose objects
1	Accumulated volume
2	Flow rate
3	Forward temperature
93	Consortia specific identifiers
94	Country specific identifiers
96	Water related service entries
…	…

- *Value group D (Processing, e.g. integration)* defines types, or the result of the processing of physical quantities identified with the value groups A to C, according to various specific algorithms. The algorithms can deliver energy and demand quantities as well as other physical quantities. The value for value group D is 0 to 255.

Value group D codes	
Code	Consortia specific identifiers (A=any, C=93)
01	SELMA Consortium
All other	Reserved

Value group D codes	
Code	Country specific identifiers (A=any, C=94)
00	Finnish identifiers
01	USA identifiers
02	Canadian identifiers
03	Serbian identifiers
07	Russian identifiers
10	Czech identifiers
11	Bulgarian identifiers
12	Croatian identifiers
…	…

Value group D codes	
Code	Electricity (A=1, C<>0,93,94,96,97,98,99)
0	Billing period average (since last reset)
1	Cumulative minimum 1
2	Cumulative maximum 1
3	Minimum 1
4	Current average 1
5	Last average 1
6	Maximum 1
7	Instantaneous value
8	Time integral 1
…	…

Value group D codes	
Code	HCA (A=4, C<> 0, 96..99)
0	Current value
1	Periodical value
2	Set data value
3	Billing data value
4	Minimum of value
5	Maximum of value
6	Test value
All other	Reserved

Value group D codes	
Code	Heat / cooling (A=5 or A=6, C<> 0, 96..99)
0	Current value
1	Periodical value 1
2	Set data value
3	Billing data value
4	Minimum of value 1
5	Maximum of value 1
6	Test value
…	…

Value group D codes	
Code	Gas (A=7, C<> 1..8,11..16, 21..26, 31..36,61..66)
0	Value at metering conditions
1	Corrected value
2	Value at base conditions
3	Current redundant value at metering conditions
6	Index difference - Value at metering conditions
7	Index difference – Corrected value
8	Index difference – Value at base conditions
…	…

Value group D codes	
Code	Water (A=8 or A=9, C<> 0, 96..99)
0	Current value
1	Periodical value
2	Set date value
3	Billing date value
4	Minimum of value
5	Maximum of value
6	Test value
All other	Reserved

- *Value group E (Classification, e.g., tarification)* defines the further processing of measurement results or classification of quantities identified with value groups A to D to tariff registers, according to the tariff(s) in use. For abstract data or for measurement results for which tariffs is not relevant, this value group can be used for further classification. The range is 0 to 255.

Value group E codes	
Code	Electricity – Tariff rates (A=1)
0	Total
1	Rate 1
2	Rate 2
3	Rate 3
…	…
63	Rate 63
128..254	Manufacturer specific code
All other	Reserved

Value group E codes	
Code	Gas – correction, conversion, compressibility values (A=7, C=51.55, D=0,2,3,10,11,12)
0	Process independent current value
1	Weighted value
11	Average, current interval, averaging period 1

12	Average, last interval, averaging period 1
13	Average, current interval, averaging period 2
14	Average, last interval, averaging period 2
…	…
All other	Reserved

- *Value group F (Historical values)* defines the storage of data identified by value groups A to E, according to different billing periods. The range is 0 to 255. If not used, 255 is set.

