INTERNATIONAL STANDARD

IEC 62056-62

Second edition
2006-11

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

Part 62: Interface classes

Reference number
IEC 62056-62:2006(E)

Publication numbering

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

Consolidated editions

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

Further information on IEC publications

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

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

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 On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

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

- **Customer Service Centre**

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

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

INTERNATIONAL STANDARD

IEC 62056-62

Second edition
2006-11

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

Part 62: Interface classes

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XF

For price, see current catalogue

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

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

Part 62: Interface classes

FOREWORD

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

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this International Standard may involve the use of a maintenance service concerning the stack of protocols on which the present standard IEC 62056-62 is based.

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

The provider of the maintenance service has assured the IEC that he is willing to provide services under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the provider of the maintenance service is registered with the IEC. Information (see also 4.6.2 and Annex E) may be obtained from:

DLMS[1] User Association

Geneva / Switzerland

www.dlms.ch

[1] Device Language Message Specification.

International Standard IEC 62056-62 Ed. 2 has been prepared by IEC technical committee 13: Equipment for electrical energy measurement and load control.

This second edition cancels and replaces the first edition published in 2002 and constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

- the list of common data types has been amended, some new types have been added;
- formatting for floating point numbers has been added;
- new HLS mechanisms have been added;
- instance specific data types have been replaced with a well-defined set of applicable data types;
- new units have been added;
- encoding of application_context_name and authentication_mechanism_name attributes of the Association LN class has has been clarified;
- new interface classes “Register table” and “Status mapping” have been added;
- a new version of the “IEC local port setup”, “Modem configuration”, “Auto connect” and “HDLC setup” interface classes have been added;
- new interface classes for setting up a TCP/IP based communication profile have been added. References to related IETF RFCs and standards, as well as related definitions have been added;
- several amendments in Annex D “Relation to OBIS” have been made.

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

FDIS	Report on voting
13/1389/FDIS	13/1400/RVD

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

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

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

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

A bilingual version of the publication may be issued at a later date.

INTRODUCTION

Driven by the need of the utilities to optimize their business processes, the meter becomes more and more part of an integrated metering and billing system. Whereas in the past the commercial value of a meter was mainly generated by its data acquisition and processing capabilities, nowadays the critical issues are system integration and interoperability.

The Companion Specification for Energy Metering (COSEM) addresses these challenges by looking at the meter as an integrated part of a commercial process, which starts with the measurement of the delivered product (energy) and ends with the revenue collection.

The meter is specified by its “behaviour” as seen from the utility's business processes. The formal specification of the behaviour is based on object modelling techniques (interface classes and objects). The specification of these objects forms a major part of COSEM.

The COSEM server model (see 4.5) represents only the externally visible elements of the meter. The client applications that support the business processes of the utilities, customers and meter manufacturers make use of this server model. The meter offers means to retrieve its structural model (the list of objects visible through the interface), and provides access to the attributes and specific methods of these objects.

The set of different interface classes form a standardized library from which the manufacturer can assemble (model) its individual products. The elements are designed so that with them the entire range of products (from residential to commercial and industrial applications) can be covered. The choice of the subset of interface classes used to build a meter, their instantiation, and their implementation are part of the product design and therefore left to the manufacturer. The concept of the standardized metering interface class library provides the different users and manufacturers with a maximum of diversity without having to sacrifice interoperability.

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

Part 62: Interface classes

1 Scope

This part of IEC 62056 specifies a model of a meter as it is seen through its communication interface(s). Generic building blocks are defined using object-oriented methods, in the form of interface classes to model meters from simple up to very complex functionality.

2 Normative references

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

IEC 60050-300:2001, *International Electrotechnical Vocabulary – Electrical and electronic measurements and measuring instruments – Chapter 311: General terms relating to measurements – Chapter 312: General terms relating to electrical measurements – Chapter 313: Types of electrical measuring instruments – Chapter 314: Specific terms according to the type of instrument*

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*

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

IEC 62051:1999, *Electricity metering – Glossary of terms*

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

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

IEC 62056-31:1999, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 31: Using local area networks on twisted pair with carrier signalling*

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

IEC 62056-47:2006, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 47: COSEM transport layers for IPv4 networks*

IEC 62056-53:2006, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 53: COSEM Application layer*

---

[2] To be published.

IEC 62056-61:2006, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 61: Object identification system(OBIS)*

ANSI C12.19:1997 / IEEE 1377:1997, *Utility Industry End Device Data Tables*

STD 0005: 1981, *Internet Protocol (Also: IETF RFC 0791, RFC 0792, RFC 0919, RFC 0922, RFC 0950, RFC 1112)*

STD 0051: 1994, *The Point-to-Point Protocol (PPP) (Also: IETF RFC 1661, RFC 1662)*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 60050-300, IEC 62051, and IEC 62051-1 apply.

3.2 Abbreviations

AARE	Application Association Response
AARQ	Application Association ReQuest
ACSE	Application Control Service Element
APDU	Application Protocol Data Unit
ASE	Application Service Element
A-XDR	Adapted eXtended Data Representation
CHAP	Challenge Handshake Authentication Protocol
COSEM	Companion Specificationfor Energy Metering
CtoS	Client to Server Challenge
DHCP	Dynamic Host Control Protocol
DLMS	Device Language Message Specification
DNS	Domain Name Server
EAP	Extensible Authentication Protocol
GMT	Greenwich Mean Time
GPS	Global Positioning System
HLS	High Level Security
IANA	Internet Assigned Numbers Authority
IC	Interface Class
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPCP	Internet Protocol Control Protocol
LCP	Link Control Protocol
LLS	Low Level Security
LN	Logical Name
LSB	Least Significant Bit
m	mandatory
MD5	Message Digest Algorithm 5
MSB	Most Significant Bit
o	Optional

OBIS	OBject Identification System
PAP	Password Authentication Protocol
PDU	Protocol Data Unit
PLMN	Public Land Mobile Network
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
ROHC	Robust Header Compression
SAP	Service Access Point
SHA-1	Secure Hash Algorithm
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SN	Short Name
StoC	Server to Client Challenge

4 Basic principles

4.1 General

This subclause describes the basic principles on which the COSEM interface classes are built. It also gives a short overview on how interface objects (instantiations of the interface classes) are used for communication purposes. Data collection systems and metering equipment from different vendors, following these specifications can exchange data in an interoperable way.

Object modelling: for specification purposes this standard uses the technique of object modelling. An object is a collection of attributes and methods.

The information of an object is organized in attributes. They represent the characteristics of an object by means of attribute values. The value of an attribute may affect the behaviour of an object. The first attribute in any object is the “logical_name”. It is one 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 interface class with a class_id. Within a specific class, the common characteristics (attributes and methods) are described once for all objects. Instantiations of an interface class are called COSEM objects.

Manufacturers may add proprietary methods or attributes to any object, using negative numbers.

Figure 1 illustrates these terms by means of an example:

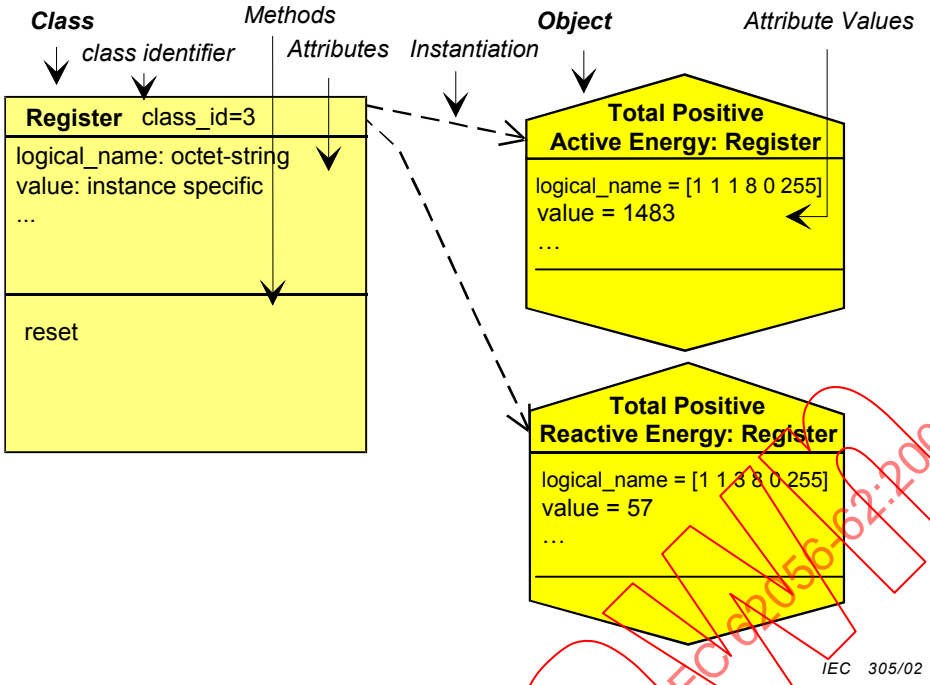


Figure 1 – An interface class and its instances

The interface class "Register" is formed by combining the features necessary to model the behaviour of a generic register (containing measured or static information) as seen from the client (central unit, hand-held terminal). The contents of the register are identified by the attribute "logical_name". The logical_name contains an OBIS identifier (see IEC 62056-61). The actual (dynamic) content of the register is carried by its "value" attribute.

Defining a specific meter means defining several specific registers. In the example of Figure 1, the meter contains two registers, i.e. two specific COSEM objects of the class "Register" are instantiated. This means that specific values are assigned to the different attributes. Through the instantiation, one COSEM object becomes a "total, positive, active energy register" whereas the other becomes a "total, positive, reactive energy register".

REMARK The COSEM objects (instances of interface classes) represent the behaviour of the meter as seen from the "outside". Therefore, modifying the value of an attribute must always be initiated from the outside (e.g. resetting the value of a register). Internally initiated changes of the attributes are not described in this model (e.g. updating the value of a register).

4.2 Class description notation

This subclause describes the notation used to define the interface classes.

A short text describes the functionality and application of the class. A table gives an overview of the class including the class name, the attributes, and the methods (class description template).

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. …		…			

Each attribute and method must be described in detail.

Class name	Describes the class (e.g. “Register”, “Clock”, “Profile generic”,...)
Cardinality	Specifies the number of instances of the class within a logical device (see 4.6). *value* — The class shall be instantiated exactly “value” times. *min...max.* — The class shall be instantiated at least “min.” times and at most “max.” times. If min. is zero (0) then the class is optional, otherwise (min. > 0) "min." instantiations of the class are mandatory.
class_id	Identification code of the class (range 0 to 65 535). The class_id of each object is retrieved together with the logical name by reading the object_list attribute of an “Association LN” / ”Association SN” object. The class_id-s from 0 to 8 191 are reserved to be specified by the DLMS UA. Class_id-s from 8 192 to 32 767 are reserved for manufacturer specific interface classes. Class_id-s from 32 768 to 65 535 are reserved for user group specific interface classes. DLMS UA reserves the right to assign ranges to individual manufacturers or user groups.
Version	Identification code of the version of the class. The version of each object is retrieved together with the logical name and the class_id by reading the object_list attribute of an “Association LN” / ”Association SN” object. **Within one logical device, all instances of a certain class must be of the same version**.
Attribute(s)	Specifies the attribute(s) that belong to the class. (*dyn.*) — Classifies an attribute that carries a process value, which is updated by the meter itself. *(static)* — Classifies an attribute, which is not updated by the meter itself (e.g. configuration data).
logical_name	octet-string — The logical name is always the first attribute of a class. It identifies the instantiation (COSEM object) of this class. The value of the logical_name conforms to OBIS (see IEC 62056-61).
Data type	Defines the data type of an attribute (see 4.3).
Min.	Specifies if the attribute has a minimum value. *x* — The attribute has a minimum value. <*empty*> — The attribute has no minimum value.
Max.	Defines if the attribute has a maximum value. *x* — The attribute has a maximum value. <*empty*> — The attribute has no maximum value.
Def.	Specifies if the attribute has a default value. This is the value of the attribute after reset. *x* — The attribute has a default value. <*empty*> — The default value is not defined by the class definition.

Specific method(s)	Provides a list of the specific methods that belong to the object.	
	Method Name ()	The method has to be described in the subsection "Method description".
m/o	Defines if the method is mandatory or optional.	
	m (mandatory)	The method is mandatory.
	o (optional)	The method is optional.

Attribute description

Describes each attribute with its data type (if the data type is not simple), its data format and its properties (minimum, maximum and default values).

Method description

Describes each method and the invoked behaviour of the instantiated COSEM object(s).

NOTE Services for accessing attributes or methods by the protocol are described in IEC 62056-53.

Selective access

The xDLMS services Read, Write, UnconfirmedWrite (used with SN referencing) and GET, SET (used with LN referencing) typically reference the entire attribute. However, for certain attributes selective access to just a part of the attribute may be provided. The part of the attribute is identified by specific selective access parameters. These are defined as part of the attribute specification.

4.3 Common data types

The following table contains the list of data types usable for attributes of COSEM objects.

Table 1 – Common data types

Type description	Tag [a]	Definition	Value range
--simple data types			
null-data	[0]		
boolean	[3]	boolean	TRUE or FALSE
bit-string	[4]	An ordered sequence of boolean values	
double-long	[5]	Integer32	-2 147 483 648... 2 147 483 647
double-long-unsigned	[6]	Unsigned32	0...4 294 967 295
octet-string	[9]	An ordered sequence of octets (8 bit bytes)	
visible-string	[10]	An ordered sequence of ASCII characters	
bcd	[13]	binary coded decimal	
integer	[15]	Integer8	-128...127
long	[16]	Integer16	-32 768...32 767
unsigned	[17]	Unsigned8	0...255
long-unsigned	[18]	Unsigned16	0...65 535
long64	[20]	Integer64	$-2^{63}...2^{63}-1$
long64-unsigned	[21]	Unsigned64	$0...2^{64}-1$

Table 1 *(continued)*

Type description	Tag [a]	Definition	Value range
enum	[22]	The elements of the enumeration type are defined in the "Attribute description" section of a COSEM interface class specification.	
float32	[23]	OCTET STRING (SIZE(4))	For formatting, see 4.4.2.
float64	[24]	OCTET STRING (SIZE(8))	
date_time	[25]	OCTET STRING (SIZE(12))	For formatting, see 4.4.1.
date	[26]	OCTET STRING (SIZE(5))	
time	[27]	OCTET STRING (SIZE(4))	
--complex data types			
array	[1]	The elements of the array are defined in the "Attribute description" section of a COSEM interface class specification.	
structure	[2]	The elements of the structure are defined in the "Attribute description" section of a COSEM interface class specification.	
compact array	[19]	The elements of the compact array are defined in the "Attribute description" section of a COSEM interface class specification.	
--CHOICE		For some attributes of some COSEM interface objects, the data type may be chosen at COSEM object instantiation, in the implementation phase of the COSEM server. The Server always shall send back the data type and the value of each attribute, so that together with the logical name, an unambiguous interpretation is ensured. The list of possible data types is defined in the "Attribute description" section of a COSEM interface class specification.	

[a] The tags are as defined in IEC 62056-53 8.3.

4.4 Data formats

4.4.1 Date and time formats

Date and time information may be represented with data type octet-string, or using the data types *date, time* and *date_time*, as defined in the relevant interface class definition.

NOTE 1 In future versions of interface classes and in newly defined interface classes, only the data types *date, time* and *date_time* will be used.

NOTE 2 The (SIZE( )) specifications do not apply if *date*, *time* or *date_time* are represented by data type octet-string.

date OCTET STRING (SIZE(5))
{
year highbyte,
year lowbyte,
month,
day of month,
day of week
}
year: interpreted as long-unsigned
range 0…big
0xFFFF = not specified
year highbyte and year lowbyte reference the 2 bytes of the

long-unsigned

month: interpreted as unsigned
range 1…12, 0xFD, 0xFE, 0xFF
1 is January
0xFD = daylight_savings_end
0xFE = daylight_savings_begin
0xFF = not specified

dayOfMonth: interpreted as unsigned
range 1…31, 0xFD, 0xFE, 0xFF
0xFD = 2nd last day of month
0xFE = last day of month
0xE0 to 0xFC = reserved
0xFF = not specified

dayOfWeek: interpreted as unsigned
range 1…7, 0xFF
1 is Monday
0xFF = not specified

For repetitive dates, the unused parts must be set to “not specified”.

The elements dayOfMonth and dayOfWeek have to be interpreted together:

- if last day of month is specified (0xFE) and day of week is wildcard, this specifies the last calendar day of the month;
- if last day of month is specified (0xFE) and an explicit day of week is specified (e.g. 7, Sunday) then it is the last occurrence of the weekday specified in the month, i.e. the last Sunday;
- if the dayOfMonth and dayOfWeek elements are both explicitly defined and they are not consistent, (for example 24th of the month is not Wednesday in the given year and month) it shall be considered as an error.

time OCTET STRING (SIZE(4))
{
hour,
minute,
second,
hundredths
}
hour: interpreted as unsigned
range 0…23, 0xFF
0xFF = not specified,
minute: interpreted as unsigned
range 0…59, 0xFF
0xFF = not specified,
second: interpreted as unsigned
range 0…59, 0xFF
0xFF = not specified,
hundredths: interpreted as unsigned
range 0…99, 0xFF
0xFF = not specified

For repetitive times the unused parts must be set to “not specified”.

deviation	long	–720…720: in minutes of local time to GMT 0x8000 = not specified
clock_status	unsigned interpreted as 8 bit string The status bits are defined as follows: bit 0 (LSB): invalid [a] value, bit 1: doubtful [b] value, bit 2: different clock base [c], bit 3: invalid clock status [d], bit 4: reserved, bit 5: reserved, bit 6: reserved, bit 7 (MSB): daylight saving active [e]	

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
	}
	Individual fields of *date_time* are encoded as defined above. Some may be set to "not specified" as described above in *date* and *time*.	

[a] Time could not be recovered after an incident. Detailed conditions are manufacturer specific (e.g. after the power to the clock has been interrupted).

[b] Time could be recovered after an incident but the value cannot be guaranteed. Detailed conditions are manufacturer specific.

[c] Bit is set if the basic timing information for the clock is at the actual moment taken from a timing source different from the source specified in clock_base.

[d] This bit indicates that at least one bit of the clock status is invalid. Some bits may be correct. The exact meaning shall be explained in the manufacturer's documentation.

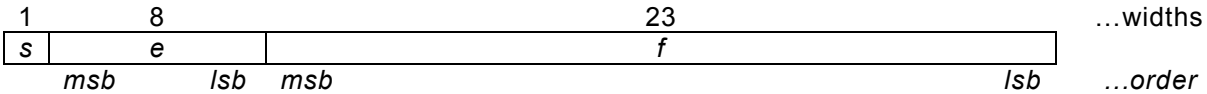
[e] Flag set to true: the transmitted time contains the daylight saving deviation (summer time), Flag set to false: the transmitted time does not contain daylight saving deviation (normal time).

4.4.2 Floating point number formats

Floating point number formats are defined in IEC 60559.

NOTE: For the following, IEC 60559 is equivalent to IEEE 754.

The single format is:

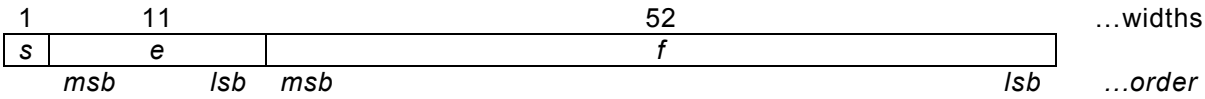


where:
- s is the sign bit;
- e is the exponent; it is 8 bits wide and the exponent bias is +127;
- f is the fraction, it is 23 bits.

With this, the value is (if 0 < *e* < 255):

$$v = (-1)^s \cdot 2^{e-127} \cdot (1.f)$$

The double format is:



where
s is the sign bit;
e is the exponent; it is 11 bits wide and the exponent bias is +1 023;
f is the fraction, it is 52 bits.

With this, the value is (of 0 < *e* < 2 047):

$$v = (-1)^s \cdot 2^{e-1023} \cdot (1.f)$$

For more detail, see IEC 60559.

Floating-point numbers shall be represented as a fixed length octet-string, containing the 4 bytes (float32) of the single format or 8 the bytes (float64) of the double format floating-point number as specified above, most significant byte first.

Example 1: The decimal value “1” represented in single floating-point format is:

Bit 31 Sign bit **0** 0: + 1: -	Bits 30-23 Exponent field **01111111** Decimal value of exponent field and exponent: **127**-127 = 0	Bits 22-0 Significand 1.**00000000000000000000000** Decimal value of the significand: 1.0000000

NOTE The significand is the binary number 1 followed by the radix point followed by the binary bits of the fraction.

