DCSAP - Data Concentrator Simple Acquisition Protocol v. 3.0.1 EN

Document revision: 104, export date: Dec 18, 2020

Document in Polish: Rewizja dokumentu: 33, data wygenerowania: gru 18, 2020

Table of contents

1. DCSAP protocol version history

Version	Date	Description	Completed
0.9	2013-01-16	Preparing 1st version of the document	Rafał Jurkiewicz Michał Mirosław Przemysław Pawełczyk Paweł Pisarczyk Dominik Bocheński
0.9.1	2013-02-10	Expanding a glossary with terms HTTPS and URL. Correcting a drawing of DCSAP session to use correct OBIS objects codes. Changing the type of field dlms-size in the header of the message. DCSAP from Unsigned32 to INTEGER. Defining codes of errors at the level of the header of DCSAP messages. Correcting typographic errors in coding of sample commands and responses. Describing persistence of objects. Adding descriptions of all attributes of the introduced classes.	Przemysław Pawełczyk

		Splitting the network meter's identifier in the list of meters into an identifier of the meter's manufacturer and the names assigned by the manufacturer – the same scheme which is used in direct DLMS communication with meters [4]. Adding additional information on the software updating process and introducing the requirement of using HTTPS protocol. Defining class Device run information and the global object of the concentrator being its instance. Specifying the recommended size of the list of meters. Specifying recommendations about time synchronization – use of the same time servers like CBP/AMI application. Better layout and consistency in text formatting. Increasing legibility and facilitating navigation in the document.	
0.9.2	2013-02-19	Explaining a change in coding from BER to A-XDR. 4, libr. Limiting errors at the level of the messages' header. Changing the name of the field describing availability of the meter on the list of meters. Describing the update status codes. Adding classes Event list and String list as well as the objects that are using them. Changing in updating software to the fully asynchronical method, with the possibility of its cancellation. Expanding class Device firmware and adjusting the descriptions of attributes and methods. Defining time after which an empty message from MDG is supposed to be sent and the time after which the session is regarded as idle in DCU. Comment about the time and sequential no. of the last change of the record at readout of topology. Supplementing description of receiving events from the system, with a paragraph about the mechanism of notifications. Adding several sequence diagrams. Correcting recommendations about the number of simultaneously operated sessions and bandwidth of a single session.	Przemysław Pawełczyk
0.9.3	2013-02-20	Simplifying a descriptor of selective choice from class Meter list. Describing the codes returned by method restart() of class Device run information. Renaming the update status code EFWUPDATE to EMAINTAIN and expanding its meaning. Supplementing a description of method start_update() with immediate cancellation of a software update order if a device restart procedure was started previously. Changing the type of status fields in class Device run information to integer.	Przemysław Pawełczyk Rafał Jurkiewicz

		Describing objects Data concentrator event list and Data concentrator NTP server list. Describing the events logged in the events log of the concentrator. Removing an unnecessary object Data concentrator configuration id. Correcting several diagrams of sequences and adding new ones. Describing reference implementation.	
0.9.4	2013-02-25	Describing common features of objects as well as introducing the rules of numbering new classes and objects. Simplifying the records of class Meter list and introducing class Device basic information as well as object Meter information. Emphasis atomic nature and completeness of changes taking place in object Data concentrator meter list. Correcting and adding several sequence diagrams.	Przemysław Pawełczyk
0.9.5	2013-02-26	Reformatting and providing details to the history of changes. Changing the type of field dlms-data in the header of message DCSAP from INTEGER to Integer32. Correcting the interface classes cardinality, which according to standard refers to logical devices. Adding a table enumerating all interface classes introduced for the needs of DCSAP protocol and their cardinal character from the point of view of DCU and the meter.	Przemysław Pawełczyk
0.9.6	2013-02-27	Adding additional information on the method of referring to objects of meters performed by the concentrator. Adding class DCSAP network statistics. Adding object Data concentrator network statistics. Changing the used integer types in the reference implementation to typedef (at places where a specific size of transferred numbers is expected), which ensures better portability of code and improves its legibility.	Przemysław Pawełczyk Rafał Jurkiewicz
0.9.7	2013-02-27	Emphasis compulsory character of implementing the notification mechanism. Simplifying and introducing a minor change in the rules of selection of OBIS codes. Renumbering the existing objects. Expanding structure event_list_entry with the sequential number of the event and adding handling selective choice from attribute event_log of class Event list. Defining behaviour of object Data concentrator event list in the case of achieving a maximum number of records. Specifying types of attributes value of objects Data concentrator meter data caching enable and Data concentrator asynchronous notification enable and their default values. Correcting the sequence of messages sent when starting the session. Removing recommendations about the notification buffer because the transience of notifications (and responses) is not permitted anywhere, and suggesting	Przemysław Pawełczyk Rafał Jurkiewicz