Appendix C: DLMS APDU

The following DLMS APDU format corresponds to standard IEC 61334-4-41 [7]. It differs to xDLMS [6] and IEC 62056-51 [6].

```
DLMSpdu ::= CHOICE {
        confirmedServiceRequest                 [0]     RequestToConfirmedService,
        initiateRequest                         [1]     RequestToInitiate,
        getStatusRequest                        [2]     RequestToGetStatus,
        getNameListRequest                      [3]     RequestToGetNameList,
        getVariableAttributeRequest             [4]     RequestToGetVariableAttribute,
        readRequest                             [5]     RequestToRead,
        writeRequest                            [6]     RequestToWrite,
        confirmedServiceResponse                [7]     ResponseToConfirmedService,
        initiateResponse                        [8]     ResponseToInitiate,
        getStatusResponse                       [9]     ResponseToGetStatus,
        getNameListResponse                     [10]    ResponseToGetNameList,
        getVariableAttributeResponse            [11]    ResponseToGetVariableAttribute,
        readResponse                            [12]    ResponseToRead,
        writeResponse                           [13]    ResponseToWrite,
        confirmedServiceError                   [14]    ErrorConfirmedService,
        unconfirmedServiceRequest               [20]    RequestToUnconfirmedService,
        abortRequest                            [21]    RequestToAbort,
        unconfirmedWriteRequest                 [22]    RequestToUnconfirmedWrite,
        unsolicitedServiceRequest               [23]    RequestToUnsolicitedService,
        informationReportRequest                [24]    RequestToInformationReport,
--- the following options are used only in case of ciphered DLMS messages
--- global ciphered pdus
        glo-confirmedServiceRequest             [32]    OCTET STRING
        …
        glo-informationReportRequest            [56]    OCTET STRING
--- dedicated ciphered pdus
        ded-confirmedServiceRequest             [64]    OCTET STRING
        ….
        ded-informationReportRequest            [88]    OCTET STRING
}

confirmedServiceRequest ::= CHOICE {
        --- tags 0 to 6 are reserved
        getDataSetAttribute             [7]     RequestToGetDataSetAttribute,
        getTIAttribute                  [8]     RequestToGetTIAttribute,
        changeScope                     [9]     RequestToChangeScope,
        start                           [10]    RequestToStart,
        stop                            [11]    RequestToStop,
        resume                          [12]    RequestToResume,
        makeUsable                      [13]    RequestToMakeUsable,
        InitiateLoad                    [14]    RequestToInitiateLoad,
        loadSegment                     [15]    RequestToLoadSegment,
        terminateLoad                   [16]    RequestToTerminateLoad,
        initiateUpLoad                  [17]    RequestToInitiateUpLoad,
        upLoadSegment                   [18]    RequestToInitiateUpLoadSegment,
        terminateUpLoad                 [19]    RequestToTerminateUpLoad
}
```

```
confirmedServiceResponse ::= CHOICE {
        --- tags 0 to 13 are reserved
        getDataSetAttribute         [14]    ResponseToGetDataSetAttribute,
        getTIAttribute              [15]    ResponseToGetTIAttribute,
        changeScope                 [16]    ResponseToChangeScope,
        start                       [17]    ResponseToStart,
        stop                        [18]    ResponseToStop,
        resume                      [19]    ResponseToResume,
        makeUsable                  [20]    ResponseToMakeUsable,
        InitiateLoad                [21]    ResponseToInitiateLoad,
        loadSegment                 [22]    ResponseToLoadSegment,
        terminateLoad               [23]    ResponseToTerminateLoad,
        initiateUpLoad              [24]    ResponseToInitiateUpLoad,
        upLoadSegment               [25]    ResponseToInitiateUpLoadSegment,
        terminateUpLoad             [26]    ResponseToTerminateUpLoad
}
confirmedServiceError ::= CHOICE {
        --- tag 0 is reserved
        initiateError               [1]     ServiceError,
        getStatus                   [2]     ServiceError,
        getNameList                 [3]     ServiceError,
        getVariableAttribute        [4]     ServiceError,
        read                        [5]     ServiceError,
        write                       [6]     ServiceError,
        getDataSetAttribute         [7]     ServiceError,
        getTIAttribute              [8]     ServiceError,
        changeScope                 [9]     ServiceError,
        start                       [10]    ServiceError,
        stop                        [11]    ServiceError,
        resume                      [12]    ServiceError,
        makeUsable                  [13]    ServiceError,
        initiateLoad                [14]    ServiceError,
        loadSegment                 [15]    ServiceError,
        terminateLoad               [16]    ServiceError,
        initiateUpLoad              [17]    ServiceError,
        upLoadSegment               [18]    ServiceError,
        terminateUpLoad             [19]    ServiceError,
}
RequestToInitiate :: = SEQUENCE {
        dedicated-key                       OCTET STRING OPTIONAL,
        response-allowed                    BOOLEAN DEFAULT TRUE,
        proposed-quality-of-service         [0] IMPLICIT Integer8 OPTIONAL,
        proposed-dlms-version-number        Unsigned8,
        proposed-conformance                Conformance,
        proposed-max-pdu-size               Unsigned16
}
ResponseToInitiate ::= SEQUENCE {
        negotiated-quality-of-service       IMPLICIT Integer8 OPTIONAL,
        negotiated-dlms-version-number      Unsigned8,
        negotiated-conformance              Conformance,
        server-max-receive-pdu-size         Unsigned16,
        vaa-name                            ObjectName
}
```

```
Conformance ::=     [APPLICATION 30] IMPLICIT BIT STRING (SIZE(16)) {
        get-data-set-attribute                  (0),
        get-ti-attribute                        (1),
        get-variable-attribute                  (2),
        read                                    (3),
        write                                   (4),
        uncofirmed-write                        (5),
        change-scope                            (6),
        start                                   (7),
        stop-resume                             (8),
        makeUsable                              (9),
        load-data-set                           (10),
        get-name-list-choice                    (11),
        access-least-bit                        (12),
        access-last-bit                         (13),
        multiple-variable-list                  (14),
        data-set-upload                         (15),
}
Conformance ::=     [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24)) {
        reserved(0)                             (0),
        reserved(0)                             (1),
        reserved(0)                             (2),
        read                                    (3),
        write                                   (4),
        uncofirmed-write                        (5),
        reserved(0)                             (6),
        reserved(0)                             (7),
        attribute0-supported-with-SET           (8),
        priority-mgmt-support                   (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)
}
RequestToRead       ::=     SEQUENCE of Variable-Access-Specification
ResponseToRead      ::=     SEQUENCE of CHOICE {
        data                    [0]     Data,
        data-access-error       [1]     IMPLICIT Data-Access-Result
}
RequestToWrite      ::=     SEQUENCE {
        variable-access-specification   SEQUENCE OF Variable-Access-Specification,
        list=of-data                    SEQUENCE OF Data
}
```

```
ResponseToWrite ::= SEQUENCE of CHOICE {
        success [0] IMPLICIT NULL,
        data-access-error [1] IMPLICIT Data-Access-Result
}

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

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

Data ::= CHOICE {
        null-data [0] NULL,
        array [1] SEQUENCE OF Data,
        structure [2] SEQUENCE OF Data,
        boolean [3] BOOLEAN,
        bit-string [4] BIT STRING,
        double-long [5] Integer32,
        double-long-unsigned [6] Unsigned32,
        floating-point [7] OCTET STRING,
        octet-string [9] OCTET STRING,
        visible-string [10] VisibleString,
        time [11] GeneralizedTime
        bcd [13] Integer8,
        integer [15] Integer8,
        long [16] Integer16,
        unsigned [17] Integer8,
        long-unsigned [18] Unsigned16
        compact-array [19] SEQUENCE {
                content-description [0] TypeDescription,
                array-contents [1] OCTET STRING
        }
        long64 [20] Integer64,
        long64-unsigned [21] Unsigned64,
        enum [22] ENUMERATED,
        float32 [23] OCTET STRING (SIZE(4)),
        float64 [24] OCTET STRING (SIZE(8)),
        date_time [25] OCTET STRING (SIZE(12)),
        date [26] OCTET STRING (SIZE(5))
        time [27] OCTET STRING (SIZE(4))
        don't-care [255] NULL
}
```

```
ServiceError ::= CHOICE {
        initiate    [6]     IMPLICIT ENUMERATED {
                    other                           (0),
                    DLMS-version-too-low            (1),
                    incompatible-conformance        (2),
                    PDU-size-too-short              (3),
                    refused-by-the-VDE-Handler      (4)
        }
}

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

Appendix D: DLMS/COSEM APDU

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. The following format of COSEM PDU is extracted from [9]. The format of ACSE-apdu is standardized by ITU-T X.227 [10][8] which is identical to IEC 8650-1.