The encoding, including the tag of the data type is (all values are hexadecimal): 17 3F 80 00 00.

Example 2: The decimal value “1” represented in double floating-point format is:

Bit 63 Sign bit **0** 0: + 1: -	Bits 62-52 Exponent field **01111111111** Decimal value of exponent field and exponent: **1023**-1023 = 0	Bits 51-0 Significand 1.**0000000000000000000000000000000000000000000000000000** Decimal value of the significand: 1.0000000000000000

The encoding, including the tag of the data type is (all values are hexadecimal): 18 3F F0 00 00 00 00 00 00.

Example 3: The decimal value “62056” represented in single floating-point format is:

Bit 31 Sign bit **0** 0: + 1: -	Bits 30-23 Exponent field **10001110** Decimal value of exponent field and exponent: **142**-127 = 15	Bits 22-0 Significand 1.**11100100110100000000000** Decimal value of the significand: 1.8937988

The encoding, including the tag of the data type is (all values are hexadecimal): 17 47 72 68 00.

Example 4: The decimal value “62056” represented in double floating-point format is:

Bit 63 Sign bit **0** 0: + 1: -	Bits 62-52 Exponent field **10000001110** Decimal value of exponent field and exponent: **1038**-1023 = 15	Bits 51-0 Significand 1.**11100100110100000000000000000000000000000000000000000** Decimal value of the significand: 1.8937988281250000

The encoding, including the tag of the data type is (all values are hexadecimal): 18 40 EE 4D 00 00 00 00 00.

4.5 The COSEM server model

The COSEM server is structured into three hierarchical levels as shown in Figure 2:

Level 1: Physical device;

Level 2: Logical device;

Level 3: Accessible COSEM objects.

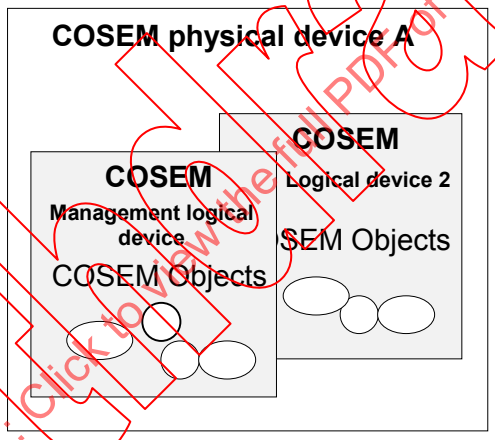


Figure 2 – The COSEM server model

The following example (see Figure 3) shows how a combined metering device can be structured using the COSEM server model.

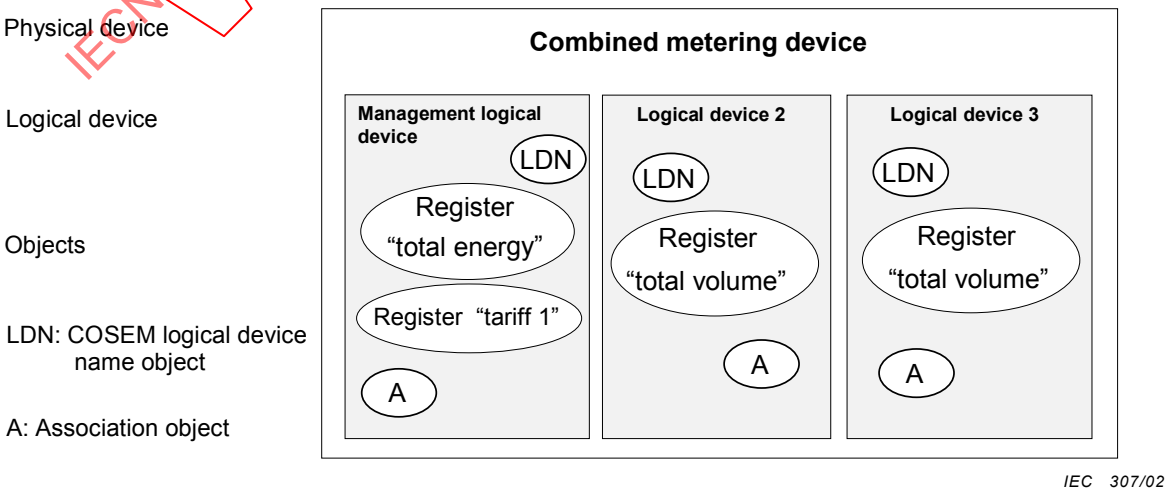


Figure 3 – Combined metering device

4.6 COSEM logical device

4.6.1 General

The COSEM logical device is a set of COSEM objects. Each physical device shall contain a "Management logical device".

The addressing of COSEM logical devices shall be provided by the addressing scheme of the lower layers of the protocol used.

4.6.2 COSEM logical device name

The COSEM logical device can be identified by its unique COSEM logical device name. This name can be retrieved from an instance of IC "SAP assignment" (see 5.14), or of a COSEM object "COSEM logical device name" (see D.2.1.24).

This name is defined as an octet-string of up to 16 octets. The first three octets uniquely identify the manufacturer of the device [3]. The manufacturer is responsible for guaranteeing the uniqueness of the octets that follow (up to 13 octets).

4.6.3 The "association view" of the logical device

In order to access COSEM objects in the server, an application association shall first be established. This characterizes the context within which the associated applications will communicate. The major parts of this context are:

- the application context;
- the authentication context;
- the xDLMS context.

This information is contained in a special COSEM object, the "Association" object. There are two types of this association object defined. One for associations using short name referencing ("Association SN") and one for using logical name referencing ("Association LN").

Depending on the association established between the client and the server, different access rights may be granted by the server. Access rights concern a set of COSEM objects – the visible objects – that can be accessed ('seen') within the given association. In addition, access to attributes and methods of these COSEM objects may also be restricted within the association (e.g. a certain type of client can only read a particular attribute of a COSEM object).

The list of the visible COSEM objects – the "association view" – can be obtained by the client by reading the "*object_list*" attribute of the appropriate association object. Additional information about the access rights (read only, write only, read and write) to the attributes and the availability of the methods (within the established association) can be found via specific attributes (logical name referencing, see 5.12) or special methods (short name referencing, see 5.13) provided by the association objects.

4.6.4 Mandatory contents of a COSEM logical device

The following objects shall be part of each COSEM logical device. They shall be accessible for GET/READ in all application associations with this logical device:

- COSEM logical device name object;
- current association (LN or SN) object.

---

[3] Administered by the DLMS User Association

4.6.5 Management logical device

As specified in 4.6.1, the management logical device is a mandatory element of any physical device, and it has a reserved address. As defined in 6.3.4 of IEC 62056-53, it must support an application association to a public client with the lowest security level. Its role is to support revealing the internal structure of the physical device and to support notification of events in the server.

In addition to the "Association" object modelling the association with the public client, the management logical device shall contain a "SAP assignment" object, giving its SAP and the SAP of all other logical devices within the physical device. The SAP assignment object must be readable at least by the public client.

If there is only one logical device within the physical device, the "SAP assignment" object may be omitted.

4.7 Authentication procedures

4.7.1 Low Level Security (LLS) authentication

As described in IEC 62056-53 the ACSE provides the authentication services for low level security (LLS). Low level security authentication is typically used when the communication channel offers adequate security to avoid eavesdropping and message (password) replay.

For LLS, all the authentication services are provided by the ACSE. The association objects provide only the method/attribute (see 5.12, 5.13) to change the "secret" (e.g. password).

For LLS authentication the client transmits a "secret" (e.g. a password) to the server, by using the "Calling_Authentication_Value" parameter of the COSEM-OPEN.request service primitive of the client application layer. The server checks if the received "secret" corresponds to the client identification. If yes, the client is authenticated and the association can be established.

4.7.2 High Level Security (HLS) authentication

As described in IEC 62056-53, the ACSE provides part of the authentication services for high level security (HLS). High level security authentication is typically used when the communication channel offers no intrinsic security and precautions have to be taken against eavesdroppers and against message (password) replay. In this case, a 4-pass authentication protocol is foreseen. The 4-pass authentication allows the authentication of the client as well as of the server in the following way.

Pass1: The client transmits "challenge" CtoS (e.g. a random number) to the server.

Pass2: The server transmits "challenge" StoC (e.g. a random number) to the client.

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

Pass3: The client processes StoC according to the rules of the HLS authentication mechanism negotiated. In case of HLS authentication_mechanism_id(2), the method of processing the challenge is secret (e.g. encrypting with a secret key) which is the HLS secret known by both the client and the server. In case of HLS authentication_mechanism_id(3), the client appends the HLS secret to the challenge StoC received during Pass2 and generates the digest using the MD5 algorithm (RFC 1321). In case of authentication_mechanism_id(4), the process is the same, but the SHA_1 algorithm is used (FIPS 180-1). The result – f(StoC) – is sent back to the server. The server checks if f(StoC) is the result of correct processing and – if correct – accepts the authentication of the client.

Pass4: If the client is authenticated, the server processes CtoS in the same way as described in Pass3. The result – f(CtoS) – is sent back to the client. The client checks if f(CtoS) is the result of the correct processing and – if correct – accepts the authentication of the server.

The HLS authentication service, supporting Pass1 is provided by the COSEM-OPEN.request service primitive of the client application layer. The parameter "Security_Mechanism_Name" carries the identifier of the HLS mechanism, and the parameter "Calling_ Authentication_Value" carries the challenge CtoS.

The HLS authentication service, supporting Pass2 is provided by the COSEM-OPEN.response service primitive of the server application layer. The parameter "Security_Mechanism_Name" carries the identifier of the HLS mechanism, and the parameter "Responding_ Authentication_Value" carries the challenge StoC.

After Pass2, the association is formally established, but the access of the client is restricted to the method "reply_to_HLS_authentication" of the current "association" object.

Pass3 and Pass4 are supported by the method **reply_to_HLS_authentication** of the association object(s), (see 5.12, 5.13). If both passes are successfully executed, then full access is granted according to the current association. Otherwise, either the client or the server aborts the association.

In addition, the association object provides the method to change the HLS "secret" (e.g. the encryption key): **change_HLS_secret.**

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

5 The interface classes

The currently defined interface classes for meters and the relations between them are illustrated in Figure 4.

NOTE 1 The interface class "base" itself is not specified explicitly. It contains only one attribute "logical_name".

NOTE 2 In the description of the "Demand register", "Clock" and "Profile generic" interface classes, the 2nd attributes are labelled differently from that of the 2nd attribute of the "Data" interface class, namely "current_average_value", "time" and "buffer" vs. "value". This is to emphasize the specific nature of the "value".

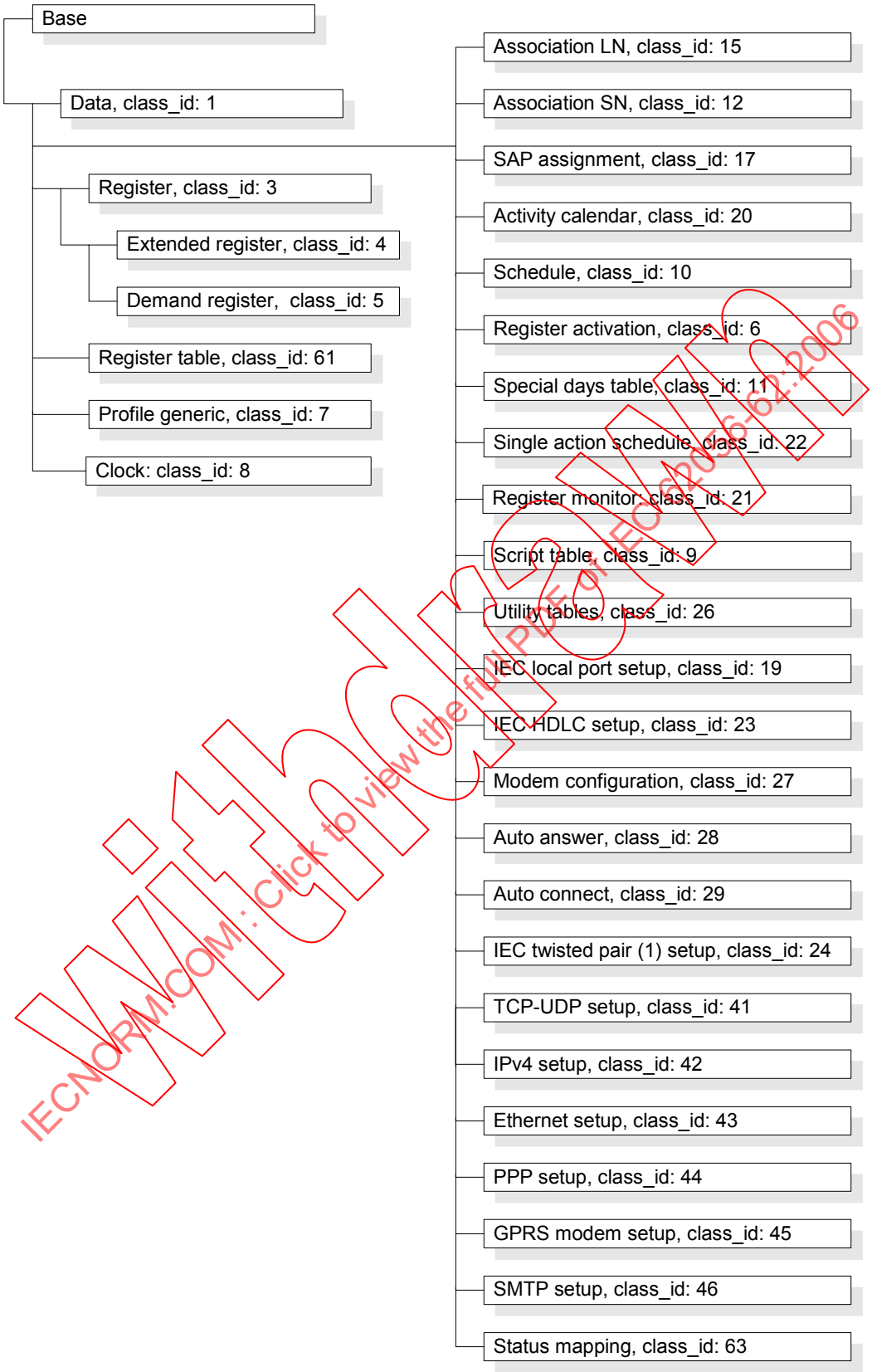


Figure 4 – Overview of the interface classes

5.1 Data (class_id: 1)

A “Data” object stores data related to internal meter object(s). The meaning of the value is identified by the logical_name. The data type of the value is CHOICE. “Data” is typically used to store configuration data and parameters.

Data	**0...n**	**class_id = 1, version = 0**		
Attribute(s)	***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name (static) 2. value	octet-string *CHOICE*			
Specific method(s)	***m/o***			

Attribute description

logical_name	Identifies the “Data” object instance. Identifiers are specified in Clause D.22 and in IEC 62056-61.
value	Contains the data.

```
CHOICE                                  The data type depends on the
{                                       instantiation defined by the “logical
--simple data types                     name” and possibly from the
null-data                [0],           manufacturer. It has to be chosen so,
boolean                  [3],           that together with the logical name, an
bit-string               [4],           unambiguous interpretation is
double-long              [5],           possible. Any simple and complex
double-long-unsigned     [6],           data types listed in 4.3 can be used,
octet-string             [9],           unless the choice is restricted in
visible-string           [10],          Annex D.
bcd                      [13],
integer                  [15],
long                     [16],
unsigned                 [17],
long-unsigned            [18],
long64                   [20],
long64-unsigned          [21],
enum                     [22],
float32                  [23],
float64                  [24],
date-time                [25],
date                     [26],
time                     [27],
--complex data types
array                    [1],
structure                [2],
compact-array            [19]
}
```


5.2 Register (class_id: 3)

A “Register” object stores a process value or a status value with its associated unit. The register object knows the nature of the process value or of the status value. The nature of the value is described by the attribute “logical name” using the OBIS identification system (see Clause D.2 and IEC 62056-61).

Register		0...n	class_id = 3, version = 0		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. value	(dyn.)	CHOICE			
3. scaler_unit	(static)	scal_unit_type			
Specific method(s)		***m/o***			
1. reset (data)		o			

Attribute description

logical_name

Identifies the “Register” object instance. Identifiers are specified in Clause D.2 and in IEC 62056-61.

value

Contains the current process or status value.

```
CHOICE                                  The data type of the value depends on
{                                       the instantiation defined by
--simple data types                     “logical_name” and possibly on the
null-data                 [0],          choice of the manufacturer. It has to
bit-string                [4],          be chosen so that, together with the
double-long               [5],          logical_name, an unambiguous
double-long-unsigned      [6],          interpretation of the value is possible.
octet-string              [9],
visible-string            [10],
integer                   [15],
long                      [16],
unsigned                  [17],
long-unsigned             [18],
long64                    [20],
long64-unsigned           [21],
float32                   [23],
float64                   [24]
}
```

When instead of a “Data” object a “Register” object is used, (with the scaler_unit attribute not used or with scaler = 0, unit = 255) then the data types allowed for the value attribute of the “Data” interface class are allowed.

scaler_unit

Provides information on the unit and the scaler of the value.

```
scal_unit_type: structure
{
scaler,
unit
}
```

scaler: integer — This is the exponent (to the base of 10) of the multiplication factor.
REMARK If the value is not numerical, then the scaler shall be set to 0.

unit: enum — Enumeration defining the physical unit; for details see below.

Method description

reset (data)	This method forces a reset of the object. By invoking this method, the value is set to the default value. The default value is an instance specific constant. data ::= integer(0)

unit ::= enum

Code	*// Unit*		*Quantity*	*Unit name*	*SI definition (comment)*
(1)	a	//	time	year	
(2)	mo	//	time	month	
(3)	wk	//	time	week	7*24*60*60 s
(4)	d	//	time	day	24*60*60 s
(5)	h	//	time	hour	60*60 s
(6)	min.	//	time	min	60 s
(7)	s	//	time (t)	second	s
(8)	°	//	(phase) angle	degree	rad*180/π
(9)	°C	//	temperature (T)	degree celsius	K-273.15
(10)	currency	//	(local) currency		
(11)	m	//	length (l)	metre	m
(12)	m/s	//	speed (v)	metre per second	m/s
(13)	m^3	//	volume (V) r_V , meter constant or pulse value (volume)	cubic metre	m^3
(14)	m^3	//	corrected volume	cubic metre	m^3
(15)	m^3/h	//	volume flux	cubic metre per hour	$m^3/(60*60s)$
(16)	m^3/h	//	corrected volume flux	cubic metre per hour	$m^3/(60*60s)$
(17)	m^3/d	//	volume flux		$m^3/(24*60*60s)$
(18)	m^3/d	//	corrected volume flux		$m^3/(24*60*60s)$
(19)	l	//	volume	litre	10^{-3} m^3
(20)	kg	//	mass (m)	kilogram	
(21)	N	//	force (F)	newton	
(22)	Nm	//	energy	newtonmeter	J = Nm = Ws
(23)	Pa	//	pressure (p)	pascal	N/m^2
(24)	bar	//	pressure (p)	bar	10^5 N/m^2
(25)	J	//	energy	joule	J = Nm = Ws
(26)	J/h	//	thermal power	joule per hour	J/(60*60s)
(27)	W	//	active power (P)	watt	W = J/s
(28)	VA	//	apparent power (S)	volt-ampere	
(29)	var	//	reactive power (Q)	var	
(30)	Wh	//	active energy r_W , active energy meter constant or pulse value	watt-hour	W*(60*60s)
(31)	VAh	//	apparent energy r_S , apparent energy meter constant or pulse value	volt-ampere-hour	VA*(60*60s)
(32)	varh	//	reactive energy r_B , reactive energy meter constant or pulse value	var-hour	var*(60*60s)
(33)	A	//	current (I)	ampere	A
(34)	C	//	electrical charge (Q)	coulomb	C = As
(35)	V	//	voltage (U)	volt	V
(36)	V/m	//	electric field strength (E)	volt per metre	V/m

Code	*// Unit*		*Quantity*	*Unit name*	*SI definition (comment)*
(37)	F	//	capacitance (C)	farad	C/V = As/V
(38)	Ω	//	resistance (R)	ohm	Ω = V/A
(39)	$\Omega m^2/m$	//	resistivity (ρ)		Ωm
(40)	Wb	//	magnetic flux (Φ)	weber	Wb = Vs
(41)	T	//	Magnetic flux density (B)	tesla	Wb/m^2
(42)	A/m	//	magnetic field strength (H)	ampere per metre	A/m
(43)	H	//	inductance (L)	henry	H = Wb/A
(44)	Hz	//	frequency (f, ω)	hertz	1/s
(45)	1/(Wh)	//	R_W, active energy meter constant or pulse value		
(46)	1/(varh)	//	R_B, reactive energy meter constant or pulse value		
(47)	1/(VAh)	//	R_S, apparent energy meter constant or pulse value		
(48)	V^2h	//	volt-squared hour r_{U2h}, volt-squared hour meter constant or pulse value	volt-squared-hours	$V^2(60*60s)$
(49)	A^2h	//	ampere-squared hour r_{I2h}, ampere-squared hour meter constant or pulse value	ampere-squared-hours	$A^2(60*60s)$
(50)	kg/s	//	mass flux	kilogram per second	kg/s
(51)	S, mho	//	conductance	siemens	1/Ω
(52)	K	//	temperature (T)	kelvin	
(53)	$1/(V^2h)$	//	R_{U2h}, volt-squared hour meter constant or pulse value		
(54)	$1/(A^2h)$	//	R_{I2h}, ampere-squared hour meter constant or pulse value		
(55)	$1/m^3$	//	R_V, meter constant or pulse value (volume)		
(56)		//	percentage	%	
(57)	Ah	//	ampere-hours	Ampere-hour	
(60)	Wh/m^3	//	energy per volume	$3,6*10^3$ J/m^3	
(61)	J/m^3	//	calorific value, wobbe		
(62)	Mol %	//	molar fraction of gas composition	mole percent	(Basic gas composition unit)
(63)	g/m^3	//	mass density, quantity of material		(Gas analysis, accompanying elements)
(64)	Pa s	//	dynamic viscosity	pascal second	(Characteristic of gas stream)
....					
(253)		//	reserved		
(254)	other	//	other unit		
(255)	count	//	no unit, unitless, count		

Examples of values:

Value	Scaler	Unit	Data
263788	-3	m^3	263,788 m^3
593	3	Wh	593 kWh
3467	0	V	3467 V

5.3 Extended register (class_id: 4)

Instances of an “Extended register” class store a process value with its associated status, unit, and time information. The extended register object knows the nature of the process value. The nature of the value is described by the attribute “logical_name” using the OBIS identification system.