		specific implementation of the guarantee of delivery notification (and its parameters) is an issue that is too specific to be examined in this document. Removing recommendations about addressing measuring systems because the manner of the concentrator's assigning numeric identifiers of devices (used by the acquisition system in addressing the messages of DCSAP protocol) is an implementation detail that in no way affects the course and quality of communication between the acquisition system and these devices.	
0.9.8	2013-03-04	Updating chapter "Object Data concentrator meter data caching enable"	Rafał Jurkiewicz
0.9.9	2013-03-04	Reviewing and adding general amendments to the document.	Dominik Bocheński
1.0	2013-03-05	Updating chapter "Description of reference implementation". Editor's corrections.	Rafał Jurkiewicz
1.0.1	2013-03-07	Reviews and general corrections to the document	Dominik Bocheński
1.0.2	2013-03-12	Reviews and general corrections to the document	Dominik Bocheński
1.0.3	2013-05-14	Correction of the sequence diagram for 4th case of the concentrator software update failure. Updating and adding recommendations with regard to the size of the meters list, events log and NTP server list. Emphasizing the optionality of the software image verification in the process of its update in the meters.	Rafał Jurkiewicz
1.0.4	2014-10-17	Clarifications of the requirements for the concentrators.	Tomasz Piasecki Michał Mirosław
2.0	2014-05-12	New design of class and objects scheme implemented by data concentrator. DCSAP header structure and error codes changes. Clarification of the relationship meter – data concentrator for ZKB case. Notation change: SAK -> MDG.	Michał Mirosław
2.0.1	2016-07-13	Changes and amendments consolidation for lines 1.x and 2.x of specifications, the specification addendum related to secured communication issues	Tomasz Piasecki
2.0.2	2016-07-15	Review and making document coherent. Templates unification and document editorial correction.	Tomasz Piasecki Rafał Jurkiewicz Michał Popiołkiewicz Bartosz Nowak
3.0.0	2020-12-03	Return to the schema of objects and classes from version 1.0.4.	Tomasz Piasecki

		Updating objects and classes to those currently used in practice. Adding an attachment with a data model. Document review	Marek Białowąs Ziemowit Leszczyński Kaja Swat
3.0.1	2020-12-18	Editorial corrections. Dicument verification.	Daniel Jarosik Jacek Górski Marek Białowąs Kaja Swat

Table 1 DCSAP protocol version history

2. Introduction

DCSAP protocol has been prepared for communication between the system of measurement data acquisition (SAK) and the concentrators of electrical power meters (DCU), constituting intermediate parties in communication with meters.

The basis in acquisition using DCSAP protocol is the assumption on communication with concentrators using a stream of acquisition instructions transferred by bidirectional, lossless connection, maintained with each concentrator. Instruction of acquisition (operation) is a single command for read/write of the attribute or calling the method of register of the measuring unit/concentrator. Acquisition by means of DCSAP has been presented schematically in Figure below.