```
COSEMpdu ::= CHOICE {

---- standardized DLMS APDUs using SN referencing
        initiateRequest                 [1]     InitiateRequest,
        readRequest                     [5]     ReadRequest,
        writeRequest                    [6]     WriteRequest,
        initiateResponse                [8]     InitiateResponse,
        readResponse                    [12]    ReadResponse,
        writeResponse                   14]     ConfirmedServiceError,
        unconfirmedWriteRequest         [22]    UnconfirmedWriteRequest
        informationReportRequest        [24]    InformationReportRequest,

--- APDUs used for data communication services using LN referencing
        get-request                     [192]   GET-Request,                    -- 0xC0
        set-request                     [193]   SET-Request,                    -- 0xC1
        even-notification-request       [194]   EVENT-NOTIFICATION-Request,     -- 0xC2
        action-request                  [195]   ACTION-Request,                 -- 0xC3
        get-response                    [196]   GET-Response,                   -- 0xC4
        set-response                    [197]   SET-Response,                   -- 0xC5
        action-response                 [199]   ACTION-Response,                -- 0xC7

--- global ciphered pdus
        glo-get-request                 [200]   OCTET STRING,
        glo-set-request                 [201]   OCTET STRING,
        glo-event-notification-request  [202]   OCTET STRING,
        glo-action-request              [203]   OCTET STRING,
        glo-get-response                [204]   OCTET STRING,
        glo-set-response                [205]   OCTET STRING,
        glo-action-response             [207]   OCTET STRING,

--- dedicated ciphered pdus
        ded-get-request                 [208]   OCTET STRING,
        ded-set-request                 [209]   OCTET STRING,
        ded-event-notification-request  [210]   OCTET STRING,
        ded-actionRequest               [211]   OCTET STRING,
        ded-get-response                [212]   OCTET STRING,
        ded-set-response                [213]   OCTET STRING,
        ded-action-response             [215]   OCTET STRING,
        exception-response              [216]   OCTET STRING
}
```

[8] See http://www.itu.int/ITU-T/formal-language/itu-t/x/x227bis/1998/ACSE-1.html [Feb 2018]

```
--- the four ACSE APDUs (see ITU-T X.227, see Appendix E)

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

Release-request-reason ::= INTEGER {
        normal (0),
        urgent(1),
        user-defined(30)
}

Release-response-reason ::= INTEGER {
        normal (0),
        not-finished(1),
        user-defined(30)
}

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-priorority        Invoke-Id-And-Prirority,
        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--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-First-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-ttribute-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-ttribute-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-Priroity,
        cosem-method-descriptor         Cosem-Method-Descriptor,
        method-invocation-parameters    Data OPTIONAL
}

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

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

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

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

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

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

EXCEPTION-Response ::=SEQUENCE {
        state_error                     [0],
        service_error                   [1]
}

state_error IMPLICIT ENUMERATED {
        service_not_allowed             [1],
        service_unknown                 [2]
}
service_error IMPLICIT ENUMERATED {
        operation_not_possible          [1]
        service_not_supported           [2]
        other_reason                    [3]
}

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

```
Cosem-Class-Id ::= Unsigned16

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

Cosem-Object-Attribute-Id ::= Integer8

Cosem-Object-Method-Id ::= Integer8

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

Cosem-Attribute-Descriptor ::= SEQUENCE {
        class-id            Cosem-Class-Id,
        instance-id         Cosem-Object-Instance-Id,
        attribute-id        Cosem-Object-Attribute-Id
}
ObjectName ::= Integer16

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

Data ::= CHOICE {
        null-data           [0]     IMPLICIT NULL,
        array               [1]     IMPLICIT SEQUENCE OF Data,
        structure           [2]     IMPLICIT SEQUENCE OF Data,
        boolean             [3]     IMPLICIT BOOLEAN,
        bit-string          [4]     IMPLICIT BIT STRING,
        double-long         [5]     IMPLICIT Integer32,
        double-long-unsigned [6]    IMPLICIT Unsigned32,
        octet-string        [9]     IMPLICIT OCTET STRING,
        visible-string      [10]    IMPLICIT VisibleString,
        bcd                 [13]    IMPLICIT Integer8,
        integer             [15]    IMPLICIT Integer8,
        long                [16]    IMPLICIT Integer16,
        unsigned            [17]    IMPLICIT Unsigned8,
        long-unsigned       [18]    IMPLICIT Unsigned16,
        compact-array       [19]    IMPLICIT SEQUENCE {
                                    contents-description [0]    TypeDescription,
                                    array-contents       [1]    IMPLICIT OCTET STRING
                                    }
        long64              [20]    IMPLICIT Integer64,
        long64-unsigned     [21]    IMPLICIT Unsigned64,
        enum                [22]    IMPLICIT ENUMERATED,
        float32             [23]    IMPLICIT OCTET STRING (SIZE(4)),
        float64             [24]    IMPLICIT OCTET STRING (SIZE(8)),
        date_time           [25]    IMPLICIT OCTET STRING (SIZE(12)),
        date                [26]    IMPLICIT OCTET STRING (SIZE(5))
        time                [27]    IMPLICIT OCTET STRING (SIZE(4))
        do-not-care         [255]   IMPLICIT NULL
}
```

```
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
}

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)
}

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

```
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
                }
}
--- APDUs using SN referencing

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
}
```

Appendix E: ACSE APDU

This is the abstract syntax of each of the ACSE APDUs. The ACSE APDUs are part of DLMS/COSEM communication during opening and closing phase. ACSE APDUs are standardized by IEC 8650-1 standard or X.227 [10]. The APDUs are encoded by BER for transmission. The *user-information* field contains a COSEMpdu encoded by A-XDR [4].