Extended register		**0...n**	**class_id = 4, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. value	(dyn.)	CHOICE			
3. scaler_unit	(static)	scal_unit_type			
4. status	(dyn.)	CHOICE			
5. capture_time	(dyn.)	octet-string			
Specific method(s)		***m/o***			
1. reset (data)		o			

Attribute description

For the definition of the attributes *value* and *scaler_unit*, see description of class "Register".

logical_name	Identifies the “Extended register” object instance. See D.2.2.1.

status	Provides “Extended register” specific status information. The meaning of the elements of the status shall be provided for each “Extended register” object instance.
	CHOICE
	{
	--simple data types
	null-data [0],
	bit-string [4],
	double-long-unsigned [6],
	octet-string [9],
	visible-string [10],
	unsigned [17],
	long-unsigned [18],
	long64-unsigned [21]
	}
	Def. — Depending on the status type definition.
	The data type and the encoding depend on the instantiation and possibly on the choice of the manufacturer. For the interpretation, extra information from the manufacturer may be necessary.

capture_time	Provides an “Extended register” specific date and time information showing when the value of the attribute "value" has been captured. octet-string, formatted as set in 4.4.1 for *date_time*

Method description

reset (data)	This method forces a reset of the object. By invoking this method, the attribute value is set to the default value. The default value is an instance specific constant. The attribute capture_time is set to the time of the reset execution. data ::= integer(0)

5.4 Demand register (class_id: 5)

Instances of a “Demand register” class store a demand value with its associated status, unit, and time information. The demand register measures and computes its *current_average_value* periodically. The time interval *T* over which the demand is measured or computed is defined by specifying “number_of_periods” and “period”.

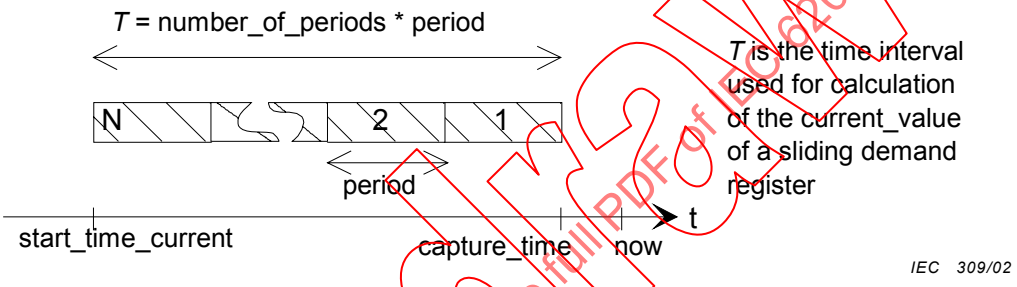


Figure 5 – The attributes when measuring sliding demand

The demand register delivers two types of demand: the current_average_value and the last_average_value (see Figure 6 and Figure 7).

The demand register knows its type of process value, which is described in “logical name" using the OBIS identification system.

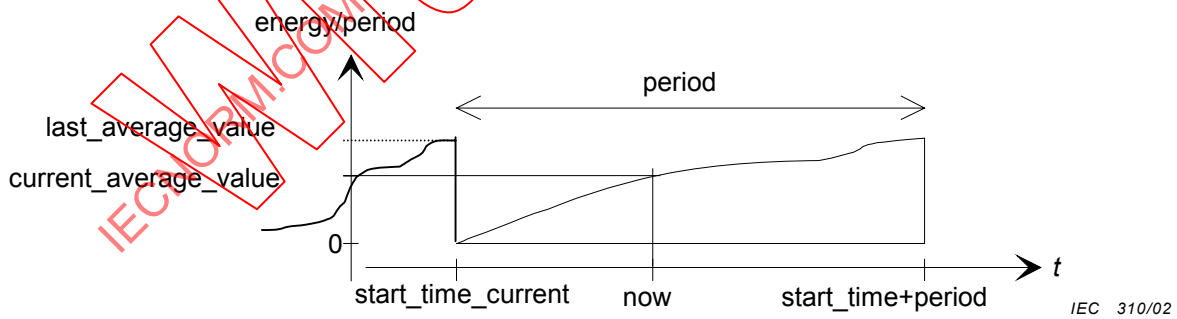


Figure 6 – The attributes when measuring current_average_value if number of periods is 1

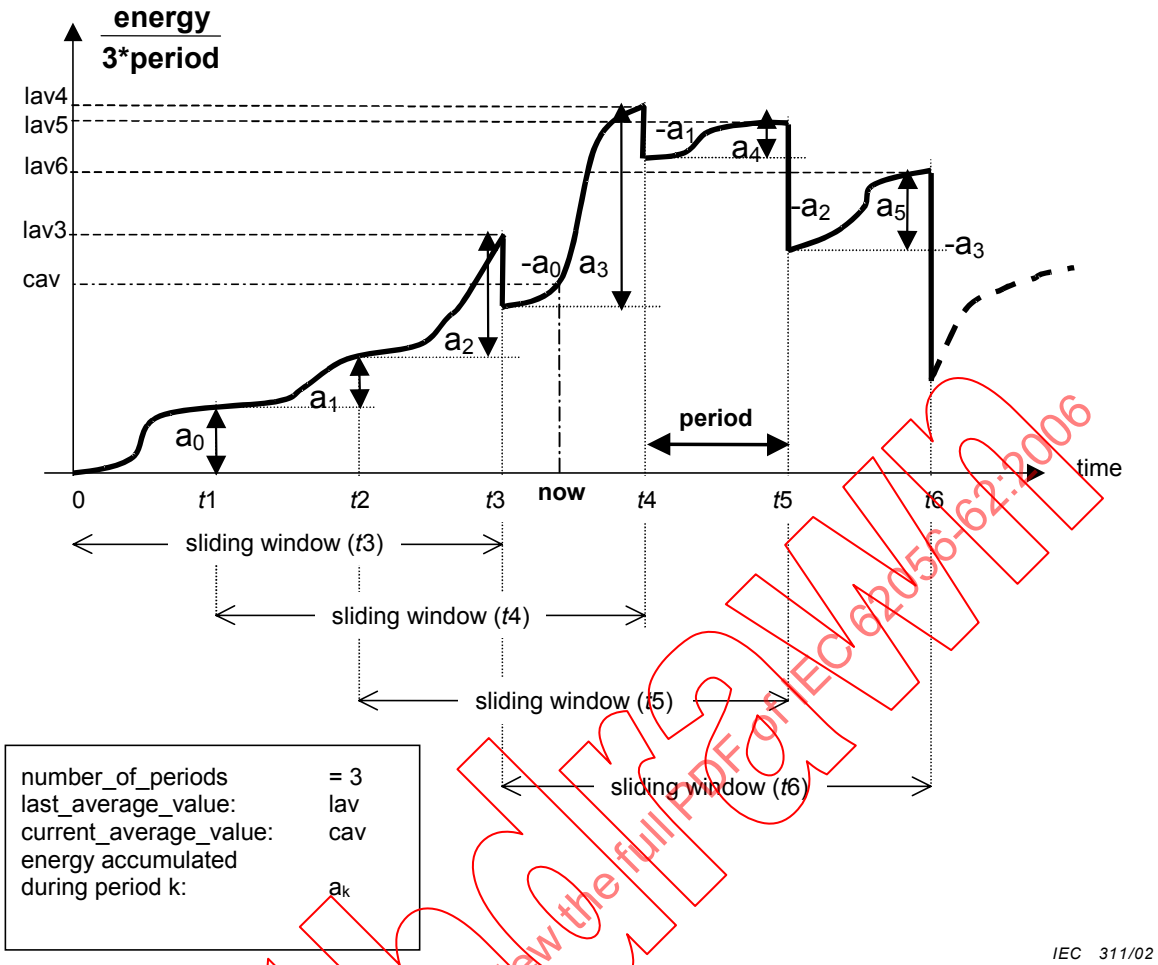


Figure 7 – The attributes if the number of periods is 3

Demand register		**0...n**	**class_id = 5, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. current_average_value	(dyn.)	CHOICE			0
3. last_average_value	(dyn.)	CHOICE			0
4. scaler_unit	(static)	scal_unit_type			
5. status	(dyn.)	CHOICE			
6. capture_time	(dyn.)	octet-string			
7. start_time_current	(dyn.)	octet-string			
8. period	(static)	double-long-unsigned	1		
9. number_of_periods	(static)	long-unsigned	1		1
Specific method(s)		***m/o***			
1. reset (data)		o			
2. next_period (data)		o			

Attribute description

For the attribute *scaler_unit*, see description of class “Register”.

logical_name	Identifies the “Demand register” object instance. See D.2.2.1.	
current_average_ value	Provides the current value (running demand) of the energy accumulated since start_time divided by number_of_periods*period. NOTE If another *quantity* than energy is measured, other calculation methods may apply (for example for calculating average values of voltage or current). CHOICE	The data type of the value depends on the instantiation defined by “logical_name” and possibly on the choice of the manufacturer. The type has to be chosen so that, together with the logical_name, unambiguous interpretation of the value is possible. For data types, see “Register” class, value attribute.
last_average_value	Provides the value of the accumulated energy (over the last number_of_periods*period) divided by number_of_periods*period. The energy of the current (not terminated) period is not considered by the calculation. NOTE If another *quantity* than energy is measured, other calculation methods may apply (for example for calculating average values of voltage or current). CHOICE	For data types, see “Register” class, value attribute.
status	Provides “Demand register” specific status information. The data type and the encoding depend on the instantiation and possibly on the choice of the manufacturer. For the interpretation, extra information from the manufacturer may be necessary. CHOICE ***Def.***	For data types, see “Extended register” class, status attribute. Depending on the status type definition.
capture_time	Provides the date and time when the last_average_value has been calculated. octet-string, formatted as set in 4.4.1 for *date_time*	
start_time_current	Provides the date and time when the measurement of the current_average_value has been started. octet-string, formatted as set in 4.4.1 for *date_time*	
period	Period is the interval between two successive updates of the last_average_value. (number_of_periods*period is the denominator for the calculation of the demand). double-long-unsigned The behaviour of the meter after writing a new value to this attribute shall be specified by the manufacturer.	Measuring period in seconds

number_of_periods	The number of periods used to calculate the last_average_value. number_of_periods >= 1	
	long-unsigned	number_of_periods > 1 indicates that the last_average_value represents “sliding demand”. number_of_periods = 1 indicates that the last_average_value represents "block demand".
	The behaviour of the meter after writing a new value to this attribute shall be specified by the manufacturer.	

Method description

reset (data)	This method forces a reset of the object. Activating this method provokes the following actions: - the current period is terminated; - the current_average_value and the last_average_value are set to their default values; - the capture_time and the start_time_current are set to the time of the execution of reset(data). data ::= integer(0)
next_period (data)	This method is used to trigger the regular termination (and restart) of a period. Closes (terminates) the current measuring period. Updates capture_time and start_time and copies current_average_value to last_average_value, sets current_average_value to its default value. Starts the next measuring period. REMARK The old last_average_value (and capture_time) can be read during the time “period”. The old current_average_value is not available any more at the interface. data ::= integer(0)

5.5 Register activation (class_id: 6)

Instances of the “Register activation” class are used to handle different tariffication structures. To each “Register activation” object, groups of “Register”, “Extended register” or “Demand register” objects, modelling different kind of quantities (for example active energy, active demand, reactive energy, etc.) are assigned. Subgroups of these registers, defined by the activation_masks define different tariff structures (for example day tariff, night tariff). One of these activation masks, the active_mask, defines which subset of the registers, assigned to the “Register activation” object instance is active. Registers, which are not defined in the register_assignment attribute of any “Register activation” object, are always enabled by default.

Register activation		**0...n**	**class_id = 6, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. register_assignment	(static)	array			
3. mask_list	(static)	array			
4. active_mask	(dyn.)	octet-string			

Specific method(s)	***m/o***
1. add_register (data)	o
2. add_mask (data)	o
3. delete_mask (data)	o

Attribute description

logical_name	Identifies the “Register activation” object instance. See D.2.1.9.

register_assignment	Specifies an ordered list of COSEM objects assigned to the “Register activation” object. The list may contain different kinds of COSEM objects, for example “Register”, “Extended register” or “Demand register”.
	
	array object_definition
	
	object_definition ::= structure
	
	{
	class_id: long-unsigned,
	logical_name: octet-string
	}

mask_list	Specifies a list of register activation masks. Each entry (mask) is identified by its mask_name and contains an array of indices referring to the registers assigned to the mask (the first object in register_assignment is referenced by index 1, the second object by index 2, ...,).
	
	array register_act_mask
	
	register_act_mask ::= structure
	{
	mask_name: octet-string,
	index_list: index_array
	}
	
	index_array ::= array unsigned
	mask_name has to be uniquely defined within the object.

active_mask	Defines the currently active mask. The mask is defined by its mask_name (see mask_list). octet-string This is a mask_name from the mask_list. The active_mask defines the registers currently enabled; all other registers listed in the register_assignment are disabled.

Method description

add_register (data)	Adds one more register to the attribute register_assignment. The new register is added at the end of the array; i.e. the newly added register has the highest index. The indices of the existing registers are not modified.
	data ::= structure
	{
	class_id: long-unsigned,
	logical_name: octet-string
	}

add_mask (data)	Adds another mask to the attribute mask_list. If there exists already a mask with the same name, the existing mask will be overwritten by the new mask. data ::= register_act_mask (see above)
delete_mask (data)	Deletes a mask from the attribute mask_list. The mask is defined by its mask name. data ::= octet-string (mask_name)

5.6 Profile generic (class_id: 7)

The “Profile generic” class defines a generalized concept to store dynamic process values of capture objects. Capture objects are appropriate attributes or element of (an) attribute(s) of COSEM objects. The capture objects are collected periodically or occasionally.

A profile has a buffer to store the captured data. To retrieve only a part of the buffer, either a value range or an entry range may be specified, asking to retrieve all entries whose values or entry numbers fall within the given range.

The list of capture objects defines the values to be stored in the buffer (using the method *capture*). The list is defined statically to ensure homogenous buffer entries (all entries have the same size and structure). If the list of capture objects is modified, the buffer is cleared. If the buffer is captured by other “Profile generic” objects, their buffer is cleared as well, to guarantee the homogeneity of their buffer entries.

The buffer may be defined as sorted by one of the registers or by a clock, or the entries are stacked in a “last in first out” order. So for example, it is very easy to build a “maximum demand register” with a one entry deep sorted profile capturing and sorted by a demand register. It is just as simple to define a profile retaining the three largest values of some period.

The size of profile data is determined by three parameters:

a) the number of entries filled. This will be zero after clearing the profile;

b) the maximum number of entries to retain. If all entries are filled and a capture () request occurs, the least important entry (according to the requested sorting method) will get lost. This maximum number of entries may be specified. Upon changing it, the buffer will be adjusted;

c) the physical limit for the buffer. This limit typically depends on the objects to capture. The object will reject an attempt of setting the maximum number of entries that is larger than physically possible.

Profile generic		**0...n**	**class_id = 7, version = 1**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. buffer	(dyn.)	compact array or array			x
3. capture_objects	(static)	array			
4. capture_period	(static)	double-long-unsigned			x
5. sort_method	(static)	enum			x
6. sort_object	(static)	object_definition			x
7. entries_in_use	(dyn.)	double-long-unsigned	0		0
8. profile_entries	(static)	double-long-unsigned	1		1

Specific method(s)	***m/o***
1. reset (data)	o
2. capture (data)	o
3. reserved from previous versions	o
4. reserved from previous versions	o

Attribute description

logical_name Identifies the “Profile generic” object instance. For examples, see D.2.1.13, D.2.1.26, D.2.1.29, D.2.1.30, D.2.1.36, D.2.2.3.

buffer The buffer attribute contains a sequence of entries. Each entry contains values of the captured objects (as they would be returned to a GET or READ.request).

compact-array or array entry

```
entry ::=
    structure
    {
        CHOICE
        {
        --simple data types
            null-data                [0],
            boolean                  [3],
            bit-string               [4],
            double-long              [5],
            double-long-unsigned     [6],
            octet-string             [9],
            visible-string           [10],
            bcd                      [13],
            integer                  [15],
            long                     [16],
            unsigned                 [17],
            long-unsigned            [18],
            long64                   [20],
            long64-unsigned          [21],
            enum                     [22],
            float32                  [23],
            float64                  [24],
            date-time                [25],
            date                     [26],
            time                     [27],
        --complex data types
            array                    [1],
            structure                [2],
            compact-array            [19]
    }
```

	The number and the order of the elements of the structure holding the entries is the same as in the definition of the capture_objects. The buffer is filled by auto captures or by subsequent calls of the method (capture). The sequence of the entries within the array is ordered according to the sort method specified. Default: The buffer is empty after reset. REMARK 1 Reading the entire buffer delivers only those entries, which are "in use". REMARK 2 The value of a captured object may be replaced by "null-data" if it can be unambiguously recovered from the previous value (e.g. for time: if it can be calculated from the previous value and capture_period; or for a value: if it is equal to the previous value). **selective access** (see 4.2) to the attribute buffer may be available (optional). The selective access parameters are as defined below.

capture_objects	Specifies the list of capture objects that are assigned to the object instance. Upon a call of the capture (data) method or automatically in defined intervals, the selected attributes are copied into the buffer of the profile.
	
	array capture_object_definition
	capture_object_definition ::= structure
	{
	class_id: long-unsigned,
	logical_name: octet-string,
	attribute_index: integer,
	data_index: long-unsigned
	}
	- where attribute_index is a pointer to the attribute within the object. attribute_index 1 refers to the first attribute (i.e. the logical_name), attribute_index 2 to the 2nd, etc.); attribute_index 0 refers to all public attributes;
	- where data_index is a pointer selecting a specific element of the attribute. The first element in the attribute structure is identified by data_index 1. If the attribute is not a structure, then the data_index has no meaning. If the capture object is the buffer of a profile, then the data_index identifies the captured object of the buffer (i.e. the column) of the inner profile.
	
	data_index 0: references the whole attribute

capture_period	>= 1: Automatic capturing assumed. Specifies the capturing period in seconds. 0: No automatic capturing; capturing is triggered externally or capture events occur asynchronously.
sort_method	If the profile is unsorted, it works as a "first in first out" buffer (it is hence sorted by capturing, and not necessarily by the time maintained in the clock object). If the buffer is full, the next call to capture () will push out the first (oldest) entry of the buffer to make space for the new entry. If the profile is sorted, a call to capture () will store the new entry at the appropriate position in the buffer, moving all following entries and probably losing the least interesting entry. If the new entry would enter the buffer after the last entry and if the buffer is already full, the new entry will not be retained at all.

	enum	(1) fifo (first in first out), (2) lifo (last in first out), (3) largest, (4) smallest, (5) nearest_to_zero, (6) farest_from_zero
	Def.	fifo
sort_object	If the profile is sorted, this attribute specifies the register or clock that the ordering is based upon.	
	capture_object_definition	See above.
	Def.	no object to sort by (only possible with sort_method fifo or lifo)
entries_in_use	Counts the number of entries stored in the buffer. After a call of reset (), the buffer does not contain any entries, and this value is zero. Upon each subsequent call of capture (), this value will be incremented up to the maximum number of entries that will get stored (see profile_entries).	
	double-long-unsigned	0...profile_entries
	Def.	0
profile_entries	Specifies how many entries shall be retained in the buffer.	
	double-long-unsigned	1...(limited by physical size)
	Def.	1

Parameters for selective access to the buffer attribute

Access selector value	Parameter	Comment
1	range_descriptor	Only buffer elements corresponding to the range_descriptor shall be returned in the response.
2	entry_descriptor	Only buffer elements corresponding to the entry_descriptor shall be returned in the response.

range_descriptor ::= structure
{

restricting_object	capture_object_definition	Defines the capture_object restricting the range of entries to be retrieved. Only simple data types are allowed.
from_value	CHOICE {--simple data types double-long [5], double-long-unsigned [6], octet-string [9], visible-string [10], integer [15], long [16], unsigned [17], long-unsigned [18], long64 [20], long64-unsigned [21], float32 [23], float64 [24], date-time [25], date [26], time [27] }	Oldest or smallest entry to retrieve
to_value	CHOICE { see above }	Newest or largest entry to retrieve
selected_values	array capture_object_definition	List of columns to retrieve. If the array is empty (has no entries), all captured data are returned. Otherwise, only the columns specified in the array are returned. The type *capture_object_definition* is specified above (*capture_objects*)

}
entry_descriptor ::= structure
{

from_entry	double-long-unsigned	first entry to retrieve,
to_entry	double-long-unsigned	last entry to retrieve to_entry == 0: highest possible entry,
from_selected_value	long-unsigned	index of first value to retrieve,
to_selected_value	long-unsigned	index of last value to retrieve to_selected_value == 0: highest possible selected_value

}
NOTE from_entry and to_entry identify the lines, from_selected_value to_selected_value identify the columns of the buffer to be retrieved.