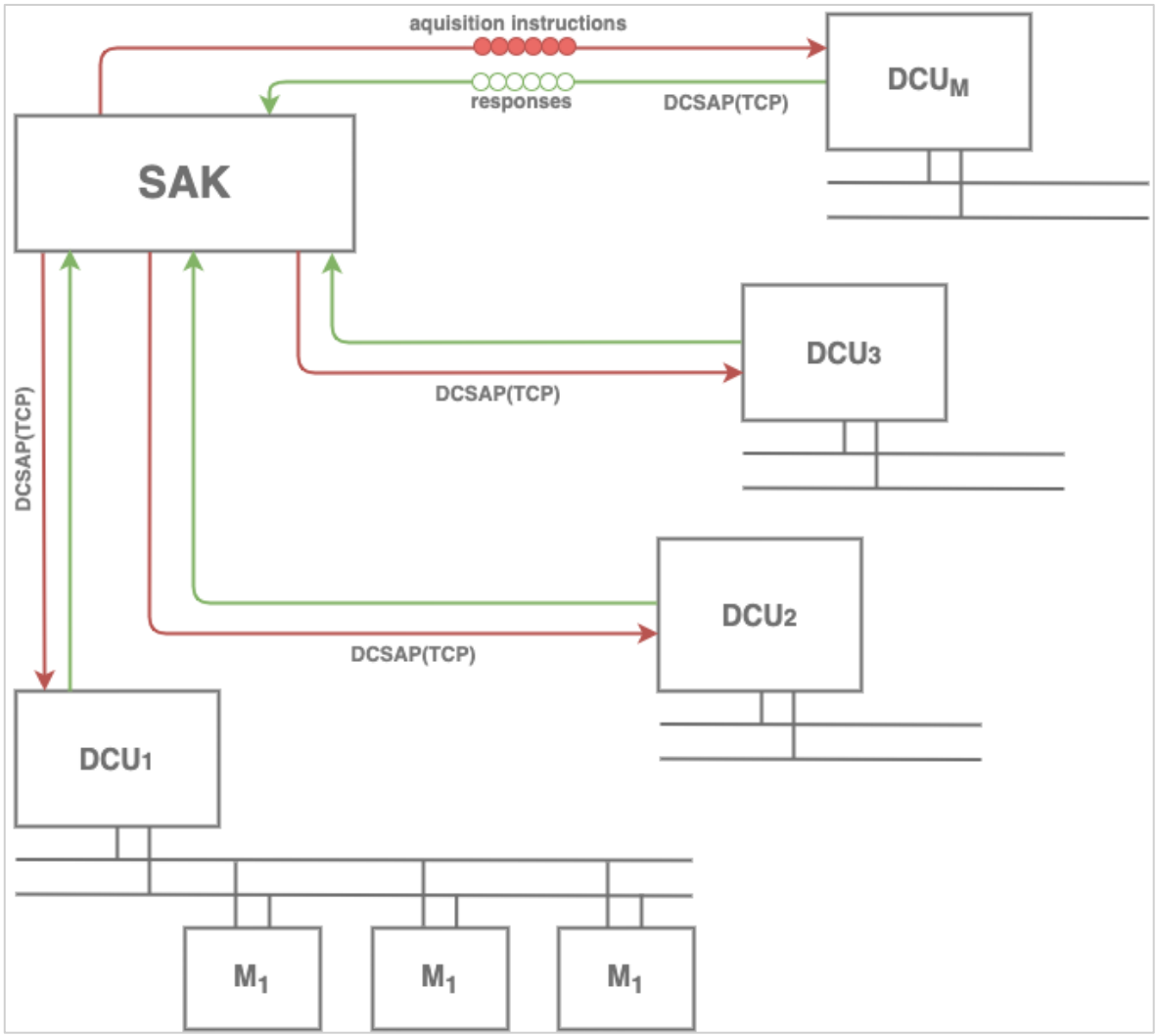

Figure 1 Acquisition of data based on DCSAP protocol

Acquisition system (SAK) establishes connections with all concentrators and then orders them execution of various operations. They include operations for changing state of concentrators and meters. Most of operations, however, applies to measurement data readout. The same connection is used for collecting results of the ordered measurement data readout operations and asynchronous notifications of events coming from the concentrator and meters.

Communication with concentrators and meters is implemented in a uniform way, i.e. instructions of acquisition for measuring units and concentrators have the same format and apply to measurement or control objects, available both in measurement units and in concentrators. Method of transferring instructions for acquisition to a concentrator and measuring systems is presented schematically in figure below.