```
AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE {     --- APPLICATION 0 = 60H = 96
        protocol-version                [0]     IMPLICIT BIT STRING { version 1(0)} DEFAULT {version 1},
        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-ide        [5]     AE-invocation-identifier OPTIONAL,
        calling-AP-title                [6]     AP-title OPTIONAL,
        calling-AE-quantifier           [7]     AE-qualifier OPTIONAL,
        calling-AP-invocation-id        [8]     AP-invocation-identifier OPTIONAL,
        calling-AE-invocation-id        [9]     AE-invocation-identifier OPTIONAL,
        sender-acse-requirements        [10]    IMPLICIT ACSE-requirements OPTIONAL,
        mechanism-name                  [11]    IMPLICIT Mechanism-name OPTIONAL,
        calling-authentication-value    [12]    EXPLICIT Authentication-value OPTIONAL,
        application-context-name-list   [13]    IMPLICIT Application-context-name-list OPTIONAL,
        p-context-definition-list       [14]    Syntactic-context-list OPTIONAL,
        called-asoi-tag                 [15]    IMPLICIT ASOI-tag OPTIONAL,
        calling-asoi-tag                [16]    IMPLICIT ASOI-tag 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 {version 1(0)} DEFAULT {version 1},
        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,
        responder-acse-requirements     [8]     IMPLICIT ACSE-requirement OPTIONAL,
        mechanism-name                  [9]     IMPLICIT Mechanism-name OPTIONAL,
        responding-authentication-value [10]    EXPLICIT Authentication-value OPTIONAL,
        application-context-name-list   [11]    IMPLICIT Application-context-name-list OPTIONAL,
        p-context-result-list           [12]    IMPLICIT P-context-result-list OPTIONAL,
        called-asoi-tag                 [13]    IMPLICIT ASOI-tag OPTIONAL,
        calling-asoi-tag                [14]    IMPLICIT ASOI-tag OPTIONAL,
        implementation-information      [29]    IMPLICIT Implementation-data OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}
```

```
RLRQ-apdu ::= [APPLICATION 2] IMPLICIT SEQUENCE {     --- APPLICATION 2 = 62H = 98
        reason                          [0]     IMPLICIT Release-request-reason OPTIONAL,
        aso-qualifier                   [13]    ASO-qualifier OPTIONAL,
        asoi-identifier                 [14]    IMPLICIT ASOI-identifier OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}

RLRE-apdu ::= [APPLICATION 3] IMPLICIT SEQUENCE {      --- APPLICATION 3 = 63H = 99
        reason                          [0]     IMPLICIT Release=response-reason OPTIONAL,
        aso-qualifier                   [13]    ASO-qualifier OPTIONAL,
        asoi-identifier                 [14]    IMPLICIT ASOI-identifier OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}

ABRT-apdu ::= [APPLICATION 4] IMPLICIT SEQUENCE {
        abort-source                    [0]     IMPLICIT ABRT-source,
        abort-diagnostic                [1]     IMPLICIT ABRT-diagnostic OPTIONAL,
        aso-qualifier                   [13]    ASO-qualifier OPTIONAL,
        asoi-identifier                 [14]    IMPLICIT ASOI-identifier OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}


A-DT-apdu ::= [APPLICATION 5] IMPLICIT SEQUENCE {
        aso-qualifier                   [0]     ASO-qualifier OPTIONAL,
        asoi-identifier                 [1]     IMPLICIT ASOI-identifier OPTIONAL,
        a-user-data                     [30]    IMPLICIT User-Data
}

ABRT-diagnostic ::= ENUMERATED {
        no-reason-given                                         (1),
        protocol-error                                          (2),
        authentication-mechanism-name-not-recognized            (3),
        authentication-mechanism-name-required                  (4),
        authentication-failure                                  (5),
        authentication-required                                 (6)
}

ABRT-source ::= INTEGER {
        acse-service-user               (0),
        acse-service-provider           (1)
}

ACSE-requirements      ::= BIT STRING {
        authentication                   (0),
        application-context-negotiation (1),
        higher-level-association        (2),
        netsted-association             (3)
}

Application-context-name-list ::= SEQUENCE OF Application-context-name

Application-context-name ::= OBJECT IDENTIFIER
```

```
Authentication-value ::= CHOICE {
        charstring          [0]     IMPLICIT GraphicString,
        bistring            [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[9]

Mechanism-name ::= OBJECT IDENTIFIER

Authentication-value ::= CHOICE {
        charstring          [0]     IMPLICIT GraphicString,
        bitstring           [1]     IMPLICIT BIT STRING,
        external            [2]     IMPLICIT SEQUENCE {
                            other-mechanism-name        Mechanism-name,
                            other-mechanism-value       ANY DEFINED BY other mechanism-name
        }
}