Method description

reset (data)	Clears the buffer. The buffer has no valid entries afterwards; entries_in_use is zero after this call. This call does not trigger any additional operations of the capture objects. Specifically, it does not reset any captured buffers or registers. data ::= integer(0)
capture (data)	Copies the values of the objects to capture into the buffer by reading each capture object. Depending on the sort_method and the actual state of the buffer this produces a new entry or a replacement for the less significant entry. As long as not all entries are already used, the entries_in_use attribute will be incremented. This call does not trigger any additional operations within the capture objects such as capture () or reset (). Note, that if more than one attributes of an object need to be captured, they have to be defined one by one on the list of capture objects. If the attribute_index = 0, all attributes are captured. data ::= integer(0)

Behaviour of the object after modification of certain attributes

	Any modification of one of the capture_objects describing the static structure of the buffer will automatically call a reset () and this call will propagate to all other profiles capturing this profile. If writing to profile_entries is attempted with a value too large for the buffer, it will be rejected.

Restrictions

When defining the capture objects, circular reference to the profile shall be avoided.

Profile used to define a subset of preferred readout values

By setting profile_entries to 1, a “Profile generic” object can be used to define a set of preferred readout values. See also D.2.1.13.

Setting capture_period to 1 ensures that the values are updated every second.

5.7 Clock (class_id: 8)

An instance of the “Clock” interface class handles all information that is related to date and time, including leap years and the deviation of the local time to a generalized time reference (Greenwich Mean Time, GMT). The deviation from the local time to the generalized time reference can change depending on the season (e.g. summertime vs. wintertime). The interface to an external client is based on date information specified in day, month and year, time information given in hundredths of seconds, seconds, minutes and hours and the deviation from the local time to the generalized time reference.

It also handles the daylight saving function in that way; i.e. it modifies the deviation of local time to GMT depending on the attributes. The start and end point of that function is normally set once. An internal algorithm calculates the real switch point depending on these settings.

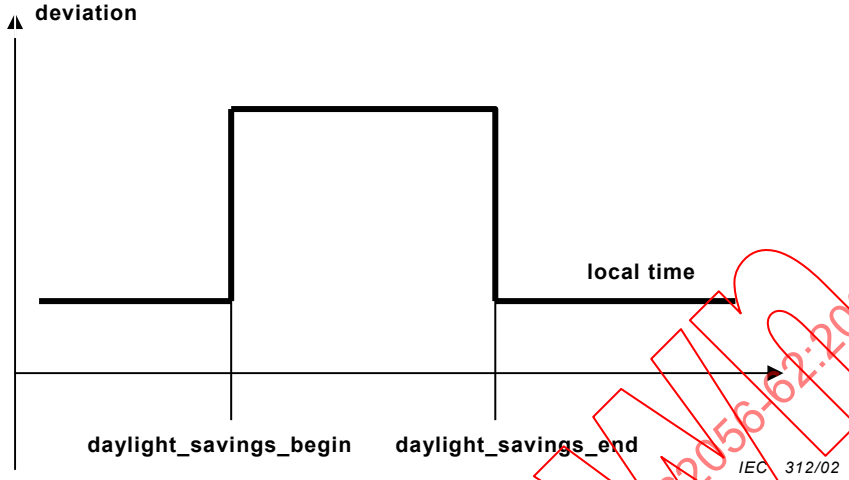


Figure 8 – The generalized time concept

Clock		**0…1**	**class_id = 8, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. time	(dyn.)	octet-string			
3. time_zone	(static)	long			
4. status	(dyn.)	unsigned			
5. daylight_savings_begin	(static)	octet-string			
6. daylight_savings_end	(static)	octet-string			
7. daylight_savings_deviation	(static)	integer			
8. daylight_savings_enabled	(static)	boolean			
9. clock_base	(static)	enum			
Specific method(s)		***m/o***			
1. adjust_to_quarter (data)		o			
2. adjust_to_measuring_period (data)		o			
3. adjust_to_minute (data)		o			
4. adjust_to_preset_time (data)		o			
5. preset_adjusting_time (data)		o			
6. shift_time (data)		o			

Attribute description

logical_name	Identifies the “Clock” object instance. See D.2.1.1.
time	Contains the meter’s local date and time, its deviation to GMT and the status. See 4.4.1. When this value is set, only specified fields of the date_time are changed. For example for setting the date without changing the time, all time relevant octets of the date_time shall be set to “not specified”. The clock_status shall always be set when writing the time. octet-string, formatted as set in 4.4.1 for *date_time*
time_zone	The deviation of local, normal time to GMT in minutes. long
status	The status is equal to the status read in *time*. See 4.4.1. unsigned, formatted as set in 4.4.1 for *clock_status*

daylight_savings_ begin	Defines the local switch date and time when the local time has to be deviated from the normal time. For generic definitions, wildcards are allowed. octet-string, formatted as set in 4.4.1 for *date_time*
daylight_savings_ end	See above. octet-string, formatted as set in 4.4.1 for *date_time*
daylight_savings_ deviation	Contains the number of minutes by which the deviation in generalized time must be corrected at daylight savings begin. integer Deviation range of up to ± 120 min
daylight_savings_ enabled	TRUE enables daylight savings function. boolean
clock_base	Defines where the basic timing information comes from. enum (0) not defined, (1) internal crystal, (2) mains frequency 50 Hz, (3) mains frequency 60 Hz, (4) GPS (global positioning system), (5) radio controlled

Method description

adjust_to_quarter (data)	Sets the meter's time to the nearest (+/-) quarter of an hour value (*:00, *:15, *:30, *:45). data ::= integer (0)
adjust_to_measuring _period (data)	Sets the meter's time to the nearest (+/-) starting point of a measuring period. data ::= integer (0)
adjust_to_minute (data)	Sets the meter's time to the nearest minute. If second_counter < 30 s, so second_counter is set to 0. If second_counter ≥ 30 s, so second_counter is set to 0, and minute_counter and all depending clock values are incremented if necessary. data ::= integer(0)
adjust_to_preset_ time (data)	This method is used in conjunction with the preset_adjusting_time method. If the meter's time lies between validity_interval_start and validity_interval_end, then time is set to preset_time. data ::= integer(0)

preset_adjusting_ time (data)	Presets the time to a new value (preset_time) and defines a validity_interval within which the new time can be activated.
	data ::= structure
	{
	preset_time: octet-string,
	validity_interval_start: octet-string,
	validity_interval_end: octet-string
	}
	all octet-strings formatted as set in 4.4.1 for *date_time*

shift_time (data)	Shifts the time by *n* (-900 <= *n* <= 900) s. data ::= long

5.8 Script table (class_id: 9)

The IC script table provides the possibility to trigger a series of actions by executing scripts using the execute (data) method.

For that purpose, script table contains a table of script entries. Each table entry (script) consists of a script_identifier and a series of action_specifications. An action_specification activates a method of a COSEM object or modifies attributes of a COSEM object within the logical device.

A specific script may be activated by other COSEM objects within the same logical device or from the outside.

If two scripts have to be executed at the same time instance, then the one with the smaller index is executed first.

Script table	**0...n**	**class_id = 9, version = 0**		
Attribute(s)	***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name (static) 2. scripts (static)	octet-string array			
Specific method(s)	***m/o***			
1. execute (data)	m			

Attribute description

logical_name	Identifies the “Script table” object instance. See D.2.1.5.

scripts	Specifies the different scripts, i.e. the lists of actions.
	
	array script
	
	script structure
	{
	script_identifier: long-unsigned,
	actions: array action_specification
	}
	The script_identifier 0 is reserved. If specified with an execute method, it results in a null script (no actions to perform).

	action_specification	structure
		{
		service_id: enum,
		class_id: long-unsigned,
		logical_name: octet-string,
		index: integer,
		parameter: service specific
		}
		where
		service_id: defines which action to be applied to the referenced object
		
		(1) write attribute,
		(2) execute specific method
		
		index: defines (with service_id 1) which attribute of the selected object is affected or (with service_id 2) which specific method is to be executed.
		
		The first attribute (logical_name) has index 1, the first specific method has index 1 as well.

NOTE The action_specification is limited to activate methods that do not produce any response (from the server to the client).

Method description	
execute (data)	Executes the script specified in parameter data. data long-unsigned If data matches one of the script_identifiers in the script table, then the corresponding action_specification is executed.

5.9 Schedule (class_id: 10)

The IC "Schedule" together with an object of the IC "Special days" table handles time and date driven activities within a device. The following picture gives an overview and shows the interactions between them:

Schedule:

Index	enable	action (script)	Switch _time	validity_ window	exec_weekdays							exec_specdays					date range	
					Mo	Tu	We	Th	Fr	Sa	Su	S1	S2	…	S8	S9	begin_ date	end_ date
120	Yes	xxxx:yy	06:00	0xFFFF	x	x	x	x	x	x							xx-04-01	xx-09-30
121	Yes	xxxx:yy	22:00	15	x	x	x	x	x								xx-04-01	xx-09-30
122	Yes	xxxx:yy	12:00	0						x							xx-04-01	xx-09-30
200	No	xxxx:yy	06:30		x	x	x	x	x	x							xx-04-01	xx-09-30
201	No	xxxx:yy	21:30		x	x	x	x	x								xx-04-01	xx-09-30
202	No	xxxx:yy	11:00							x							xx-04-01	xx-09-30

Special days table:

Index	special_day_date	day_id
12	xx-12-24	S1
33	xx-12-25	S3
77	97-03-31	S3

Schedule	**0...n**	**class_id = 10, version = 0**		
Attribute(s)	***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name (static) 2. entries (static)	octet-string array			
Specific method(s)	***m/o***			
1. enable/disable (data) 2. insert (data) 3. delete (data)	o o o			

Attribute description

logical_name	Identifies the “Schedule” object instance. See D.2.1.7.

entries	Specifies the scripts to be executed at given times. There is only one script that can be executed per entry.
	array schedule_table_entry
	schedule_table_entry structure
	{
	index: long-unsigned (1..9999),
	enable: boolean,
	script_logical_name: octet-string,
	script_selector: long-unsigned,
	switch_time: octet-string,
	validity_window: long-unsigned,
	exec_weekdays: bit-string,
	exec_specdays: bit-string
	begin_date: octet-string,
	end_date: octet-string
	}
	where:
	- script_logical_name: defines the logical name of the “Script table” object;
	- script_selector: defines the script_identifier of the script to be executed;
	- switch_time accepts wildcards to define repetitive entries. The format of the octet-string follows the rules set in 4.4.1 for *time*;
	- validity_window defines a period in minutes, in which an entry must be processed after power fail. (time between defined switch_time and actual power_up) 0xFFFF: the script must be processed any time;
	- exec_weekdays defines the days of the week on which the entry is valid;
	- exec_specdays perform the link to the IC “Special days table”, day_ID;
	- begin_date and end_date define the date period in which the entry is valid (wildcards are allowed). The format follows the rules set in 4.4.1 for *date*.

Method description

enable/disable (data)	Sets the disabled bit of range A entries to true and then enables the entries of range B.
	
	data ::= structure
	{
	firstIndexA,
	lastIndexA,
	firstIndexB,
	lastIndexB
	}
	firstIndexA — first index of the range that is disabled long-unsigned,
	
	lastIndexA — last index of the range that is disabled long-unsigned,
	
	firstIndexB — first index of the range that is enabled long-unsigned,
	
	lastIndexB — last index of the range that is enabled long-unsigned
	
	firstIndexA/B < lastIndexA/B: all entries of the range A/B are disabled/enabled
	
	firstIndexA/B == lastIndexA/B: one entry is disabled/enabled,
	
	firstIndexA/B > lastIndexA/B: nothing disabled/enabled,
	
	firstIndexA/B and lastIndexA/B > 9999: no entry is disabled/enabled

insert (data)	Inserts a new entry in the table. If the index of the entry exists already, the existing entry is overwritten by the new entry. entry — schedule_table_entry data: corresponding to entry

delete (data)	Deletes a range of entries in the table.
	
	data ::= structure
	{
	firstIndex,
	lastIndex
	}
	firstIndex — first index of the range that is deleted long-unsigned,
	
	lastIndex — last index of the range that is deleted long-unsigned
	
	firstIndex < lastIndex: all entries of the range A/B are deleted,
	firstIndex ::= lastIndex: one entry is deleted,
	firstIndex > lastIndex: nothing deleted

Remarks concerning "inconsistencies" in the table entries

- If the same script should be executed several times at a specific time instance, then it is executed only once.
- If different scripts should be executed at the same time instance, then the execution order is according to the "index". The script with the lowest "index" is executed first.

Recovery after power failure

After a power failure, the whole schedule is processed to execute all the necessary scripts that would get lost during a power failure. For this, the entries that were not executed during the power failure must be detected. Depending on the validity window attribute they are executed in the correct order (as they would have been executed in normal operation).

Handling of time changes

There are four different "actions" of time changes:

a) time setting forward;

b) time setting backwards;

c) time synchronization;

d) daylight saving action.

All these four actions need a different handling executed by the schedule in interaction with the time setting activity.

Time setting forward*

This is handled the same way as a power failure. All entries missed are executed depending on the validity window attribute. A (manufacturer specific defined) short time setting can be handled like time synchronization.

* Writing to the attribute “time” of the “Clock” object.

Time setting backward*

This results in a repetition of those entries that are activated during the repeated time. A (manufacturer specific defined) short time setting can be handled like time synchronization.

* Writing to the attribute “time” of the “Clock” object.

Time synchronization*

Time synchronization is used to correct small deviations between a master clock and the local clock. The algorithm is manufacturer specific. It shall guarantee that no entry of the schedule gets lost, or is executed twice. The validity window attribute has no effect, because all entries must be executed in normal operation.

* Using the method “adjust_to_quarter” of the “Clock” object.

Daylight saving

If the clock is put forward, then all scripts, which fall into the forwarding interval (and would therefore get lost) are executed.

If the clock is put back, re-execution of the scripts, which fall into the backwarding interval is suppressed.

5.10 Special days table (class_id: 11)

The interface class allows defining dates, which will override normal switching behaviour for special days. The interface class works in conjunction with the class "Schedule" or "Activity calendar" and the linking data item is day_id.

Special days table		0…1	class_id = 11, version = 0		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. entries	(static)	array			
Specific method(s)		***m/o***			
1. insert (data)		o			
2. delete (data)		o			

Attribute description

logical_name	Identifies the “Special days table” object instance. See D.2.1.6.

entries	Specifies a special day identifier for a given date.
	The date may have wildcards for repeating special days like Christmas.
	
	array spec_day_entry
	
	spec_day_entry structure
	{
	index: long-unsigned,
	specialday_date: octet-string,
	day_id: unsigned
	}
	where:
	- specialday_date formatting follows the rules set in 4.4.1 for *date*;
	- the range of the day_id must match the length of the bitstring exec_specdays in the related object of interface class “Schedule”.

Method description

insert (data)	Inserts a new entry in the table. entry spec_day_entry If a special day with the same index or with the same date as an already defined day is inserted, the old entry will be overwritten.
delete (data)	Deletes an entry in the table. index Index of the entry that shall be deleted. long-unsigned data ::= long-unsigned

5.11 Activity calendar (class_id: 20)

An instance of the “Activity calendar” class is typically used to handle different tariff structures. It is a definition of scheduled actions inside the meter, which follow the classical way of calendar based schedules by defining seasons, weeks… It can coexist with the more general object “Schedule” and can even overlap with it. If actions are scheduled for the same activation time in an object “Schedule” and in the object “Activity calendar”, the actions triggered by the “Schedule” object are executed first.

After a power failure, only the “last action” missed from the object “Activity calendar” is executed (delayed). This is to ensure proper tariffication after power up. If a “Schedule” object is present, then the missed “last action” of the “Activity calendar” must be executed at the correct time within the sequence of actions requested by the “Schedule” object.

The “Activity calendar” defines the activation of certain scripts, which can perform different activities inside the logical device. The interface to the object “Script table” is the same as for the object “Schedule” (see 5.9).

If an instance of the interface class “Special days table” (see 5.10) is available, relevant entries there take precedence over the “Activity calendar” object driven selection of a day profile. The day profile referenced in the “Special days table” activates the day_schedule of the day_profile_table in the “Activity calendar” object by referencing through the day_id.

Activity calendar		**0…1**	**class_id = 20, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. calendar_name_active	(static)	octet-string			
3. season_profile_active	(static)	array			
4. week_profile_table_active	(static)	array			
5. day_profile_table_active	(static)	array			
6. calendar_name_passive	(static)	octet-string			
7. season_profile_passive	(static)	array			
8. week_profile_table_passive	(static)	array			
9. day_profile_table_passive	(static)	array			
10. activate_passive_calendar_time	(static)	octet-string			
Specific method(s)		***m/o***			
1. activate_passive_calendar (data)		o			

Attribute description

Attributes called …_active are currently active, attributes called …_passive will be activated by the specific method activate_passive_calendar.

logical_name	Identifies the “Activity calendar” object instance. See D.2.1.8.
calendar_name	Typically contains an identifier, which is descriptive to the set of scripts, which are activated by the object.
season_profile	Contains a list of seasons defined by their starting date and a specific week_profile to be executed. The list is sorted according to season_start.

array season

season ::= structure
{
season_profile_name: octet-string,
season_start: octet-string,
week_name: octet-string
}
where:

- season_profile_name is a user defined name identifying the current seson_profile;
- season_start defines the starting time of the season, formatted as set in 4.4.1 for *date_time*.

  REMARK The current season is terminated by the season_start of the next season.