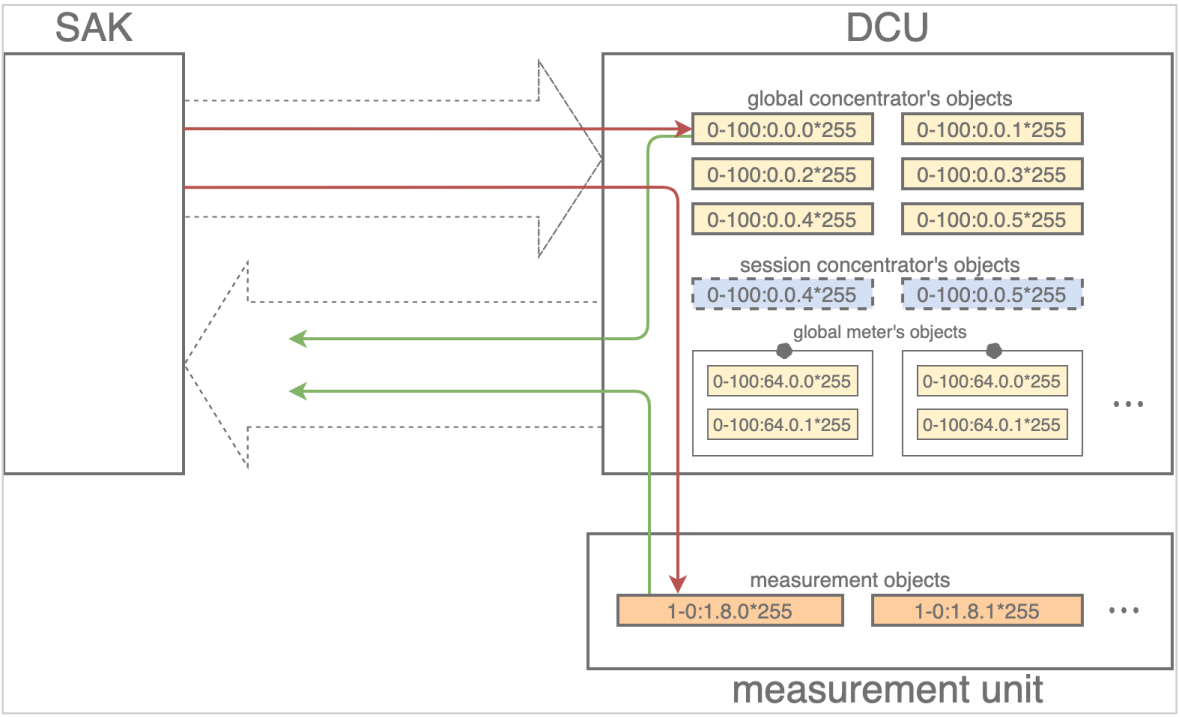

Figure 2 DCSAP session

Above the stream of instructions addressed to the concentrator's objects as well as to the objects of measuring units is marked. It has been assumed that the structure of the concentrator's objects and measuring units (class of these objects) is consistent with standard COSEM [1], [5].

The objects of the concentrator are used to implement its basic functions, such as downloading the list of available measuring units, information about network topology of these units, software updates, etc. Global objects have been distinguished as part of the concentrator's objects, shared between the sessions and session objects, available only within a given connection. Session objects have been introduced for the purpose of control of the parameters of a single session with the acquisition system. A global object is e.g. an object containing a list of available measuring units.

Objects of measuring units are standard objects implementing measurement and control functions.

The subsequent chapters of the document describe: assumptions for communication by DCSAP protocol, syntax and semantics of communication, structure of the concentrator's objects (global and session), examples of use of DCSAP protocol (use cases), recommendations for implementation of the protocol and the prepared reference implementation of the server of DCSAP protocol, which may be integrated with the concentrator's software.

3. Assumptions of the protocol

The protocol has been prepared in accordance with the following assumptions.