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-recognized        (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)
        }
}
```

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

Appendix F: ASN.1 and BER Encoding

Abstract Syntax Notation 1 (ASN.1, defined by ITU-T X.680) specifies the following categories of data types [see also 8]:

- Primitive data types (universal class 00)
  - BOOLEAN – universal class tag 1
  - INTEGER – universal class tag 2
  - BIT STRINT – universal class tag 3
  - OCTET STRING – universal class tag 4
  - NULL – universal class tag 5
  - OBJECT IDENTIFIER – universal class tag 6
  - ObjectDescriptor – universal class tag 7
  - REAL – universal class tag 9
  - ENUMERATED – universal class tag 10
  - UTF8String – universal class tag 12
  - NumericString – universal class tag 18
  - PrintableString – universal class tag 19
  - IA5String – universal class tag
  - UTCTime – universal class tag 23
  - GeneralizedTime – universal class tag 24
  - GraphicString – universal class tag 25
  - VisibleString – universal class tag 26
  - GeneralString – universal class tag 27
  - UniversalString – universal class tag 28
  - CHARCTER STRING – universal tag 29
- Application-wide data types (class 01)
  - Not standardized but defined by each application.
- Constructor data types
  - SEQUENCE, SEQUENCE OF – universal class tag 16
  - SET, SET OF – universal class tag 17
  - CHOICE
  - SELECTION
  - ANY

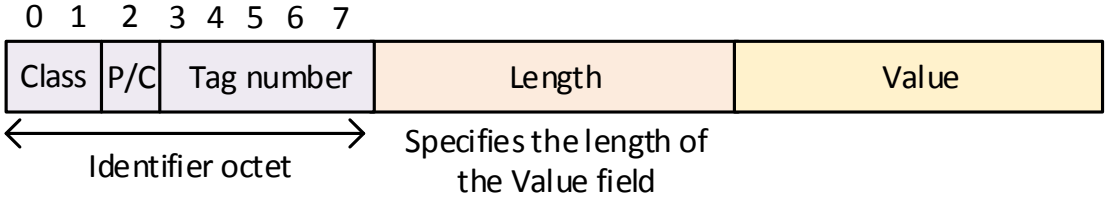
Basic Encoding Rules (BER, standard ITU-T X.690) defines transfer syntax of ASN.1 data structures transmitted between applications. BER describes a method how to encode values of ANS.1 data as a string of octets. It encodes an ASN.1 value as a triplet TLV (type-length-value) that includes an identifier of the data type, length of the value, and the value itself, see the following figure.

- *Type* (or identifier) is one-byte value that indicates the ASN.1 type, see below.
- *Length* indicates the length of the actual value representation.
- *Value* represents the value of ASN.1 type as a string of octets. For constructed types the value can be an embedded TLV triplet.

Example 1: sequence 41 02 3F 22 (hex)

- Type 41 = 0100 0001 (binary) denotes class 01 (application), primitive type (0) and the application tag is 1 (00001), see below. Application tag 1 in SNMP is Counter32 data type.
- 02 gives the length of the data, e.g., 2 bytes.
- 3F 22 is the value of the variable of type Counter 32. The value is 16,162 in decimal.

The *identifier* specifies the ASN.1 data type, the class of the type, and the method of encoding, see the following figure:



- *The first two bits* determine the class of the ASN.1 data type:
  - Universal (00)
  - Application (01)
  - Context-specific (10)
  - Private (11).
  - Universal data types (primitive or constructive) are given by the standard.
- *The third bit* describes the encoding method: primitive (0) or constructed (1) form.
  - If set to 1, constructed data types as SEQUENCE, SET, CHOICE, etc. are used. It also means that another TLV triplet is embedded as a value.
- *The last five bits* identify the data type. This is called a tag. Universal data type tags are listed above. Application, context-specific and private tags are defined by the application.

Encoding BIT STRING value

The encoding form of BIT STRING value can be primitive or constructed.

In the primitive form, the string is cut up in octets and a leading octet is added so that the number of bits left unused at the end could be identified by an integer between 0 and 7. If this octet is 0, it means that all bits are used.

For example, BIT STRING 1011 0111 0101 1 (13 bits) will be aligned to two octets (16 bits), e.g., 1011 0111 0101 1000, thus three zero bits are left added to align the bit string to octets. BER encoding will be *0000 0011* 1011 0111 0101 1000, where the first octet (italic) represents the number of left added zero bits (3) and two following octets represent the bit string without three last zero bits. For further details, see [8].

Encoding OBJECT IDENTIFIER value

BER specifies how to encode and decode OID value. There are two rules for encoding:

1. Since the OID represents a path in the MIB tree where the prefix is usually the same, thus the first two bytes X, Y of OID string are compressed by BER into one byte using the following formula: Z = (X . 40) + Y.
    - Decoding of the first BER-encoded byte representing OID value is as follows:
        - if 0 <= Z <= 39, then X = 0 and Y = Z
        - if 40 <= Z <= 79, then X = 1 and Y = Z - 40
        - if Z >= 80, then X = 2 and Y = Y – 80
2. Every integer (except the first two) is encoded on a series of octets, with 7 bits being used from each octet and the most significant bit (MSB) is set to 1 in all but the last octet. The fewest possible number of octets must be used.
    - For example number 250 (1111 1010) will be encoded as two bytes, where the first bit will be in the first byte and the remaining seven bits will be in the second byte, e.g., 1000 0001 and 0111 1010 where the MSB of the first byte is set to 1 and MSB of the last byte is set to 0. The representation in hexadecimal format is 81 7A.

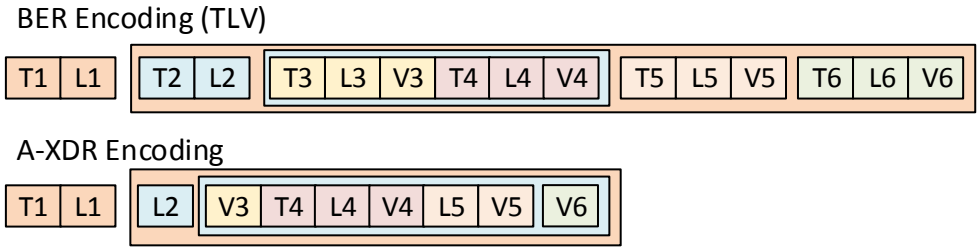
Example 2: sequence 30 10 06 0A 2B 06 01 02 01 02 02 01 11 04 41 02 3F 22 (hex)

- Type 30 = 0011 0000 (binary) means universal class (00), constructed type (1) with tag 16. Tag 16 for universal data types denotes SEQUENCE (see above).
- The length of the sequence is 16 bytes (10 in hex)
- The value contains an embedded TLV triplet:
    - Type 06 = 0000 0110 (binary) defines universal class (00), primitive type (0), and data type with tag 6 which is OBJECT IDENTIFIER.
    - The length of the value is 10 bytes (0x0A).
    - The value is 2B 06 01 02 01 02 02 01 11 04 (10 bytes).
        - Thus 0x2B = 43 in decimal, i.e., X = 1, Z = 3. The resulting OID is 1.3.6.1.2.1.2.2.1.17.4.
- The remaining data is another embedded TLV triplet that is a part of SEQUENCE data type.
    - This triplet 41 02 3F 22 is decoded in Example 1.

Appendix G: A-XDR Encoding

A-XDR (Adjusted External Data Representation) encoding is a transfer syntax that is used to encode ASN.1 data for transmission. It is standardized by IEC 61334-6 standard [4] and used in DLMS APDUs.

A-XDR has a compact form. While Basic Encoding Rules (BER) encodes an ASN.1 value as the triple TLV (Type-Length-Value), see Appendix F, A-XDR encodes the type and the length only when it is necessary for clarity. This also means that without knowing the type of an encoding value the encoding structure cannot be determined. The figure below shows difference between BER and A-XDR encoding when applied on six ASN.1 data structures.



Example 1: Suppose the following data structure:

```
value := SEQUENCE {
        A       Integer16,
        B       Unsigned16
}
```

Having data {0x1234, 0x5678}, the BER encoding would be as follows: 30 08 02 02 12 34 02 02 56 78.

- Type 30 (0011 0000) means the universal data type (00), constructive form (1), data type tag 10000 (16 in decimal) means SEQUENCE
- The length is 8 bytes, the value is a sequence of two TLV triplets.
    - 02 (0000 0010) means the universal data type (00), primitive form (0), tag 2 means INTEGER
    - The length is 2 bytes and the value is 0x1234
    - Another triplet 02 02 56 78 denotes INTEGER of length 2 bytes and value 0x5678.

A-XDR encoding of the same data will be (T) 12 34 56 78

- (T) means type which is required only in the case that the SEQUENCE structure is a part of CHOICE structure. Otherwise T will be skipped
- 12 34 is the value of the first item of the SEQUENCE, e.g., 0x1234
- 56 78 is the value of the second item of the SEQUENCE, e.g., 0x5678

TLV Structure

A-XDR rules specify when T (type) field and L (length) field are present in the encoded structure:

- Field T (Type) is present only in the following cases:
    - in the CHOICE structure the T field is used and indicates the chosen item,
    - in the SEQUENCE structure T is used to identify existence of an OPTIONAL item,
    - in the SEQUENCE structure T is used to identify existence of an DEFAULT item.[10]
- Field L (Length) is only used before variable-length INTEGER, BIT STRING, OCTET STRING.

[10] Existence of OPTIONAL or DEFAULT items is indicated by BOOLEAN existence indicator where 0x00 is non-existence and 0x01 means existence of the item.

A-XDR Encoding

In this section, encoding rules for common universal data types will be presented, incl. examples.

1. INTEGER value
    - Fixed-length integers are encoded in binary and aligned to bytes.
        - E.g., value 61478 is encoded as 0xF0 26
    - Variable-length integers are encoded as follows:
        - Numbers 0 <= x <= 127 are encoded as one byte where first bit is 0.
            - E.g., number 123 will be encoded as 0x7B.
        - Other numbers contain L field and V field. The length of the L field is 1 byte. The first bit of the L field is 1, the next 7 bits represent the length of V field in bytes. That means that the maximal length of INTEGER value is 1016 bits.
        - E.g., 128 will be encoded as 0x82 00 80, -128 as 0x82 FF 80.
2. BOOLEAN value
    - TRUE is 0xFF, FALSE is 0x00
3. ENUMERATED value
    - Range of enumerated data type is 0..255 and is expressed as the fixed-length INTEGER.
4. BIT STRING value
    - For the fix-length BIT STRING, A-XDR encodes only the value. The unused bits up to the byte are set to 0.
        - E.g., bit sequence 01100111 01010 (13) is encoded as 0x67 50.
    - For the variable-length BIT STRING, field Length is added. The value in L field denotes number of bits in BIT STRING. The L field is encoded as variable-length INTEGER, where the first bit denotes if the length is only in 1 byte (bit = 0) or several bytes (bit=1).
        - E.g., bit sequence 01100111 01010 (13 bits) will be encoded as 0x 0D 67 50 where 0D is the length.
        - E.g., 131 bits sequence will start with L field 0x81 83 where the first bit of the first byte says that the length is encoded by more than 1 byte, the length of the L field is 1 byte (0x81), the value is of the length field is 0x83 (=131). After the L field, there will be 17 bytes with the bits.
5. OCTET STRING value
    - For fixed-length OCTET STRING, only V field is present.
        - E.g., string “ABCD” is encoded as 0x41 42 43 44.
    - For variable-length OCTET STRING, the L field and V field are present. The length is encoded as variable-length INTEGER.
        - E.g., string “ABC” is encoded as 0x03 41 42 43 where 03 is the length.
        - E.g., a string of 347 bytes has the following L field: 0x81 01 5B which says that the length of the L field is 2 bytes (0x81) and the length itself is 0x01 5B (=347).
6. CHOICE value
    - A-XDR requires that all CHOICE items are explicitly specified. The CHOICE identifier is encoded as one-byte integer number.
        - Example: Suppose the following data structure