- week_name defines the week_profile active in this season

week_profile_table

Contains an array of week_profiles to be used in the different seasons. For each week_profile, the day_profile for every day of a week is identified.

```
array                week_profile

week_profile ::=     structure
                     {
                          week_profile_name:    octet-string,
                          monday:               day_id,
                          tuesday:              day_id,
                          wednesday:            day_id,
                          thursday:             day_id,
                          friday:               day_id,
                          saturday:             day_id,
                          sunday:               day_id
                     }
                     day_id: unsigned
```

where:
- week_profile_name is a user defined name identifying the current week_profile;
- Monday defines the day_profile valid on Monday;
- ...
- Sunday defines the day_profile valid on Sunday.

day_profile_table

Contains an array of day_profiles, identified by their day_id. For each day_profile, a list of scheduled actions is defined by a script to be executed and the corresponding activation time (start_time). The list is sorted according to start_time.

```
array                day_profile

day_profile ::=      structure
                     {
                       day_id:              unsigned,
                       day_schedule:        array day_profile_action
                     }

day_profile_action ::=  structure
                     {
                       start_time:                  octet-string,
                       script_logical_name:         octet-string,
                       script_selector:             long-unsigned
                     }
```

where:
- day_id is a user defined identifier, identifying the current day_profile;
- start_time: defines the time when the script is to be executed (no wildcards); the format follows the rules set in 4.4.1for *time*;
- script_logical_name: defines the logical name of the “Script table” object;
- script_selector: defines the script_identifier of the script to be executed.

activate_passive_ calendar_time	Defines the time when the object itself calls the specific method activate_passive_calendar. A definition with "not specified" notation in all fields of the attribute will deactivate this automatism. Partial "not specified" notation in just some fields of date and time are not allowed. octet-string, formatted as set in 4.4.1 for *date_time*

Method description

activate_passive_ calendar(data)	This method copies all attributes called …_passive to the corresponding attributes called …_active, data ::= integer(0)

5.12 Association LN (class_id: 15)

COSEM logical devices able to establish application associations within a COSEM context using logical name referencing, model the associations through instances of the "Association LN" class. A COSEM logical device has one instance of this IC for each association the device is able to support.

Association LN		**0…MaxNbofAss.**	**class_id = 15, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. object_list	(static)	object_list_type			
3. associated_partners_id		associated_partners_type			
4. application_context_name		application-context-name			
5. xDLMS_context_info		xDLMS-context-type			
6. authentication_mechanism _name		mechanism-name			
7. LLS_secret		octet-string			
8. association_status		enum			
Specific method(s)		***m/o***			
1. reply_to_HLS_authentication (data)		o			
2. change_HLS_secret (data)		o			
3. add_object (data)		o			
4. remove_object (data)		o			

Attribute description

logical_name	Identifies the “Association LN” object instance. See D.2.1.22.

object_list	Contains the list of visible COSEM objects with their class_id, version, logical name and the access rights to their attributes and methods within the given application association.
	
	object_list_type ::= array object_list_element
	
	object_list_element ::= structure
	{
	class_id: long-unsigned,
	version: unsigned,
	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,
	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
	}
	where:
	- the attribute_access_descriptor and the method_access_descriptor always contain all implemented attributes or methods;
	- access_selectors contain a list of the supported selector values.
	
	selective access (see 4.2) to the attribute object_list may be available (optional). The selective access parameters are as defined below.

associated_partners_id

Contains the identifiers of the COSEM client and the COSEM server (logical device) application processes within the physical devices hosting these processes, which belong to the application association modelled by the “Association LN” object.

```
associated_partners_type ::= structure
{
        client_SAP:         integer,
        server_SAP          long-unsigned:
}
```

The range for the client_SAP is 0…0x7F.
The range for the server_SAP is 0x000…0x3FFF.

---

application_context_name

In the COSEM environment, it is intended that an application context pre-exists and is referenced by its name during the establishment of an application association. This attribute contains the name of the application context for that association.

The application context name is specified as OBJECT IDENTIFIER in 7.3.7.1 of IEC 62056-53.

The application_context_name attribute includes the arc labels of the OBJECT IDENTIFIER.

```
application_context_name ::= structure
        {
          joint-iso-ctt-element:              unsigned,
          country-element:                    unsigned,
          country-name-element:               long-unsigned,
          identified-organization-element:    unsigned,
          DLMS-UA-element:                    unsigned,
          application-context-element:        unsigned,
          context-id-element:                 unsigned
        }
```

For existing implementations, the attribute may hold the value of the OBJECT IDENTIFIER encoded in BER, as an octet string. See Clause C.2 of IEC 62056-53.

```
application_context_name ::= octet-string
// holds the value of the OBJECT identifier encoded in BER
```

Example:

In case of context_id(1) the A-XDR encoding as a structure (all values are hexadecimal):

02 07 11 02 11 10 12 02 F4 11 05 11 08 11 01 11 01

The A-XDR encoding as an octet-string, holding the value of the OBJECT IDENTIFIER encoded in BER (all values are hexadecimal):

09 07 60 85 74 05 08 01 01

xDLMS_context_info

Contains all the necessary information on the xDLMS context for the given association.

xDLMS-context-type::= structure
{
conformance: bitstring(24),
max_receive_pdu_size: long-unsigned,
max_send_pdu_size: long-unsigned,
dlms_version_number: unsigned,
quality_of_service: integer,
cyphering_info: octet-string
}
where

- the conformance element contains the xDLMS conformance block supported by the server;
- the max_receive_pdu_size element contains the maximum length for an xDLMS APDU, expressed in bytes that the client may send. This is the same as the server-max-receive-pdu-size parameter of the DLMS-Initiate.response pdu (see IEC 62056-53);
- the max_send_pdu_size, in an active association contains the maximum length for an xDLMS APDU, expressed in bytes that the server may send. This is the same as the client-max-receive-pdu-size parameter of the DLMS-Initiate.request pdu (see IEC 62056-53);
- the dlms_version_number element contains the DLMS version number supported by the server;
- the quality_of_service element is not used;
- the cyphering_info, in an active association, contains the dedicated key parameter of the DLMS-Initiate.request pdu (See IEC 62056-53).

authentication_mechanism_name

Contains the name of the authentication mechaism for the association (see IEC 62056-53).

The authentication mechanism name is specified as an OBJECT IDENTIFIER in 7.3.7.2 of IEC 62056-53.

The authentication_mechanism_name attribute includes the arc labels of the OBJECT IDENTIFIER.

authentication_mechanism_name ::= structure
{
joint-iso-ctt-element: unsigned,
country-element: unsigned,
country-name-element: long-unsigned,
identified-organization-element: unsigned,
DLMS-UA-element: unsigned,
authentication-mechanism-name-element: unsigned,
mechanism-id-element: unsigned
}

For existing implementations, the attribute may hold the value of the OBJECT IDENTIFIER encoded in BER, as an octet string. See Clause C.2 of IEC 62056-53.

	authentication_mechanism_name ::= octet-string // holds the value of the OBJECT identifier encoded in BER Example: In case of mechanism_id(1) the A-XDR encoding as a structure (all values are hexadecimal): 02 07 11 02 11 10 12 02 F4 11 05 11 08 11 02 11 01 The A-XDR encoding as an octet-string, holding the value of the OBJECT IDENTIFIER encoded in BER (all values are hexadecimal): 09 07 60 85 74 05 08 02 01
LLS_secret	Contains the authentication value for the LLS authentication process.

association_status	Indicates the current status of the association, which is modelled by the object.
	
	association-status: enum
	{
	(0) non-associated,
	(1) association-pending,
	(2) associated
	}

A SET operation on an attribute of an association LN object becomes effective when this association object is used to establish a new association.

Parameters for selective access to the object_list attribute

- If no selective access is requested, (no Access_Selection_Parameters parameter is present in the GET request (.indication) service invocation for the object_list attribute) the corresponding .response (.confirmation) service shall contain all object_list_elements of the object_list attribute.
- When selective access is requested to the object_list attribute (the Access_Selection_Parameters parameter is present), the response shall contain a 'filtered' list of object_list_elements, as follows:

Access selector value	Service parameters	Comment
1	NULL	All information excluding the access_rights shall be included in the response.
2	class_list	Access by class. In this case, only those object_list_elements of the object_list shall be included in the response, which have a class_id equal to one of the class_id-s of the class-list. No access_right information is included. class_list ::= array class_id class_id ::= long-unsigned
3	object_Id_list	Access by object. The full information record of object instances on the object_Id_list shall be returned. object_Id_list ::= array object_Id

4	object_Id	The full information record of the required COSEM object instance shall be returned.
		
		object_Id ::= structure
		
		{
		class_id: long-unsigned,
		logical_name: octet-string
		}

Method description

reply_to_HLS_ authentication (data)	The remote invocation of this method delivers the client's "secretly" processed "challenge StoC" (f(StoC)) back to the server as the *data* service parameter of the invoked ACTION.request service. See 4.7.2. data ::= octet-string client's response to the challenge If the authentication is accepted, then the response (ACTION.confirm) contains Result == OK and the server's "secretly" processed "challenge CtoS" (f(CtoS)) back to the client in the *data* service parameter of the response service. data ::= octet-string server's response to the challenge If the authentication is not accepted, then the result parameter in the response shall contain a non-OK value, and no data shall be sent back.
change_HLS_secret (data)	Changes the HLS secret (e.g. encryption key). data ::= octet-string[a] new HLS secret
add_object (data)	Adds the referenced object to the object_list. data ::= object_list_element (see above)
remove_object (data)	Removes the referenced object from the object_list. data ::= object_list_element (see above)

[a] The structure of the "new secret" depends on the security mechanism implemented. The "new secret" may contain additional check bits and it may be encrypted.

5.13 Association SN (class_id: 12)

COSEM logical devices able to establish application associations within a COSEM context using short name references, model the associations through instances of the "Association SN" class. A COSEM logical device may have one instance of this IC for each association the device is able to support.

The **short_name** of the "Association SN object itself is fixed within the COSEM context. It is given in Clause C.3 as 0x**FA00**.

Association SN		**0...n**	**class_id = 12, version = 1**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. object_list	(static)	objlist_type			
Specific method(s)		***m/o***			
1. reserved from previous versions		o			
2. reserved from previous versions		o			
3. read_by_logicalname (data)		o			
4. get_attributes&methods (data)		o			
5. change_LLS_secret (data)		o			
6. change_HLS_secret (data)		o			
7. *reserved from previous versions*					
8. reply_to_HLS_authentication (data)		o			

Attribute description

logical_name	Identifies the “Association SN” object instance. See D.2.1.22.

object_list	Contains the list of all objects with their base_names (short_name), class_id, version and logical_name. The base_name is the DLMS objectName of the first attribute (logical_name).
	
	objlist_type ::= array objlist_element
	objlist_element ::= structure
	{
	base_name: long,
	class_id: long-unsigned,
	version: unsigned,
	logical_name: octet-string
	}
	
	selective access (see 4.2) to the attribute object_list may be available (optional). The selective access parameters are as defined below.

Parameters for selective access to the object_list attribute

Access selector value	Parameter	Comment
1	class_id: long-unsigned	Delivers the subset of the object_list for a specific class_id. For the response: data ::= objlist_type

2	structure
	{
	class_id: long-unsigned,
	logical_name: octet-string
	}
		Delivers the entry of the object_list for a specific class_id and logical_name.
		For the response: data ::= objlist_element

Method description

read_by_ logicalname (data)	Reads attributes for selected objects. The objects are specified by their class_id and their logical_name.
	
	data ::= array attribute_identification
	
	attribute_identification ::= structure
	{
	class_id: long-unsigned,
	logical_name: octet-string,
	attribute_index: integer
	}
	where attribute_index is a pointer (i.e. offset) to the attribute within the object.
	
	attribute_index 0 delivers all attributes[a], attribute_index 1 delivers the first attribute (i.e. logical_name), etc.).
	
	For the response: data is according to the type of the attribute.

get_attributes& methods(data)

Delivers information about the access rights to the attributes and methods within the actual association. The objects are specified by their class_id and their logical_name.

data ::= array object_identification

object_identification ::= structure
{
class_id: long-unsigned,
logical_name: octet-string
}

For the response
data ::= array access_description

access_description ::= structure

{
read_attributes: bit-string,
write_attributes: bit-string,
methods: bit-string
}
The position in the bit-string identifies the attribute/method (first position ↔ first attribute, first position ↔ first method) and the value of the bit specifies whether the attribute/method is available (bit set) or not available (bit clear).

NOTE Depending on the implementation, some attributes or methods of some objects may not be needed. In this case, such attributes or methods may not be accessible (neither read access, nor write access to attributes, no access to methods).

change_LLS_secret (data)

Changes the LLS secret (e.g. password).

data ::= octet-string new LLS secret

change_HLS_secret (data)

Changes the HLS secret (e.g. encryption key).

data ::= octet-string[b] new HLS secret

reply_to_HLS_ authentication (data)

The remote invocation of this method delivers the client's "secretly" processed "challenge StoC" (f(StoC)) back to the server as the *data* service parameter of the invoked Write.request service (see 4.7.2).

data ::= octet-string client's response to the challenge

If the authentication is accepted, then the response (Write.confirm) contains Result == OK and the server's "secretly" processed "challenge CtoS" (f(CtoS)) back to the client in the *data* service parameter of the response service.

data ::= octet-string server's response to the challenge

If the authentication is not accepted, then the result parameter in the response shall contain a non-OK value, and no data shall be sent back.

[a] If at least one attribute has no read access right under the current association, then a read_by_logicalname() to attribute index 0 reveals the error message "scope of access violation" (see IEC 61334-4-41, p. 221).
[b] The structure of the "new secret" depends on the security mechanism implemented. The "new secret" may contain additional check bits and it may be encrypted.

5.14 SAP assignment (class_id: 17)

The interface class “SAP assignment” contains the information on the assignment of the logical devices to their SAP-s (see IEC 62056-53).

SAP assignment		**0...1**	**class_id = 17, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. SAP_assignment_list	(static)	asslist_type			0
Specific method(s)		***m/o***			
1. connect_logical_device (data)		o			

Attribute description

logical_name	Identifies the “SAP assignment” objects instance. See D.2.1.23.

SAP_assignment_ list	Contains the list of all logical devices and their SAP addresses within the physical device.
	
	asslist_type ::= array asslist_element
	asslist_element ::= structure
	{
	SAP: long-unsigned,
	logical_device_name: octet-string
	}
	
	REMARKS
	- The actual addressing is performed by the supporting communication layers.
	- For the 3-layer, connection-oriented, HDLC based profile, see IEC 62056-46 and IEC 62056-53, Clause B.2.
	- For the TCP-UDP/IP based profile, see IEC 62056-47 and IEC 62056-53, Clause B.3.

Method description

connect_logical_ device (data)	Connects a logical device to a SAP. Connecting to SAP 0 will disconnect the device. More than one device cannot be connected to one SAP (exception SAP 0) data ::= asslist_element

5.15 Register monitor (class_id: 21)

This interface class allows defining a set of scripts (see 5.8) that are executed when the value of an attribute of a monitored register type object “Data”, “Register”, “Extended register”, Demand register, etc. crosses a set of threshold values.

The IC “Register monitor” requires an instantiation of the IC “Script table” in the same logical device.

Register monitor		**0...n**	**class_id = 21, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. thresholds	(static)	array			
3. monitored_value	(static)	value_definition			
4. actions	(static)	array			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the “Register monitor” object instance. See D.2.1.11 and D.2.2.20.
thresholds	Provides the threshold values to which the attribute of the referenced register is compared. array threshold threshold: The threshold is of the same type as the monitored attribute of the referenced object.

monitored_value	Defines which attribute of an object is to be monitored. Only values with simple data types are allowed.
	
	value_definition ::= structure
	{
	class_id: long-unsigned,
	logical_name: octet-string,
	attribute_index: integer
	}

actions	Defines the scripts to be executed when the monitored attribute of the referenced object crosses the corresponding threshold. The attribute “actions” has exactly the same number of elements as the attribute “thresholds”. The ordering of the action_items corresponds to the ordering of the thresholds (see above).
	
	array action_set
	
	action_set ::= structure
	{
	action_up: action_item,
	action_down: action_item
	}
	where:
	- action_up defines the action when the attribute value of the monitored register crosses the threshold in the upwards direction;
	- action_down defines the action when the attribute value of the monitored register crosses the threshold in the downwards direction
	
	action_item ::= structure
	{
	script_logical_name: octet-string,
	script_selector: long-unsigned
	}

5.16 Utility tables (class_id: 26)

An instance of the “Utility tables” class encapsulates ANSI C12.19 table data.

With this interface class definition, each “table” is represented as an instance. The specific instance is identified by its logical_name.

Utility tables		0...n	class_id = 26, version = 0		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. table_ID	(static)	long-unsigned			
3. length		double-long-unsigned			
4. buffer		octet-string			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the “Utility tables” object instance. See D.2.1.25.
table_ID	Table number. This table number is as specified in the ANSI standard and may be either a standard table or a manufacturer’s table.
length	Number of octets in table buffer.
buffer	Contents of table. **Selective access** (see 4.2) to the attribute buffer may be available (optional). The selective access parameters are as defined below.

Parameters for selective access to the buffer attribute

Access selector	Parameter	Comment
1	offset_access	Access to table by offset and count using offset_selector for parameter data.
2	index_access	Access to table by element id and number of elements using index_selector for parameter data.

offset_selector ::= structure
{

Offset	double-long-unsigned	offset in octets to the start of access area, relative to the start of the table
Count	long-unsigned	number of octets requested or transferred

}
index_selector ::= structure
{

Index	array of long-unsigned	sequence of indices to identify elements within the table’s hierarchy
Count	long-unsigned	number of elements requested or transferred. Values of count greater than 1 return up to that many elements. A value of zero, when given in the context of a request, refers to the entire sub-tree of the hierarchy starting at the selection point

}

5.17 Single action schedule (class_id: 22)

Many applications request periodic actions within a meter. These actions are not necessarily linked to tariffication (activity calendar or schedule). The IC "Single action schedule " models such actions.

Single action schedule		**0...n**	**class_id = 22, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	*(static)*	octet-string			
2. executed_script	*(static)*	script			
3. type	*(static)*	enum			
4. execution_time	*(static)*	array			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the "Single action schedule" object instance. See D.2.1.10.

executed_script	Contains the logical name of the "Script table" object and the script selector of the script to be executed.
	script structure
	{
	script_logical_name octet-string,
	script_selector long-unsigned
	}
	Script_logical_name and script_selector define the script to be executed.

type	enum (1) size of execution_time = 1; wildcard in *date* allowed, (2) size of execution_time = *n*; all *time* values are the same, wildcards in *date* not allowed, (3) size of execution_time = *n*; all *time* values are the same, wildcards in *date* are allowed, (4) size of execution_time = *n*; *time* values may be different, wildcards in *date* not allowed, (5) size of execution_time = *n*; *time* values may be different, wildcards in *date* are allowed

execution_time	Specifies the time of day the script is executed.
	array {octet-string, octet-string}
	The two octet-strings contain *time* and *date*, in this order.
	structure
	{
	time: octet-string,
	date: octet-string
	}
	time and *date* are formatted as defined in 4.4.1.
	WILDCARDS in "*time*" are not allowed; seconds and hundredths of seconds must be zero.

5.18 Register table (class_id: 61)

Instances of the “Register table” interface class store homogenous entries, identical attributes of multiple objects, which are all instances of the same interface class, and the value in value groups A to D and F of their logical name (OBIS code) is identical. The possible values in value group E are defined in IEC 62056-61 in a tabular form: the table header defines the common part of the OBIS code and each table cell defines one possible value of value group E. A “Register table” object may capture attributes of some or all of those objects.

NOTE 1 Some examples are the “UNIPEDE voltage dip quantities” table, see IEC 62056-61, Table 13, or the “Extended phase angle measurement” table, see IEC 62056-61, Table 11.

NOTE 2 If more complex functionality is needed, the “Profile generic” interface class can be used.

Register table		**0...n**	**class_id = 61, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. table_cell_values	(dyn.)	array			
3. table_cell_definition	(static)	structure			
4. scaler_unit	(static)	scaler_unit_type			
Specific method(s)		***m/o***			
1. reset (data)		o			
2. capture (data)		o			

Attribute description

logical_name

Identifies the “Register table” object instance.

When the format of the logical name is A.B.C.D.255.F; the values A to D and F define the common part of the logical name of the objects, the attributes of which are captured. Only one attribute of the objects concerned can be captured (for example the *value* attribute).