Maintaining network session with DCU

Communication between the server of the acquisition system and concentrator takes place through a single TCP/IP connection forming sessions. The session is not temporarily limited and may be maintained as long as telecommunication conditions permit it. At the same time, presence of option *keepalive* TCP connection is not assumed – acquisition system, in the case of lack of requests, periodically sends an empty message that the concentrator must send back in an unchanged form. This allows checking whether the connection with the concentrator has not been lost. Connection is established from the side of the acquisition system. For this reason all concentrators must be reached using addresses in the IP layer for the acquisition system. The acquisition system may establish many session with a single DCU.
At least 3 parallel sessions have to be supported.

Idempotent operations and session separation

All operations and their results are sent as part of one session and do not propagate to subsequent ones. In the case of closing or terminating the TCP/IP connection, all non-executed commands are cancelled. It is assumed that the acquisition system, after the start of another session, will re-order operations, whose execution has not been so far confirmed. Such a solution results in the possibility of multiple order of a given operation if it has been performed in the previous session, and the confirmation has not been delivered. It has not been decided to counteract such situations, for example by the use of unique identifiers of operations. As a result, the protocol and its implementation become easier and there is no need to maintain any context propagating between the sessions on the side of a concentrator. For this reason all commands must be idempotent, and the responsibility for maintenance of the desired condition of concentrators and meters is moved to the acquisition system.

Asynchronous transfer of responses

Responses to the ordered commands are transferred back to the acquisition system using the same TCP/IP connection. This process transfer takes place asynchronically as compared to the sequentially transferred operations, thanks to which the acquisition system may order next operations before it receives confirmation of execution of previous ones. The same way as responses asynchronous notifications from the concentrator and meters are transferred. The command processing order is determined by data concentrator with regard to *priority* in DLMS commands.

Use of DLMS/COSEM protocol

DCSAP protocol must ensure the possibility of executing DLMS commands on particular meters. For this reason, it was decided to use protocol DLMS as base and all additional functionalities were reached by defining specific COSEM objects representing logic of the concentrator. There is a possibility to define additional, specific objects used to control unique mechanisms prepared by the manufacturers of concentrators. It means that the DCSAP protocol is easily expandable and will catch up with new demands. Data concentrator is fully responsible for management of DLMS associations with meters. It is assumed that following client association types are supported:

- Reading (ID = 2) - for reading selected parameters
- Public (ID = 16) – for meter identification
- Management (ID=1) – for regular cooperation with the acquisition system
- Firmware Update (ID=3) – for meter firmware update

- Pre-Established (ID = 102) - for emergency broadcast

Due to the efficiency of the data acquisition process, it is recommended that the concentrator keep the Management associations continuously open with all served meters.
Mechanism of delivery and management of information needed to authenticate and encrypt the communication is beyond the scope of this specification.

Expanded addressing DLMS

In the communication layer, the DLMS protocol has been broadened by the identifier /device address to which a given command applies. As a result, after establishing a session with the concentrator, DLMS commands can be transferred for all meters operated by a given concentrator. The concentrator itself has also a well specified identifier in this space and may be the addressee of DLMS commands.

Optional protection with SSL

Encryption of communication may be implemented using the SSL protocol at the level of TCP/IP connection. It is, however, optional and it is assumed that management of keys and other issues related to enciphering will be solved outside the DCSAP protocol. It is possible to control optional mechanisms of the concentrator related to encryption using the DCSAP protocol, provided that the manufacturer defines appropriate COSEM objects intended for this.

4. Communication

In communication between the acquisition system and concentrators there are used messages coded according to standard A-XDR [3] Originally, it was planned to use BER coding, but because A-XDR is used in DLMS communication and in COSEM application layer, coding the same data in a smaller number of bytes than other communication protocols dedicated to *Smart Metering* [10] solutions allow it, it was finally decided to use more effective coding.