```
PDU := CHOICE {
        a       [0]     INTEGER,
        b       [1]     OCTET STRING (4)
}
```

- A-XDR encoding of "a=3715" is 0x00 82 0E 83 where 00 is the first choice (identifier 0), 0x 82 OE 83 is representation of variable-length integer where 0x 82 specifies the length of the value (2 bytes). The value is 0xOE 83.
- A-XDR encoding of "b=ABCD" is 0x01 41 42 43 44 where 0x01 represents the choice (option with identification 1) and 0x 41 42 43 44 are ASCII values of the string.

7. SEQUENCE value
   - SEQUENCE data type is a composed of embedded data types (primitive or constructive). Each item of the SEQUENCE must be present in the SEQUENCE data type instance, with exception of items denoted as OPTIONAL or DEFAULT. The order of items must be preserved.
     - Example: Suppose the following data structure

```
PDU := SEQUENCE {
        a       INTEGER (0.127),
        b       OCTET STRING (4) OPTIONAL,
        c       [1] BOOLEAN DEFAULT TRUE
}
```

- A-XDR encoding of "a=37, b=ABCD, c=FALSE" is 0x25 01 41 42 43 44 01 00 where 0x25 represent value 37, 0x01 means OPTIONAL item is present, 0x41 42 43 44 decode ABCD, 0x01 means that c item is present and is different to default value, i.e. is FALSE (value 0x00).
- A-XDR encoding of "a=37, b is omitted, c=FALSE" is 0x25 00 01 00 where 0x25 is the value of variable a, 0x00 means b is not present, 0x01 means c is present, 0x00 is value of variable c which is FALSE.
- A-XDR encoding of "a=37, b=ABCD, c=TRUE" is 0x25 01 41 42 43 44 00 where 0x25 means value 37 of item a, 0x01 indicates the existence of OPTIONAL string (T field), 0x41 42 43 44 represents "ABCD" and the last 0x00 indicates non-existence of item c, thus default value TRUE is used.

8. SEQUENCE OF value
   - ANS.1 type SEQUENCE OF is defined as a constructive type. A-XDR provides two kinds of encoding that depends on whether SEQUENCE OF has fixed length or not.
   - Fixed-length SEQUENCE OF encoding
     - A-XDR encoding includes only values of the SEQUENCE OF items. The length N indicates the number of items. The order of the items must be kept as defined in ANS.1 declaration.
     - Example: Suppose the following data structure