When the format of the logical name is A.B.98.10.x.255, several instances of the “Register table” class can be used to capture different attributes of the objects concerned. The value group E numbers the instances.

table_cell_values

Holds the value of the attributes captured, as they would be returned to a GET or Read .request to the individual attributes.

table_cell_values ::= compact array or array of table_cell_entry

table_cell_entry

```
CHOICE
{
--simple data types
null-data                [0],
bit-string               [4],
double-long              [5],
double-long-unsigned     [6],
octet-string             [9],
visible-string           [10],
bcd                      [13],
integer                  [15],
long                     [16],
unsigned                 [17],
long-unsigned            [18],
long64                   [20],
long64-unsigned          [21],
float32                  [23],
```

	float64 [24], --complex data types structure [2] } If the captured attribute is attribute_0, redundant values may be replaced by "null_data", if their value can be unambiguously recovered (for example scaler_unit).

table_cell_definition	Specifies the list of attributes captured in the register table.
	
	table_cell_definition ::= structure
	{
	class_id: long-unsigned,
	logical_name: octet_string,
	group_E_values: array of cell_identifier,
	{
	cell_identifier: unsigned
	}
	attribute_index integer
	}
	
	where:
	- class_id defines the common class_id of the objects the attributes of which are captured;
	- logical_name contains the common logical name of the objects, with E = 255 (wildcard);
	- group_E_values contain the list of cell identifiers, as defined in the respective table of IEC 62056-61;
	- attribute_index is a pointer to the attribute within the object. attribute_index 0 refers to all public attributes.
	
	If the logical name of the "Register table" object is in the format A.B.C.D.255.F and the defined attribute of all objects identified in the respective table in IEC 62056-61 are captured, then attribute 3 may not be accessible. In this case:
	- the class_id shall be 1 "Data", 3 "Register" or 4 "Extended register";
	- the logical name of the objects to be captured is defined by the logical name of the "Register table" object and the respective table in IEC 62056-61;
	- the attribute index shall be 2 (value).

scaler_unit	See the description of class "Register". In the case when "value" attributes of "Register" or "Extended register" objects are captured, the scaler_unit shall be common for all objects and this attribute shall hold a copy. If other attributes or interface classes are captured, the scaler_unit attribute has no meaning and shall be inaccessible.
Method description	
reset (data)	Clears the table_cell_values. It has no effect on the attributes captured. data ::= integer(0)
capture (data)	Copies the values of the attributes into the table_cell_values. If the attribute_index = 0, all attributes are captured.

Behaviour of the object after modification of the table_cell_definition attribute

Any modification to this attribute will automatically call the reset(data) method and this will propagate to all other profiles capturing this object.

If writing to table_cell_definition is attempted with a value too large the buffer holding the table_cell_values attribute, it will be rejected.

5.19 Status mapping (class_id: 63)

Instances of the “Status mapping” class store status words together with the mapping of each bit in the status word to positions in a reference status table.

Status mapping		**0...n**	**class_id = 63, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. status word	(dynamic)	CHOICE			
3. mapping_table	(static)	structure			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the “Status word” object instances. See D.2.1.29, D.2.1.30, D.2.1.34 and D.2.2.17.

status_word	Contains the current value of the status word.
	
	CHOICE
	{
	[4] bit-string,
	[6] double-long-unsigned,
	[9] octet-string,
	[10] visible-string,
	[17] unsigned,
	[18] long-unsigned,
	[21] long64-unsigned
	}
	The size of the status_word is n*8 bits, the maximum size is 65 536 bits.
	
	NOTE Manufacturers may choose any of the types listed above. However, the status word shall be always interpreted as a bit-string.

mapping_table	Contains the mapping of the status word to the positions in the reference table.
	
	mapping_table::= structure
	{ ref_table_id: unsigned,
	CHOICE
	{
	first_entry long-unsigned,
	array of table_entries
	}
	}
	
	table_entry long-unsigned
	where:
	- ref_table_id is the identifier of the reference status table
	
	NOTE Reference status tables are maintained by the DLMS UA.
	
	- first_entry is the entry in the reference table corresponding to bit 0 of the status word. Bit 1 corresponds to the next entry and so on, the last entry is defined by the length of the status word;
	- if the “array” choice is taken, the array of table entries maps the bits of the status word to the entries in the reference status table. The position in the array defines the position in the status word (the first position corresponds to bit 0).

6 Maintenance of the interface classes

Any modification of interface classes as described below in 6.2 will be recorded by moving the old or obsolete version of an interface class into Annex E.

Versions of interface classes prior to the establishment of this standard are kept by the DLMS UA only.

6.1 New interface classes

The DLMS UA reserves the right to be the exclusive administrator of interface classes.

6.2 New versions of interface classes

Any modification of an existing interface class affecting the transmission of service requests or responses results in a new version (version ::= version+1) and shall be documented accordingly. The following rules shall be followed:

a) new attributes and methods may be added;

b) existing attributes and methods may be invalidated BUT the indices of the invalidated attributes and methods shall not be re-used by other attributes and methods;

c) if these rules cannot be met, then a new interface class shall be created;

d) new versions of COSEM interface classes are administered by the DLMS UA.

6.3 Removal of interface classes

Besides one association object and the logical device name object no instantiation of an interface class is mandatory within a meter. Therefore, even unused interface classes will not be removed from the standard. They will be kept to ensure compatibility with possibly existing implementations.

Annex A
(normative)

Protocol related interface classes

A.1 General

Each communication device and/or communication profile needs some setup parameters to be defined for proper operation.

A.2 IEC local port setup (class_id: 19)

Instances of this interface class define the operational parameters for communication using IEC 62056-21. Several ports can be configured.

IEC local port setup		**0...n**	**class_id = 19, version = 1**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. default_mode	(static)	enum			
3. default_baud	(static)	enum			
4. prop_baud	(static)	enum			
5. response_time	(static)	enum			
6. device_addr	(static)	octet-string			
7. pass_p1	(static)	octet-string			
8. pass_p2	(static)	octet-string			
9. pass_w5	(static)	octet-string			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the "IEC local port setup" object instance. See D.2.1.12.	
default_mode	Defines the protocol used by the meter on the port.	
	enum	(0) protocol according to IEC 62056-21 (modes A...E), (1) protocol according to IEC 62056-46. Using this enumeration value all other attributes of this class are not applicable, (2) protocol not specified. Using this enumeration value, attribute 4), prop_baud is used for setting the communication speed on the port. All other attributes are not applicable.
default_baud	Defines the baud rate for the opening sequence.	
	enum	(0) 300 baud, (1) 600 baud, (2) 1 200 baud, (3) 2 400 baud, (4) 4 800 baud, (5) 9 600 baud, (6) 19 200 baud, (7) 38 400 baud, (8) 57 600 baud, (9) 115 200 baud

prop_baud	Defines the baud rate to be proposed by the meter. enum (0) 300 baud, (1) 600 baud, (2) 1 200 baud, (3) 2 400 baud, (4) 4 800 baud, (5) 9 600 baud, (6) 19 200 baud, (7) 38 400 baud, (8) 57 600 baud, (9) 115 200 baud
response_time	Defines the minimum time between the reception of a request (end of request telegram) and the transmission of the response (begin of response telegram). enum (0) 20 ms, (1) 200 ms
device_addr	Device address according to IEC 62056-21. octet-string
pass_p1	Password 1 according to IEC 62056-21. octet-string
pass_p2	Password 2 according to IEC 62056-21. octet-string
pass_w5	Password W5 reserved for national applications. octet-string

A.3 Modem configuration (class_id: 27)

NOTE The name of this interface class has been changed from "PSTN modem configuration" to "Modem configuration", as it can be used not only for configuring PSTN modems, but modems of other networks, for example GSM. Otherwise, the definition remains unchanged.

An instance of the "Modem configuration" class stores data related to the initialization of modems, which are used for data transfer from/to a device. Several modems can be configured.

Modem configuration		**0...n**	**class_id = 27, version = 1**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. comm_speed	(static)	enum	0	9	5
3. initialization_string	(static)	array			
4. modem_profile	(static)	array			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the “Modem configuration” object instance. See D.2.1.2.
comm_speed	The communication speed between the device and the modem, not necessarily the communication speed on the WAN. enum: (0) 300 baud, (1) 600 baud, (2) 1 200 baud, (3) 2 400 baud, (4) 4 800 baud, (5) 9 600 baud, (6) 19 200 baud, (7) 38 400 baud, (8) 57 600 baud (9) 115 200 baud

initialization_ string	Contains all the necessary initialization commands to be sent to the modem in order to configure it properly. This may include the configuration of special modem features.
	
	initialization_string ::= array
	initialization_string_element ::= structure
	{
	request: octet-string,
	response: octet-string,
	delay_after_response: long-unsigned
	}
	If the array contains more than one initialization_string_element, the requests are sent in a sequence. The next request is sent after the expected response matching the previous request and waiting a delay_after_response time [ms], to allow the modem to execute the request.
	
	REMARK: It is assumed that the modem is pre-configured so that it accepts the initialization_string. If no initialization is needed, the initialization string is empty.

modem_profile	Defines the mapping from Hayes standard commands/responses to modem specific strings.
	
	modem_profile::= array
	{
	modem_profile_element: octet-string
	}
	The modem_profile array must contain the corresponding strings for the modem used in the following order :
	
	Element 0: OK,
	Element 1: CONNECT,
	Element 2: RING,
	Element 3 NO CARRIER,
	Element 4: ERROR,
	Element 5: CONNECT 1 200,
	Element 6 NO DIAL TONE,
	Element 7: BUSY,
	Element 8: NO ANSWER,
	Element 9: CONNECT 600,

Element 10: CONNECT 2 400,
Element 11: CONNECT 4 800,
Element 12 CONNECT 9 600,
Element 13: CONNECT 14 400,
Element 14: CONNECT 28 800,
Element 15: CONNECT 36 600,
Element 16: CONNECT 56 000

A.4 Auto answer (class_id: 28)

NOTE The name of this interface class has been changed from "PSTN auto answer " to "Auto answer", as it can be used not only for PSTN networks, but also for other networks, for example GSM. Otherwise, the definition remains unchanged.

An instance of the "Auto answer" class stores data related to the management of data transfer between a device and a modem, which is used to answer incoming calls. Several modems can be configured.

Auto answer		**0...n**	**class_id = 28, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. mode	(static)	enum			
3. listening_window	(static)	array			
4. status	(dyn.)	enum			
5. number_of_calls	(static)	unsigned			
6. number_of_rings	(static)	nr_rings_type			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the "Auto answer" object instance. See D.2.1.4.
mode	Defines the working mode of the line when the device is auto answer. mode ::= enum (0) line dedicated to the device, (1) shared line management with a limited number of calls allowed. Once the number of calls is reached, the window status becomes inactive until the next start date, whatever the result of the call, (2) shared line management with a limited number of successful calls allowed. Once the number of successful communications is reached, the window status becomes inactive until the next start date, (3) currently no modem connected, (200...255) manufacturer specific modes

listening_window	Contains the start and end instant when the window becomes active (for the start instant), and inactive (for the end instant). The start_date defines implicitly the period.
	
	Example: when the day of month is not specified (equal to 0xFF) this means that we have a daily share line management. Daily, monthly …window management can be defined.
	
	listening_window ::= array window_element
	
	window_element ::= structure
	{
	start _time: octet-string,
	end_time: octet-string
	}
	start_time and end_time are formatted as set in 4.4.1 for *date_time*

status	Here is defined the status of the window. status ::= enum (0) Inactive: the device will manage no new incoming call. This status is automatically reset to Active when the next listening window starts. (1) Active: the device can answer to the next incoming call. (2) Locked: This value can be set automatically by the device or by a specific client when this client has completed its reading session and wants to give the line back to the customer before the end of the window duration. This status is automatically reset to Active when the next listening window starts.
number_of_calls	This number is the reference used in modes 1 and 2. When set to 0, this means there is no limit.

number_of_rings	Defines the number of rings before the meter connects the modem. Two cases are distinguished: number of rings within the window defined by attribute “listening_window” and number of rings outside the “listening_window”.
	
	nr_rings_type := structure
	{
	nr_rings_in_window: unsigned,
	(0: no connect in window)
	nr_rings_out_of_window: unsigned
	(0: no connect out of window)
	}

A.5 Auto connect (class_id: 29)

NOTE The name of this interface class has been changed from “PSTN auto dial” to “Auto connect”, as its use has been generalized; it is not only suitable for the configuration of sending messages – generally protocol telegrams – via PSTN modems, but also sending messages of various types over various communication infrastructures.

An instance of the “Auto connect” class controls and stores data related to the management of the data transfer from the metering device to one or several destinations.

The messages to be sent, the conditions on which they shall be sent and the relation between the various modes, the calling windows and destinations are not defined here.

Depending on the mode, one or more instances of this interface class may be necessary to perform the function of sending out messages.

Auto connect		0...n	class_id = 29, version = 1		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. mode	(static)	enum			
3. repetitions	(static)	unsigned			
4. repetition_delay	(static)	long-unsigned			
5. calling_window	(static)	array			
6. destination_list	(static)	array			
Specific method(s)		***m/o***			

Attribute description

logical_name Identifies the "Auto connect" object instance. See D.2.1.3.

mode Defines the mode controlling the auto dial functionality concerning the timing, the message type to be sent and the infrastructure to be used.

mode ::= enum

(0)	no auto dialling,
(1)	auto dialling allowed anytime,
(2)	auto dialling allowed within the validity time of the calling window,"regular"
(3)	auto dialling allowed within the validity time of the calling window; "alarm" initiated auto dialling allowed anytime,
(4)	SMS sending via Public Land Mobile Network (PLMN),
(5)	SMS sending via PSTN,
(6)	email sending,
(200..255)	manufacturer specific modes

repetitions The maximum number of trials in case of unsuccessful dialling attempts.

repetition_delay The time delay, expressed in seconds until an unsuccessful dial attempt can be repeated.

repetition_delay 0 means delay is not specified

calling_window Contains the start and end date/time stamp when the window becomes active (for the start instant), or inactive (for the end instant). The start_date defines implicitly the period.

Example: when day of month is not specified (equal to 0 x FF) this means that we have a daily share line management. Daily, monthly …window management can be defined.

calling_window ::= array window_element
window_element ::= structure
{
start_time: octet-string,
end_time: octet-string
}
start_time and end_time are formatted as set in 4.4.1 for *date_time*

destination_list	Contains the list of destinations (e.g. phone numbers, email addresses or their combinations) where the message(s) has(have) to be sent under certain conditions.
	
	The conditions and their link to the elements of the array are not defined here.
	
	destination_list ::= array destination
	{
	destination: octet_string
	}

A.6 IEC HDLC setup class (class_id: 23)

An instance of the "HDLC setup class" contains all data necessary to set up a communication channel according to IEC 62056-46. Several communication channels can be configured.

IEC HDLC setup		**0...n**	**class_id = 23, version = 1**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. comm_speed	(static)	enum	0	9	5
3. window_size_transmit	(static)	unsigned	1	7	1
4. window_size_receive	(static)	unsigned	1	7	1
5. max_info_field_length _transmit	(static)	long-unsigned	32	2030	128
6. max_info_field_length _receive	(static)	long-unsigned	32	2030	128
7. inter_octet_time_out	(static)	long-unsigned	20	6000	25
8. inactivity_time_out	(static)	long-unsigned	0		120
9. device_address	(static)	long-unsigned	0x0010	0x3FFD	
Specific method(s)		***m/o***			

NOTE 1 The maximum value of the attributes max_info_field_length_transmit and max_info_field_length_receive has been increased from 128 to 2030 for efficiency reasons.

NOTE 2 In order to ensure a minimal performance, the primary station should offer at least a max_info_field_length_receive of 128 bytes.

NOTE 3 The maximum value of the inter-octet-time-out attribute has been increased from 1000 ms to 6000 ms in order to allow using communication media, where long delays may occur. The default value has been changed to 25 ms to align with 6.4.4.3.3 of IEC 62056-46.

Attribute description

logical_name	Identifies the "IEC HDLC setup" object instance. See D.2.1.14.
comm_speed	The communication speed supported by the corresponding port enum: (0) 300 baud, (1) 600 baud, (2) 1 200 baud, (3) 2 400 baud, (4) 4 800 baud, (5) 9 600 baud, (6) 19 200 baud, (7) 38 400 baud, (8) 57 600 baud, (9) 115 200 baud This communication speed can be overridden if the HDLC mode of a device is entered through a special mode of another protocol.

window_size_ transmit	The maximum number of frames that a device or system can transmit before it needs to receive an acknowledgement from a corresponding station. During logon, other values can be negotiated.
window_size_ receive	The maximum number of frames that a device or system can receive before it needs to transmit an acknowledgement to the corresponding station. During logon, other values can be negotiated.
max_info_length_ transmit	The maximum information field length that a device can transmit. During logon, a smaller value can be negotiated.
max_info_length_ receive	The maximum information field length that a device can receive. During logon, a smaller value can be negotiated.
inter_octet_time_out	Defines the time, expressed in milliseconds, over which, when any character is received from the primary station, the device will treat the already received data as a complete frame.
inactivity_time_out	Defines the time, expressed in seconds over which, when any frame is received from the primary station, the device will process a disconnection. When this value is set to 0, this means that the inactivity_time_out is not operational.
device_address	Contains the physical device address of a device. In the case of single byte addressing: 0x00 NO_STATION Address, 0x01...0x0F Reserved for future use, 0x10...0x7D Usable address space, 0x7E 'CALLING' device address, 0x7F Broadcast address In the case of double byte addressing: 0x0000 NO_STATION address, 0x0001..0x000F Reserved for future use, 0x0010..0x3FFD Usable address space, 0x3FFE 'CALLING' physical device address, 0x3FFF Broadcast address

A.7 IEC twisted pair (1) setup (class_id: 24)

An instance of the "IEC twisted pair (1)" setup class contains all data which are necessary to set up a communication channel according to IEC 62056-31. Several communication channels can be configured.

IEC twisted pair (1) setup		0...n	class_id = 24, version = 0		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. secondary_address	(static)	octet-string			
3. primary_address_list	(static)	primary_adress_list_type			
4. tabi_list	(static)	tabi_list_type			
5. fatal_error	(dynamic)	enum			
Specific method(s)		***m/o***			

Attribute description

logical_name

Identifies the “IEC twisted pair setup” object instance. See D.2.1.15.

secondary_address

Secondary_address memorizes the ADS of the secondary station (see IEC 62056-31) that corresponds to the real equipment.

octet-string (SIZE(6))

primary_address_list

Primary_address_list memorizes the list of ADP or primary station physical addresses for which each logical device of the real equipment (the secondary station) has been programmed (see IEC 62056-31).

```
primary_adress_list_type ::=  array
{
        primary_address_element
}
primary_address_element ::= octet-string (SIZE(1))
```

tabi_list

tabi_list represents the list of the TAB(i) for which the real equipment (the secondary station) has been programmed in case of forgotten station call (see IEC 62056-31).

```
tabi_list_type ::= array tabi_element

tabi_element ::= integer
```

fatal_error

FatalError represents the last occurrence of one of the fatal errors of the protocols described in IEC 62056-31.

The initial default value of this variable is "00"H. Then, each fatal error is spotted.

```
enum    (0)     No-error,
        (1)     t-EP-1F,
        (2)     t-EP-2F,
        (3)     t-EL-4F,
        (4)     t-EL-5F,
        (5)     eT-1F,
        (6)     eT-2F,
        (7)     e-EP-3F,
        (8)     e-EP-4F,
        (9)     e-EP-5F,
        (10)    e-EL-2F
```

A.8 TCP-UDP setup (class_id: 41)

An instance of the TCP-UDP setup class contains all data necessary to set up the TCP or UDP sub-layer of the COSEM TCP or UDP based transport layer of a TCP-UDP/IP based communication profile.

In TCP-UDP/IP based communication profiles, all application associations between a physical device hosting one or more COSEM client application processes and a physical device hosting one or more COSEM server application processes rely on a single TCP or UDP connection. The TCP or UDP entity is wrapped in the COSEM TCP-UDP based transport layer. Within a physical device, each application process – client application process or server logical device - is bound to a Wrapper Port (WPort). The binding is done with the help of the SAP Assignment object.

On the other hand, a COSEM TCP or UDP based transport layer may be capable to support more than one TCP or UDP connections, between a physical device and several peer physical devices hosting COSEM application processes.

NOTE When a COSEM physical device supports various data link layers (e.g. Ethernet and PPP), then an instance of the TCP-UDP setup object is necessary for each of them.

TCP-UDP setup		**0...n**	**class_id = 41, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def***
1. logical_name	(static)	octet-string			
2. TCP-UDP_port	(static)	long-unsigned			
3. IP_reference	(static)	octet-string			
4. MSS	(static)	long-unsigned	40	65,535	576
5. nb_of_sim_conn	(static)	unsigned	1		
6. inactivity_time_out		long-unsigned			180
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the TCP-UDP setup object instance. See D.2.1.16.
TCP-UDP_port	Holds the TCP-UDP port number on which the physical device is listening for the DLMS/COSEM application.
IP_reference	References an IP setup object by its logical name. The referenced object contains information about the IP Address settings of the IP layer supporting the TCP-UDP layer.
MSS	With the help of the Maximum Segment Size (MSS) option, a TCP can indicate the maximum receive segment size to its partner. Note, that: - this option must only be sent in the initial connection request (i.e. in segments with the SYN control bit sent); - if this option is not present, conventionally MSS is considered as its default value, 576; - MSS is not negotiable; its value is indicated by this attribute.
nb_of_sim_conn	The maximum number of simultaneous connections the COSEM TCP-UDP based transport layer is able to support.

inactivity_time_out	Defines the time, expressed in seconds over which, if no frame is received from the COSEM client, the inactive TCP connection shall be aborted. When this value is set to 0, this means that the inactivity_time_out is not operational. In other words, a TCP connection, once established, in normal conditions – no power failure, etc. - will never be aborted by the COSEM server. Note, that all actions related to the management of the inactivity time-out function – measuring the inactivity time, aborting the TCP connection if the time-out is over, etc. – are managed inside the TCP-UDP layer implementation.

A.9 IPv4 setup (class_id: 42)

An instance of the IPv4 setup class handles all information that is related to the IP Address settings associated to a given device and to a lower layer connection on which these settings are used.

There shall be an instance of this class in a device for each different network interface implemented. For example, if a device has two serial interfaces (and is using the TCP/IP profile on both of them), there shall be two instances of the IPv4 Setup class in that device: one for each of these interfaces.

IPv4 setup	**0...n**	**class_id = 42, version = 0**		
Attribute(s)	***Data type***	***Min.***	***Max.***	***Def***
1. logical_name (static)	octet-string			
2. DL_reference (static)	octet-string			
3. IP_address	double-long-unsigned			
4. multicast_IP_address	array			
5. IP_options	array			
6. subnet_mask	double-long-unsigned			
7. gateway_IP_address	double-long-unsigned			
8. use_DHCP_flag (static)	boolean			
9. primary_DNS_address	double-long-unsigned			
10. secondary_DNS_address	double-long-unsigned			

Specific method(s)	***m/o***
1. add_mc_IP_address (data)	o
2. delete_mc_IP_address (data)	o
3. get_nbof_mc_IP_addresses (data)	o

Attribute description

logical_name	Identifies the IPv4 setup object instance. See D.2.1.17.
DL_reference	References a Data link layer (Ethernet or PPP) setup object by its logical name. The referenced object contains information about the specific settings of the data link layer supporting the IP layer.

IP_address	Carries the value of the IP address (IPv4 address) of this physical device on the network to which the device is connected via the referenced lower layer interface. It can be either (static) or (dynamic). In the latter case, dynamic IP address assignment (e.g. DHCP) is used. If no IP address is assigned, the value is 0.
multicast_IP_ address	Contains an array of IP addresses. IP addresses in this array must fall into the multicast group address range ("Class D" addresses, including IP addresses in the range of 224.0.0.0 to 239.255.255.255). When a device receives an IP datagram with one of these IP addresses in the destination IP address field, it shall consider that this datagram is addressed to it. multicast_IP_address ::= array double-long-unsigned

IP_options	Contains the necessary parameters to support the selected IP options, for example Datagram time-stamping or security services (IPSec).
	IP_options ::= array IP_options_element
	
	IP_options_element ::= SEQUENCE
	{
	IP-Option-Type unsigned,
	IP-Option-Length unsigned,
	IP-Option-Data octet-string
	}
	
	Allowed IP-Option-Types:
	
	- Security - 0x82
	
	If this option is present, the device shall be allowed to send security, compartmentation, handling restrictions and TCC (closed user group) parameters within its IP Datagrams. The value of the IP-Option-Length Field must be 11, and the IP-Option-Data shall contain the value of the Security, Compartments, Handling Restrictions and Transmission Control Code values, as specified in STD 0005/RFC 791.
	
	- Loose Source and Record Route - 0x83
	
	If this option is present, the device shall supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.
	
	The IP-Option-length and IP-Option-Data values are specified in STD 0005/RFC 791.
	
	- Strict Source and Record Route - 0x89
	
	If this option is present, the device shall supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.
	
	The IP-Option-length and IP-Option-Data values are specified in STD 0005/RFC 791.

	- Record Route - 0x07 If this option is present, the device shall as well: - send originated IP Datagrams with that option, providing means to record the route of these Datagrams; - as a router, send routed IP Datagrams with the route option adjusted according to this option. The IP-Option-length and IP-Option-Data values are specified in STD 0005/RFC 791. - Internet Timestamp - 0x44 If this option is present, the device shall as well: - send originated IP Datagrams with that option, providing means to time-stamp the datagram in the route to its destination; - as a router, send routed IP Datagrams with the time-stamp option adjusted according to this option. The IP-Option-length and IP-Option-Data values are specified in STD 0005/RFC 791.
subnet_mask	Contains the subnet mask. When sub-networking is used in a network segment, each device concerned must behave conforming to the sub-networking rules. In order to do that, the device, besides of its IP address, needs also to know, how the IP address is structured within this sub-networked segment. The subnet_mask attribute carries this information. With IPv4, the subnet_mask is a 32 bits word, expressed exactly in the same format as an IP Address (e.g. 255.255.255.0), but has another meaning: the '0' bits of the subnet_mask indicate the portion of the IP Address which is still used as Device_ID on a sub-networked IP Network [4].
gateway_IP_address	Contains the IP Address of the gateway device. In most IP implementations, there is a code in the module that handles outgoing datagrams to decide if a datagram can be sent directly to the destination on the local network or if it must be sent to a gateway. In order to be able to send non-local datagrams to the gateway, the device must know the IP address of the gateway device assigned to the given network segment. If no IP address is assigned, the value is 0.
use_DHCP_flag	When this flag is set to TRUE, the device uses DHCP (Dynamic Host Configuration Protocol) to dynamically determine the IP_address, subnet_mask and gateway_IP_address parameters. On the other hand, when this flag is set to FALSE, the IP_address, subnet_mask and gateway_IP_address parameters must be locally set.

[4] See more about sub-networking in RFC 940 and RFC 950.

primary_DNS_ address	The IP Address of the primary Domain Name Server (DNS). If no IP address is assigned, the value is 0.
secondary_DNS_address	The IP Address of the secondary Domain Name Server (DNS). If no IP address is assigned, the value is 0.

Method description

add_mc_IP_address (IP_Address)	Adds one multicast IP address to the multicast_IP_Address array. IP_Address ::= double-long-unsigned
delete_mc_IP_ address (IP_Address)	Deletes one IP Address from the multicast_IP_Address array. The IP Address to be deleted is identified by its value. IP_Address ::= double-long-unsigned
get_nbof_mc_IP_ addresses (data)	Returns the number of IP Addresses contained in the multicast_IP_Address array. data ::= unsigned

A.10 Ethernet setup (class_id: 43)

An instance of the Ethernet setup class handles all information that is related to Ethernet settings associated to a given physical device and to a lower layer connection on which these settings are used.

There shall be an instance of this class for each network interface of a physical device, using the Ethernet protocol.

Ethernet setup		**0...n**	**class_id = 43, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. MAC_address		octet-string			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the Ethernet setup object instance. See D.2.1.18.
MAC_address	Holds the MAC address.

A.11 PPP setup (class_id: 44)

An instance of the PPP setup class handles all information that is related to PPP settings associated to a given physical device and to a lower layer connection on which these settings are used.

There shall be an instance of this class for each network interface of a physical device, using the PPP protocol.

PPP setup		**0...n**	**Class_id = 44, version = 0**		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. PHY_reference	(static)	octet-string			
3. LCP_options	(static)	LCP_options_type			
4. IPCP_options	(static)	IPCP_options_type			
5. PPP_authentication	(static)	PPP_auth_type			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the PPP setup object instance. See D.2.1.19.
PHY_reference	References another object by its logical_name. The object referenced contains information about the specific physical layer interface, supporting the PPP layer.
LCP_options	This attribute contains the parameters for the Link Control Protocol options. These options include: - Maximum-Receive-Unit (MRU, Type 1, STD 0051/RFC 1661). This configuration option may be sent to inform the peer that the implementation can receive larger packets, or to request that peer send smaller packets. The default value is 1500 octets; - Async-Control-Character-Map (ACCM, Type 2, STD 0051/RFC 1662): This configuration option provides a method to negotiate the use of control character transparency on asynchronous links; - Authentication-Protocol (Type 3, STD 0051/RFC 1661, PAP, CHAP or EAP); This configuration option provides a method to negotiate the use of a specific protocol for authentication. By default, authentication is not required; - Magic-Number (Type 5, STD 0051/RFC 1661). This configuration option provides a method to detect looped-back links and other data link layer anomalies; - Protocol-Field-Compression (PFC, Type 7, STD 0051/RFC 1661). This configuration option provides a method to negotiate the compression of the PPP protocol field; - Address-and-Control-Field-Compression (ACFC, Type 8, STD 0051/RFC 1661). This configuration option provides a method to negotiate the compression of the data link layer address and control fields; - FCS-Alternatives (Type 9, RFC 1570). This configuration option provides a method for an implementation to specify another FCSS format to be sent by the peer, or to negotiate away the FCS altogether; - Call-back (Type 13, RFC 1570). This configuration option provides a method for an implementation to request a dial-up peer to call back. This provides enhanced security by ensuring that the remote site can connect only from a single location as defined by the call-back number. The structure of this attribute is the following: LCP_options ::= SEQUENCE OF LCP_options_element LCP_options_element ::= SEQUENCE { LCP-option-type unsigned, LCP-option-length unsigned,

```
    LCP-option-data     CHOICE

                        {
                        MRU             [1] long-unsigned,
                        ACCM            [2] double-long-unsigned,
                        Auth-Prot       [3] long-unsigned,
                        Mag-Num         [5] double-long-unsigned,
                        ProtF-Compr     [7] boolean,
                        AdCtr-Compr     [8] boolean,
                        FCS-Alter       [9] unsigned,
                        Callback        [13] callback-data
                        }
}

LCP_option_type  ::= ENUMERATED
{
        Max-Rec-Unit                    (1),
        Async-Control-Char-Map          (2),
        Auth-Protocol                   (3),
        Magic-Number                    (5),
        Protocol-Field-Compression      (7),
        Address-and-Ctr-Compression     (8),
        FCS-Alternatives                (9),
        Callback                        (13)
}
```

For the LCP-option-data element, the following applies:

The default value of MRU is 1500.

The value of the Auth-Prot (Authentication Protocol) element indicates the authentication protocol used on the given PPP link.

Possible values today are:

0x0000 - No authentication protocol is used,
0xc023 - The PAP protocol is used,
0xc223 - The CHAP protocol is used,
0xc227 - The EAP protocol is used.

The value of the FCS-Alter (FCS Alternatives) options field identifies the FCS used. These are assigned as follows:

```
        bit 1   Null FCS,
        bit 2   CCITT 16-bit FCS,
        bit 3   CCITT 32-bit FCS
```

```
Callback-data ::= SEQUENCE
{
        callback-active         boolean,        // default: false,
        callback-data-length    unsigned,
        callback-operation      unsigned,
        callback-message        octet-string
}
```

The callback-active member indicates whether the callback option is active on this PPP link.

```
callback-operation  ::= ENUMERATED
{
        Location-is-determined-by-user-authentication        (0),
        Dialling-string                                      (1),
        Location-identifier                                  (2),
        E.164-number                                         (3),
        X500-distinguished-name                              (4),
        Location-is-determined-during-CBCP-negotiation       (6)
}
```

The callback-message field is zero or more octets, and its general contents are determined by the callback-operation field. The actual format of the information is site or application specific (see in RFC 1570).

NOTE 1 For more details on Link Control Protocol, please refer to RFC 1661.
NOTE 2 For latest assigned numbers, see RFC 1700.

IPCP_options

This attribute contains the parameters for the IP Control Protocol – the Network Control Protocol module of the PPP for negotiating IP parameters on the PPP link options. These options include:

- IP-Compression-Protocol (Type 2, RFC 1332). This parameter indicates the IP compression protocol supported within the PPP link described by this object;
- Preferred-Local-IP-Address (Type 3, RFC 1332). This configuration option provides a way to negotiate the IP address to be used on the local end of the link. It allows the sender of the Configure-Request to state, which IP-address is desired, or to request that the peer provide the information. The peer can provide this information by NAK-ing the option, and returning a valid IP-Address;
- Preferred-Peer-IP-Addresses. This configuration option provides a way to negotiate the IP Address to be used on the remote end of the link. When the Grant-Access-Only-to-Pref-Peer-on-List parameter is set to be TRUE, the device shall accept PPP connection only with a remote device having one of the IP Addresses on this list. When the Use-Static-IP-Pool parameter is set to TRUE, the COSEM Server device shall try to assign one of these IP Addresses to the remote device;
- Grant-Access-Only-to-Pref-Peer-on-List. (GAO) This parameter indicates whether the device can accept PPP connection only with peer devices with IP Address on the above list or not. Its default value is FALSE;
- Use-Static-IP-Pool (USIP). This parameter indicates whether the device should try to assign one of the IP Addresses of the Preferred-Peer-IP-Addresses to the remote end device during the IP Address negotiation phase or not.

The structure of this attribute is as follows:

```
IPCP_options  ::= SEQUENCE OF IPCP_options_element

IPCP_options_element        ::= SEQUENCE
{
        IPCP-option-type            unsigned,
        IPCP-option-length          unsigned,
        IPCP-option-data            CHOICE
        {
        IP-Comp-Prot  [2] long-unsigned,
        Pref-Local-IP [3] double-long-unsigned,
        Pref-Peer-IP  [20] SEQUENCE OF double-long-unsigned,
        GAO           [21] boolean,
        USIP          [22] boolean
        }
```

```
}
IPCP-option-type  ::= ENUMERATED
{
        IP-Comp-Prot  (2),
        Pref_Local-IP  (3),
        Pref-Peer-IP   (20),
        GAO            (21),
        USIP           (22)
}
```

Possible values for the IP-Compression-Protocol (IP-Comp-Prot) parameter today are:

0x0000 – No IP Compression is used (default),
0x002d – Van Jacobson (RFC 1232),
0x0061 – IP Header Compression (RFC 2507, 3544)
0x0003 – Robust Header Compression (RFC 3241)

NOTE 1 For more details on IPCP, please refer to RFC 1332.
NOTE 2 For latest assigned numbers, see RFC 1700.

PPP_ authentication

Contains the parameters required by the PPP authentication procedure used.

```
PPP_authentication  ::=  CHOICE {
      No-authentication:    [0] NULL,
      PAP-login             [1] structure,
                            {
                              user-name              octet-string,
                              PAP-password           octet-string
                            }
      CHAP-algorithm        [2] structure,
                            {
                              user-name              octet-string
                              algorithm_id           unsigned
                              --default: 5 (MD5)
                            }
      EAP-params:           [3] supported-EAP-types
}
```

Possible values for CHAP-algorithm-id parameter today are as follows:
0x05 – CHAP with MD5 (default),
0x06 – SHA-1,
0x80 – MS-CHAP,
0x81 – MS-CHAP-2

New values can be used as become assigned.

NOTE When CHAP is used, a “secret” is also required to verify the “challenge”[5] sent by the Client. This “secret” is not accessible in the PPP setup object.

```
supported-EAP-types ::= SEQENCE
{
        md5-challenge          boolean,
        one-time-password      boolean,
        generic-token-card     boolean
}
```

[5] For more details about CHAP, please refer to RFC 1994.

A.12 GPRS modem setup (class_id: 45)

A "GPRS modem setup" object stores all the necessary data for a GPRS modem management.

GPRS modem setup		0...n	class_id = 45, version = 0		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. APN	(static)	octet-string			
3. PIN_code	(static)	long-unsigned			
4. quality_of_service	(static)	structure			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the "GPRS modem setup" object instance. See D.2.1.20.
APN	Defines the access point name of the network. octet-string
PIN_code	Holds the personal identification number. long-unsigned

quality_of_service	Specifies the quality of service parameters. It is a structure of 2 elements:
	- the first one defines the default or minimum characteristics of the concerned network. These parameters have to be set to best effort value;
	- the second element defines the requested parameters.
	quality_of_service ::= structure
	{
	default qos_element,
	requested qos_element
	}
	
	qos_element ::= structure
	{
	precedence unsigned,
	delay unsigned,
	reliability unsigned,
	peak throughput unsigned,
	mean throughput unsigned
	}

A.13 SMTP setup (Class_id: 46)

An SMTP setup object allows defining the parameters for SMTP setup.

SMTP setup		0...n	class_id = 46, version = 0		
Attribute(s)		***Data type***	***Min.***	***Max.***	***Def.***
1. logical_name	(static)	octet-string			
2. server_port	(static)	long-unsigned			25
3. user_name	(static)	octet-string			
4. login_password	(static)	octet-string			
5. server_address	(static)	octet-string			
6. sender_address	(static)	octet-string			
Specific method(s)		***m/o***			

Attribute description

logical_name	Identifies the “SMTP setup” object instance. See D.2.1.21.
server_port	Defines the value of the TCP-UDP port related to this protocol. By default, this value is the SMTP port Id given by IETF: 25.
user_name	Defines the user name to be used for the login to the SMTP server.
login_password	Password to be used for login. When the string is void, this means that there is no authentication.
server_address	Defines the server address as an octet string. This server address can be a name, which must be resolvable by the primary DNS or the secondary DNS. In the case when it is directly the IP address of the server, which is specified here, it shall be a string in dotted format. Example: 163.187.45.87
sender_address	Defines the sender address as an octet string. This server address can be a name. In the case when it is directly the IP address of the server, which is specified here, it will be a string in dotted format.

Annex B
(normative)

Data model and protocol

The data model uses generic building blocks to define the complex functionality of the metering equipment. It provides a view of this functionality of the meter, as it is available at its interface(s). The model does not cover internal, implementation specific issues.

The communication protocol defines how the data can be accessed and exchanged.

This is illustrated in the figure below:

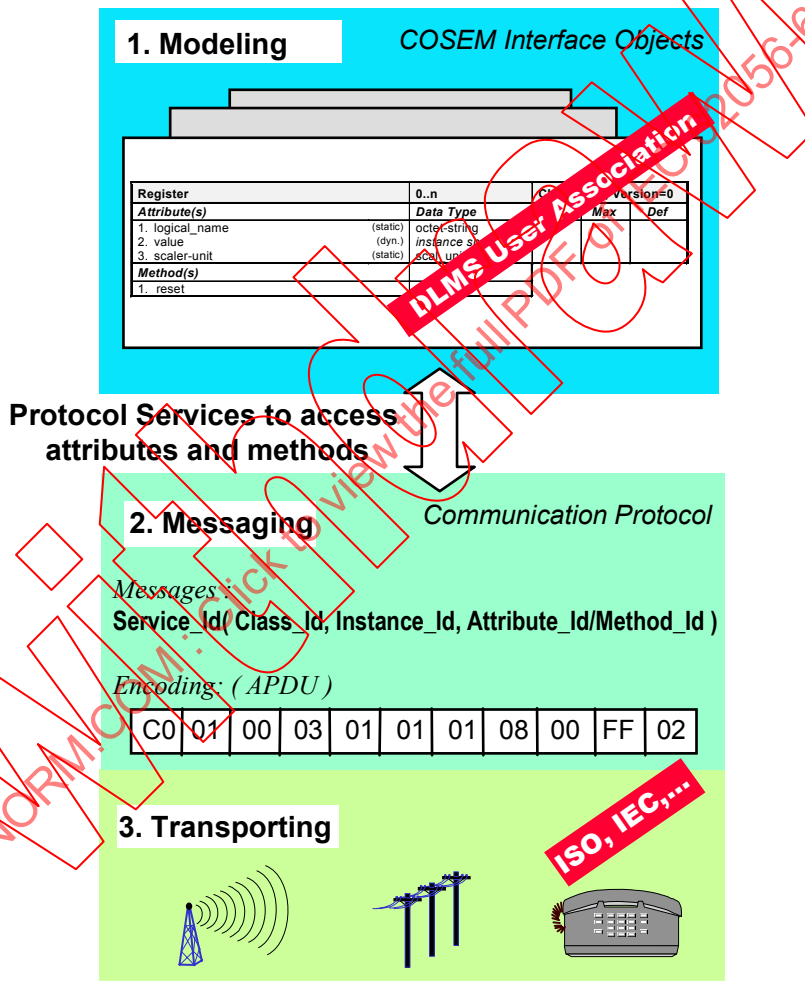


Figure B.1 – The three step approach of COSEM

- The COSEM specification specifies metering domain specific interface classes. The functionality of the meter is defined by the instances of these interface classes, called COSEM objects. This is defined in this standard. Logical names, (OBIS codes), identifying the COSEM objects are defined in IEC 62056-61.
- The attributes and methods of these COSEM objects can be accessed and used via the messaging services of the application layer.
- The lower layers of the protocol transport the information.