```
PDU := SEQUENCE (LENGTH(2)) OF BIT STRING.
```

If the value is 00101 and 1101 0010 1000, then the encoding is 0x05 28 0C D2 80 where 0x05 is the length of the first BIT STRING, 0x28 is the value (00101 + 000 unused bits). The second BIT STRING is encoded as 0x0C (length, i.e., 12 bits), 0xD2 (first part of the

BIT STRING, i.e., 11010010) and 0x80 (the second part of the BIT STRING, .i.e., 1000 + four unused bits).

- Variable-length SEQUENCE OF encoding
  - A-XDR encoding includes length (L field) and value (V field) where the length describes the number of items of SEQUENCE OF data structure. The L field is encoded as the variable-length BIT STRING (see above). The V field is composed of N data values of the type given in ASN.1 definition where N is given in L field. The order of the values must be the same as the order of items in SEQUENCE OF.
  - Example: Suppose the following data structure

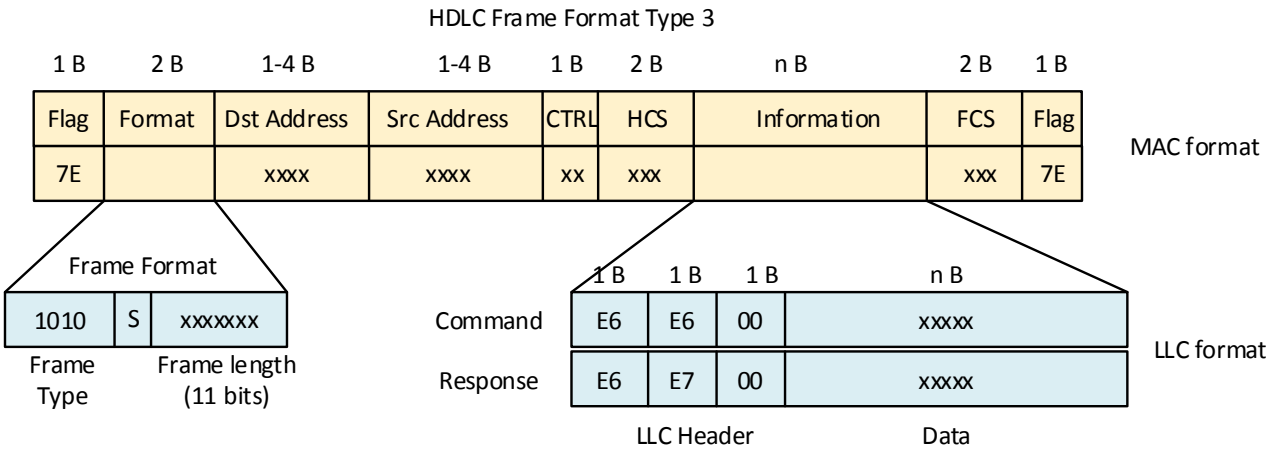
    PDU := SEQUENCE OF INTEGER (0..4000).

    The AXD-R encoding of the sequence of two items with integer values 1956 and 3624 will be 0x02 07 A4 0E 08 where 0x02 gives the the number of items (length), 0x07A4 is fixed-length value 1956 and 0x0e08 is another value 3624 encoded as fixed-length integer.

9. VISIBLE STRING value
   - A-XDR encodes VISIBLE STRING value as OCTET STRING with variable length.
10. UNIVERSAL TIME value
    - UNIVERSAL TIME is encoded as OCTET STRING with variable length.

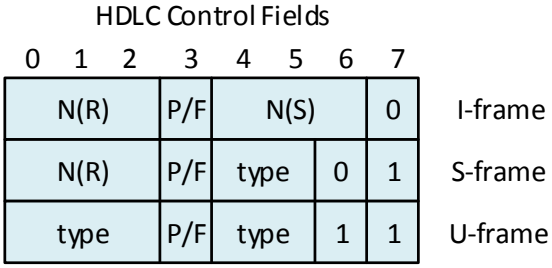
Appendix H: HDLC Encapsulation

Data transmitted by DLMS/COSEM protocol are usually encapsulated in HDLC frames defined by ISO/IEC 13239. The standard uses HDLC frame format type 3 described as follows [5]:



Frames that do not have an information field, do not contain an HCS, only an FCS.

- The first four bits of the frame format identify the format of HDLC frame. DLMS uses HDLC frame format type 3 which is 1010 (0xA).
- The control field (CTRL) indicates the type of commands or responses, and contains HDLC sequence numbers, where appropriate. The last bits of the control field (CTRL) identify the type of the HDLC frame:



  - *I-frame* (information frame) transport user data from the network layer.
  - *S-frame* (supervisory frame) are used for flow and error control when piggybacking is not possible. They do not have information fields. There are three S-frame types:
    - 00: Receive ready (RR)
    - 01: Receive not read (RNR)
    - 10: Reject (REJ)
    - 11: Selective reject (SREJ)
  - *U-frame* (unnumbered frame) are used for link management (mode settings, recovery, miscellaneous), and can be also used to transfer user data.
    - Unnumbered Information (UI): 000 P/F 00 11 – unacknowledged data with a payload.
    - Unnumbered Poll (UP): 001 P 00 11 – used to solicit control information.

HDLC communicates on the data link layer which has two sublayers: the Media Access Control (MAC) and the Logical Link Control (LLC).

For DLMS/COSEM parsing it is necessary to identify outer encapsulation. HDLC can be identified by the opening and closing flag which is 0x7E. The length of HDLC header is variable depending on the length of source and destination addresses which can be 1 to 4 bytes. For the purposes of DLMS/COSEM, the LCC PDU has fixed format, i.e. the header is as follows:

- The destination LLC address is 0xE6.
- The source LLC address is either 0xE6 (for commands) or 0xE7 (for responses).
- The LLC quality byte is always 8-bits long and must be set to 0x00.

Example 1: 7E A0 19 95 75 54 68 35 E6 E6 00 C0 01 81 00 08 00 00 01 00 00 FF 01 00 0D FD 7E

- This PDU is a HDLC-encapsulated data.
- Frame format is 0xA0 19 (1010 0000 0001 1001) where 1010 is the frame type no. 3 (I-frames), 0 is the segmentation flat, and the last 11 bits give the length of HDLC frame without the opening and closing flag (7e). In this case the length is 25 bytes.
- Destination address is 0x95 (1001 0101). The last bit (1) says that this byte is the last byte of the address. Thus, destination HDLC address is upper 7 bits, which is 0x4A.
- Source address is 0x75 (0100 1011) which is also 1-byte address where the address itself is again 7 upper bits, e.g., the address is 0x25.
- Control field indicates the type of commands or responses: 54 (0101 0100) means I-frame (last bit is 0), the next receiving frame no. 2 (010, the first three bits), polling bit set to 1 (the 4th bit) and the next sending frame no. 2 (010, bits 5-7)
- Header check sequence(HCS) is 68 35
- LLC header is E6 E6 00
- DLMS data is C0 01 81 00 08 00 00 01 00 00 FF 01 00, see Example 12.
- Frame check sequence (FCS) is 0D FD
- 7E is the closing flag

Example 2: 7e a0 07 03 21 71 13 c5 7e

- 7e is the opening flag of HDLC.
- a0 07 (1010 0000 0000 0111) is the frame format field which says it is frame format type no. 3 (1010), no segmentation (0), the length of the HDLC frame is 7 bytes (without the opening and closing flag).
- 03 (0000 0011) is the destination address field. The LSB (1) says it is just one-byte address. Its value is contained in 7 upper bits, thus the address is 0x01, which is 1 in decimal.
- 21 (0010 0001) is the source address field. The LSB (1) means it is a one-byte address with value 0010 000 which is 16 in decimal.
- 71 (0111 0001) is the control field which says that it is a S-Frame (01), more specifically RR (receiver ready) frame.
- Since this frame does not have any information field nor HCS, the before last two bytes 13 c5 form frame check sequence (FCS).
- 7e is the closing flag.

Example 3: 7e a8 7e 21 03 96 a4 09 0f 01 16 00 02 02 0f 02 16 00 02 02 0f 03 16 00 02 02 0f 04 16 00 02 02 0f 05 16 00 02 02 0f 06 16 00 02 04 12 00 17 11 00 09 06 00 00 16 00 00 ff 02 02 01 09 02 03 0f 01 16 01 00 02 03 0f 02 16 01 00 02 03 0f 03 16 01 00 02 03 0f 04 16 01 00 02 03 0f 05 16 01 00 02 03 0f 06 16 01 00 02 03 0f 07 16 01 00 02 03 0f 08 16 01 00 02 03 0f 09 16 01 00 01 6f f3 7e

- 7e is the opening flag of HDLC.
- a8 7e (1010 1000 0111 1110) is the frame format field: 1010 is the frame format type no. 3, 1 is the segmentation bit set to 1, the length of the data in HDLC frame is 126 bytes.
- 21 (0010 0001) is the destination address field where the address is 16 in decimal.
- 03 (0000 0011) is the source address field where the address is 1 in decimal.
- 96 (1001 0110) is the control field which says that it is an I-frame.
- a4 09 is header check sequence (HCS)
- 0f 01 16 … 01 00 01 is the payload of length 126-2-1-1-1-2-2 = 117 bytes.
- 6f f3 is the FCS
- 7e is the closing flag.