Annex C
(normative)

Using short names for accessing attributes and methods

C.1 Introduction – referencing methods

Attributes and methods of COSEM objects can be referenced in two different ways:

Using COSEM logical names: In this case, the attributes and methods of a COSEM object are referenced via the identifier of the COSEM object instance to which they 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

where

- *attribute_index* is used as the identifier of the required attribute;
- *method_index* is used as the identifier of the required method.

Using 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 for the short name is the same as the syntax of the name of a DLMS named variable.

C.2 Guidelines for assigning short names

This clause gives guidelines for assigning short names for public attributes and methods.

Data **class_id = 1, version = 0**	Short name	Remarks
***Attribute*(s)**		
logical_name	*x*	*x* is the base_name of the object.
value	*x*+8	
Specific method(s)		

Register **class_id = 3, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
value	*x*+8	
scaler_unit	*x*+16	
Specific method(s)		
reset (data)	*x*+40	

Extended register class_id = 4, version = 0	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
value	*x*+8	
scaler_unit	*x*+16	
status	*x*+24	
capture_time	*x*+32	
Specific method(s)		
reset (data)	*x*+56	

Demand register class_id = 5, version = 0	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
current_average_value	*x*+8	
last-average_value	*x*+16	
scaler_unit	*x*+24	
status	*x*+32	
capture_time	*x*+40	
start_time_current	*x*+48	
period	*x*+56	
number_of_periods	*x*+64	
Specific method(s)		
reset (data)	*x*+72	
next_period (data)	*x*+80	

Register activation class_id = 6, version = 0	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
register_assignment	*x*+8	
mask_list	*x*+16	
active_mask	*x*+24	
Specific method(s)		
add_register (data)	*x*+48	
add_mask (data)	*x*+56	
delete_mask (data)	*x*+64	

Profile generic **class_id = 7, version = 1**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Buffer	*x*+8	Selective access to the buffer is provided using parameterized access.
capture_objects	*x*+16	
capture_period	*x*+24	
sort_method	*x*+32	
sort_object	*x*+40	
entries_in_use	*x*+48	
profile_entries	*x*+56	
Specific method(s)		
reset (data)	*x*+88	
capture (data)	*x*+96	

Clock **class_id = 8, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Time	*x*+8	
time_zone	*x*+16	
Status	*x*+24	
daylight_savings_begin	*x*+32	
daylight_savings_end	*x*+40	
daylight_savings_deviation	*x*+48	
daylight_savings_enabled	*x*+56	
clock_base	*x*+64	
Specific method(s)		
adjust_to quarter (data)	*x*+96	
adjust_to_measuring_period (data)	*x*+104	
adjust_to_minute (data)	*x*+112	
adjust_to_preset_time (data)	*x*+120	
preset_adjusting_time (data)	*x*+128	
shift_time (data)	*x*+136	

Script table **class_id = 9, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Scripts	*x*+8	
Specific method(s)		
execute (data)	*x*+32	

Schedule **class_id = 10, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Entries	*x*+8	
Specific method(s)		
enable/disable (data)	*x*+32	
insert (data)	*x*+40	
delete (data)	*x*+48	

Special days table **class_id = 11, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Entries	*x*+8	
Specific method(s)		
insert (data)	*x*+16	
delete (data)	*x*+24	

Activity calendar **class_id = 20, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
calendar_name_active	*x*+8	
season_profile_active	*x*+16	
week_profile_table_active	*x*+24	
day_profile_table_active	*x*+32	
calendar_name_passive	*x*+40	
season_profile_passive	*x*+48	
week_profile_table_passive	*x*+56	
day_profile_table_passive	*x*+64	
activate_passive_calendar_time	*x*+72	
Specific method(s)		
activate_passive_calendar (data)	*x*+80	

Association SN **class_id = 12, version = 1**	Short name	Remarks
Attribute(s)		
logical_name	*x* [a]	*x* is the base name of the object.
object_list	*x*+8	Selective access to the object_list is provided using parameterized access.
Specific method(s)		
read_by_logicalname (data)	*x*+48	With this method, the parameterized access feature can also be used.
get_attributes&methods (data)	*x*+56	With this method, the parameterized access feature can also be used.
change_LLS_secret (data)	*x*+64	
change_HLS_secret (data)	*x*+72	
reply_to_HLS_authentication (data)	*x*+88	

[a] The base name of the object "Association SN" corresponding to the current association is 0xFA00.

SAP assignment **class_id = 17, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
SAP_assignment_list	*x*+8	
Specific method(s)		
connect_logical_device (data)	*x*+32	

Register monitor **class_id = 21, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Thresholds	*x*+8	
monitored_value	*x*+16	
Actions	*x*+24	
Specific method(s)		

Utility tables **class_id = 26, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
table_ID	*x*+8	
Length	*x*+16	
Buffer	*x*+24	
Specific method(s)		

Single action schedule **class_id = 22, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
executed_script	*x*+8	
Type	*x*+16	
execution_time	*x*+24	
Specific method(s)		

Register table **class_id = 61, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
table_cell_values	*x*+8	
table_cell_definition	*x*+16	
scaler_unit	*x*+24	
Specific method(s)		
Reset	x+40	
Capture	*x*+48	

Status mapping **class_id = 63, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
status_word	*x*+8	
mapping_table	*x*+16	
Specific method(s)		

IEC local port setup **class_id = 19, version = 0 or 1**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
default_mode	*x*+8	
default_baud	*x*+16	
prop_baud	*x*+24	
response_time	*x*+32	
device_addr	*x*+40	
pass_p1	*x*+48	
pass_p2	*x*+56	
pass_w5	*x*+64	
Specific method(s)		

Modem configuration **class_id = 27, version = 0 or 1**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
□niti_speed	*x*+8	
□nitialization_string	*x*+16	
modem_profile	*x*+24	
Specific method(s)		

Auto answer **class_id = 28, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Mode	*x*+8	
listening_window	*x*+16	
Status	*x*+24	
number_of_calls	*x*+32	
number_of_rings	*x*+40	
Specific method(s)		

PSTN auto dial **class_id = 29, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Mode	*x*+8	
Repetitions	*x*+16	
repetition_delay	*x*+24	
calling_window	*x*+32	
phone_list	*x*+40	
Specific method(s)		

Auto connect **class_id = 29, version = 1**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
Mode	*x*+8	
Repetitions	*x*+16	
repetition_delay	*x*+24	
calling_window	*x*+32	
destination_list	*x*+40	
Specific method(s)		

IEC HDLC setup **class_id = 23, version = 0 or 1**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
comm_speed	*x*+8	
window_size_transmit	*x*+16	
window_size_receive	*x*+24	
max_info_length_transmit	*x*+32	
max_info_length_receive	*x*+40	
inter_octet_time_out	*x*+48	
inactivity_time_out	*x*+56	
device_address	*x*+64	
Specific method(s)		

IEC twisted pair (1) setup **class_id = 24, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
secondary_address	*x*+8	
primary_address_list	*x*+16	
tabi_list	*x*+24	
fatal_error	*x*+32	
Specific method(s)		

TCP-UDP setup class_id = 41, version = 0	Short name	Remarks
Attribute(s)		
logical_name	x	*x* is the base_name of the object.
TCP-UDP_port	*x*+8	
IP_reference	*x*+16	
MSS	*x*+24	
nb_of_sim_conn	*x*+32	
inactivity_time_out	*x*+40	
Specific method(s)		

IPv4 setup class_id = 42, version = 0	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
DL_reference	*x*+8	
IP_address	*x*+16	
multicast_IP_address	*x*+24	
IP_options	*x*+32	
subnet_mask	*x*+40	
gateway_IP_address	*x*+48	
use_DHCP_flag	*x*+56	
primary_DNS_address	*x*+64	
secondary_DNS_address	*x*+72	
Specific method(s)		
add_mc_IP_address	*x*+96	
delete_mc_IP_address	*x*+104	
get_nbof_mc_IP_addresses	*x*+112	

Ethernet setup class_id = 43, version = 0	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
MAC_address	*x*+8	
Specific method(s)		

PPP setup class_id = 44, version = 0	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
PHY_reference	*x*+8	
LCP_options	*x*+16	
IPCP_options	*x*+24	
PPP authentication	*x*+32	
Specific method(s)		

GPRS modem setup **class_id = 45, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
APN	x+8	
PIN_code	x+16	
quality_of_service	x+24	
Specific method(s)		

SMTP setup **class_id = 46, version = 0**	Short name	Remarks
Attribute(s)		
logical_name	*x*	*x* is the base_name of the object.
server_port	*x*+8	
user_name	*x*+16	
login_password	*x*+24	
server_address	*x*+32	
sender_address	*x*+40	
Specific method(s)		

C.3 Reserved base_names for special COSEM objects

In order to grant access for devices offering accessing by short_names some short_names are reserved as base_names for special COSEM objects. The reserved range of names is from 0x**FA00** to 0x**FFF8**.

The following specific base_names are defined:

Base_name (objectName)	**COSEM object**
0x **FA00**	Association SN
0x **FB00**	Script table (instantiation: “broadcast_receiver script”)
0x **FC00**	SAP assignment
0x **FD00**	“Data” or “Register” object containing the “COSEM logical device name” in the attribute "value"

Annex D
(normative)

Relation to OBIS

D.1 General

The OBIS identification system serves as a basis for the COSEM logical names. The system of naming COSEM objects is defined in the basic principles (see Clause 4), the identification of real data items is specified in IEC 62056-61.

The following clauses define the usage of those definitions in the COSEM environment.

All codes, which are not explicitly listed, but outside the manufacturer specific range are reserved for future use.

D.2 Mapping of data items to COSEM objects and attributes

This clause defines the usage of OBIS identifications and their mapping to COSEM objects of certain interface classes and their attributes.

D.2.1 Abstract COSEM objects

This subclause contains definitions for data items not directly linked to an energy type.

Value group C Abstract objects (A = 0)	
0	General purpose COSEM objects
1	COSEM objects of IC "Clock"
2	COSEM objects IC “Modem configuration” and related ICs
10	COSEM objects of IC "Script table"
11	COSEM objects of IC "Special days table"
12	COSEM objects of IC "Schedule"
13	COSEM objects of IC "Activity calendar"
14	COSEM objects of IC “Register activation”
15	COSEM objects of IC "Single action schedule"
16	COSEM objects of IC “Register monitor”
20	COSEM objects of IC "IEC local port setup"
21	Standard readout definitions
22	COSEM objects of IC “IEC HDLC setup”
23	COSEM objects of IC "IEC twisted pair (1) setup"
25	COSEM objects of IC “TCP-UDP setup”, “IPv4 setup”, “Ethernet setup”, “PPP setup”, “GPRS modem setup”, “SMTP setup”.
40	COSEM objects of IC “Association SN/LN”
41	COSEM objects of IC “SAP assignment”

Value group C Abstract objects (A = 0)	
42	COSEM logical device name
65	COSEM objects of IC “Utility tables”
128 ...199	Manufacturer specific COSEM related abstract objects
All other	Reserved

D.2.1.1 Clock (class_id:8)

This COSEM object controls the system clock of the physical device. It is an instance of the interface class "Clock".

Clock	IC	OBIS identification					
		A	B	C	D	E	F
Clock object	Clock	0	x	1	0	x	255

The usage of value group E shall be:

- if just one object is instantiated, value E shall be 0;
- if more than one object is instantiated in the same physical device, the value group E shall number the instantiations from zero to the needed maximum value.

D.2.1.2 Modem configuration (class_id: 27)

This COSEM object defines and controls the behaviour of the device regarding the communication through a modem. It is an instance of the interface class “Modem configuration".

Modem configuration	IC	OBIS identification					
		A	B	C	D	E	F
Modem configuration object	Modem configuration	0	x	2	0	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channel.

D.2.1.3 Auto connect (class_id: 29)

This COSEM object defines the necessary parameters for the management of sending information from the metering device to one or more destinations. It is an instance of the interface class “Auto connect".

Auto connect	IC	OBIS identification					
		A	B	C	D	E	F
Auto connect object	Auto connect	0	x	2	1	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, its value group B shall number the communication channel.

D.2.1.4 Auto answer (class_id: 28)

This COSEM object defines and controls the behaviour of the device regarding the auto answering function using a modem. It is an instance of the interface class “Auto answer".

Auto answer	IC	OBIS identification					
		A	B	C	D	E	F
Auto answer object	Auto answer	0	*x*	2	2	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, its value group B shall number the communication channel.

D.2.1.5 Script tables (class_id: 9)

These COSEM objects control the behaviour of the device.

Several instances of the interface class "Script table" are predefined and normally available as hidden scripts only with access to the execute() method.

The following table contains only the identifiers for the “standard” instances of the listed scripts. Implementation specific instances of these scripts should use values different from zero in value group D.

Script table objects	IC	OBIS identification					
		A	B	C	D	E	F
Global meter reset[a]	Script table	0	*x*	10	0	0	255
MDI reset / end of billing period[a]		0	*x*	10	0	1	255
Tariffication script table		0	*x*	10	0	100	255
Activate test mode[a]		0	*x*	10	0	101	255
Activate normal mode[a]		0	*x*	10	0	102	255
Set output signals		0	*x*	10	0	103	255
Switch optical test output[b, c]		0	*x*	10	0	104	255
Power quality measurement management		0	*x*	10	0	105	255
Broadcast script table		0	*x*	10	0	125	255

[a] The activation of these scripts is performed by calling the execute() method to the script identifier 1 of the corresponding script object.

[b] The optical test output is switched to measuring quantity Y and the test mode is activated by calling the execute method of the script table object 0.x.10.0.104.255 using Y as parameter; where Y is given by Table 5 of IEC 62056-61 (OBIS). The default value of A is 1 (Electricity).

Example: In case of electricity meters, A = 1, default, execute (21) switches the test output to display the active power + of phase 1.

[c] The optical test output is also switched back to its default value when this script is activated.

The tariffication script table defines the entry point into tariffication by standardizing utility-wide how to invoke the activation of certain tariff conditions.

The broadcast script table allows standardising utility wide the entry point into regularly needed functionality.

D.2.1.6 Special days table (class_id: 11)

This COSEM object defines and controls the behaviour of the device regarding calendar functions on special days for clock control. It is an instance of the interface class "Special days table".

Special days table	IC	OBIS identification					
		A	B	C	D	E	F
Special days table object	Special days table	0	*x*	11	0	0	255

D.2.1.7 Schedule (class_id: 10)

This COSEM object defines and controls the behaviour of the device in a sequenced way. It is an instance of the interface class "Schedule".

Schedule	IC	OBIS identification					
		A	B	C	D	E	F
Schedule object	Schedule	0	*x*	12	0	x	255

The usage of value group E shall be:

- if just one object is instantiated, value E shall be 0;
- if more than one object is instantiated in the same physical device, the value group E shall number the instantiations from zero to the needed maximum value.

D.2.1.8 Activity calendar (class_id: 20)

This COSEM object defines and controls the behaviour of the device in a calendar-based way. It is an instance of the interface class "Activity calendar".

Activity calendar	IC	OBIS identification					
		A	B	C	D	E	F
Activity calendar object	Activity calendar	0	*x*	13	0	0	255

D.2.1.9 Register activation (class_id: 6)

This COSEM object is used to handle different tariffication structures. It is an instance of the interface class "Register activation".

Register activation	IC	OBIS identification					
		A	B	C	D	E	F
Register activation object	Register activation	0	*x*	14	0	x	255

The usage of value group E shall be:

- if just one object is instantiated, value E shall be 0;
- if more than one object is instantiated in the same physical device, the value group E shall number the instantiations from zero to the needed maximum value.

D.2.1.10 Single action schedule (class_id: 22)

These COSEM objects control the behaviour of the device. One instance of the interface class "Single action schedule" is predefined. Implementation specific instances of these interface classes should use values different from zero in value group D.

Single action schedule	IC	OBIS identification					
		A	B	C	D	E	F
End of billing period	Single action schedule	0	*x*	15	0	0	255

D.2.1.11 Register monitor (class_id: 21)

These COSEM objects control the register monitoring function of the device. They define the value to be monitored, the set of thresholds to which the value is compared, and the actions to be performed when a threshold is crossed.

In general, the following logical name(s) shall be used:

Register monitor	IC	OBIS identification					
		A	B	C	D	E	F
Register monitor object	Register monitor	0	*x*	16	0	x	255

The use of value group E shall be:

- if just one “Register monitor” object is instantiated the value of E shall be 0;
- if more than one object is instantiated in the same logical device, the value group E shall number the instantiations from zero to the needed maximum value.

See also D.2.2.19 and D.2.2.20.

D.2.1.12 IEC local port setup (class_id: 19)

These COSEM objects define and control the behaviour of the device regarding the communication parameters on the local port according to IEC 62056-21. They are instances of the interface class "IEC local port setup".

IEC local port setup	IC	OBIS identification					
		A	B	C	D	E	F
IEC optical port setup object	IEC local port setup	0	x	20	0	0	255
IEC electrical port setup object		0	x	20	0	1	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channel.

D.2.1.13 Standard readout profile (class_id: 7)

A set of COSEM objects is defined to carry the standard readout as it would appear with IEC 62056-21 (modes A to D).

Standard readout	IC	OBIS identification					
		A	B	C	D	E	F
General local port readout	Profile generic	0	0	21	0	0	255
General display readout		0	0	21	0	1	255
Alternate display readout		0	0	21	0	2	255
Service display readout		0	0	21	0	3	255
List of configurable meter data		0	0	21	0	4	255
Additional readout profile 1		0	0	21	0	5	255
.....							
Additional readout profile *n*		0	0	21	0	N	255

For the parametrization of the standard readout “Data” objects can be used:

Standard readout parametrization	IC	OBIS identification					
		A	B	C	D	E	F
Standard readout parametrization object	Data	0	0	21	0	x	255

The usage of value group E shall be:

- if just one object is instantiated, value E shall be 0;
- if more than one object is instantiated in the same physical device, the value group E shall number the instantiations from zero to the needed maximum value.

Standard readout objects can also be related to an energy type and to a channel. See IEC 62056-61.

D.2.1.14 IEC HDLC setup (class_id: 23)

These COSEM objects define and control the behaviour of the device at the association negotiation instant using HDLC protocol. They are instances of the interface class “IEC HDLC setup".

IEC HDLC setup	IC	OBIS identification					
		A	B	C	D	E	F
IEC HDLC setup object	IEC HDLC setup	0	*x*	22	0	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channel.

D.2.1.15 IEC twisted pair (1) setup (class_id: 24)

These COSEM objects define and control the behaviour of the device regarding the communication parameters according to IEC 62056-31. They are instances of the interface class "IEC twisted pair (1) setup".

IEC twisted pair (1) setup	IC	OBIS identification					
		A	B	C	D	E	F
IEC twisted pair (1) setup object	IEC twisted pair (1) setup	0	*x*	23	0	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channel.

D.2.1.16 TCP-UDP setup (class_id: 41)

COSEM objects of the IC "TCP-UDP setup" handle all information related to the setup of the TCP and UDP layer of the Internet based communication profile(s) and point to the IPv4 setup object(s) handling the setup of the IP layer on which the TCP-UDP connection(s) is (are) used.

TCP-UDP setup	IC	OBIS identification					
		A	B	C	D	E	F
TCP-UDP setup object	TCP-UDP setup	0	x	25	0	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channels used for Internet based communication.

D.2.1.17 IPv4 setup (class_id: 42)

COSEM objects of the IC "IPv4 setup" handle all information related to the setup of the IP layer of the Internet based communication profile(s) and point to the data link layer setup object(s) handling the setup of the data link layer on which the IP connection(s) is (are) used.

IPv4 setup	IC	OBIS identification					
		A	B	C	D	E	F
IPv4 setup object	IPv4 setup	0	*x*	25	1	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channels used for internet-based communication.

D.2.1.18 Ethernet setup (class_id: 43)

COSEM objects of the IC "Ethernet setup" handle all information related to the setup of the Ethernet data link layer of the Internet based communication profile(s).

Ethernet setup	IC	OBIS identification					
		A	B	C	D	E	F
Ethernet setup object	Ethernet setup	0	x	25	2	0	255

The usage of value group B shall be:

- if more than one object is instantiated in the same physical device, the value group B shall number the communication channels used for internet-based communication.