The basis is DLMS message to which a device identifier, message identifier and DLMS message size, becoming an error code in the case of negative values, were added. Listing DCSAP-PDU below presents description in format ASN.1. Types *Unsigned32*, *Unsigned64* and *Integer32* are defined in document [2], [6] and these are *INTEGER* types with relevant scopes of accepted values, and thus occupy respectively 4, 8 and 4 bytes in the coded form. In this way, the message header has always constant size amounting to 16 bytes. Type xDLMS-APDU was defined in the same document.

```
DCSAP-PDU ::= SEQUENCE
{
       device-id      Unsigned32     -- device identifier
       message-id     Unsigned64     -- message identifier
       data-size      Integer32      -- DLMS data size or error code
       dlms-data      xDLMS-APDU     -- DLMS data (field present only when data-size> 0)
}
```

Table 2 Structure of the message of DCSAP protocol

The DCSAP-PDU messages are used to send commands from the acquisition system to the concentrator as well as to send responses and notifications about the events from the concentrator to the acquisition system. Messages are sent asynchronically in both directions. Meaning of subsequent fields of the message are explained in comments to Listing.

A device identifier indicates the concentrator or meter registered in it, which is the addressee of commands. In the case of responses coming from the concentrator, a device identifier indicates the sender of the message. In the case of responses or notifications, it determines from which meter or concentrator it comes. 0 value means concentrator, on the other hand subsequent values are dynamically allocated by the concentrator to detected and registered meters. Policy of allocation of identifiers depends on the manufacturer of the concentrator.

The command message identifier must be used by the concentrator in the message being a response to it. Policy of using messages identifiers depends on the acquisition system, but should be used to link responses with a command, especially if the acquisition system sends subsequent commands not waiting for answers to previous ones. In the notification message, this field must be set to value 0.

DLMS data size must be consistent with further content of the message. This value is used for separation of messages without the need to analyse DLMS data structure. Non-positive values mean there is no DLMS data, namely there is no field *dlms-data*. Value 0 is used to control the connection and such a message must be sent back by the concentrator in an unchanged form. Negative values may occur only in responses of the concentrator to improperly addressed requests, incorrectly completed, not receiving responses from the target device in the maximum time planned by the concentrator or addressed to the devices, in which maintenance activities have been initiated earlier (like software update), requiring cessation of standard functions by these devices for some time. The error codes returned at that time are listed in the **table below**.

Types of data used in DLMS messages are described in detail in document **[2]**, **[7]**.

EC	Symbol	Description
-1	*EUNKNOWN*	Unknown device identifier.
-2	*EWRONGSIZE*	Incorrect data size (negative).
-3	*EPARTIAL*	Incomplete data.
-4	*EINVALID*	Incorrect data.
-5	*ETIMEOUT*	Waiting time for response from a device expired.
-6	*EINACCESSIBLE*	Device is knowingly unavailable (e.g. during updating).
-11	EASKLATER	Request cannot be fulfilled at this time (e.g. some values are not cached yet)
-12	EINCONSISTENTTARGET	Single request has sub-commands that cannot be handled together (eg. local and remote objects)
-13	EINTERNALERR	DCU internal error
-14	EINVALIDRESP	Meter returned incorrect response (w.r.t. dlms protocol or lower layers)

-15	EHANDSHAKEFAIL	Unable to establish communication with meter (eg. AARQ was rejected by the meter)
-16	EACCESS	Request was not processed by the target device because sending requests is not permitted

Table 3 Error codes available in messages of DCSAP protocol

Error codes may be submitted only by the concentrator. It should use them only if the message is improperly addressed, incorrectly built, an attempt to obtain response to the request expressed in it to the target device is unsuccessful in time, which the concentrator deems as maximum or the concentrator initiated an order previously whose realisation requires temporary cessation of provision of dedicated services by the device. These errors should not be confused with the errors provided in types *Data-Access-Result* and *Action-Result* (used in responses), related to communication in the DLMS/COSEM layer.

Commands

The following types and options of DLMS messages may be used in the DCSAP commands:

```
get-request [192] IMPLICIT Get-Request
    get-request-normal [1] IMPLICIT Get-Request-Normal
    get-request-with-list [3] IMPLICIT Get-Request-With-List

set-request [193] IMPLICIT Set-Request
    set-request-normal [1] IMPLICIT Set-Request-Normal
    set-request-with-list [4] IMPLICIT Set-Request-With-List

action-request [195] IMPLICIT Action-Request
    action-request-normal [1] IMPLICIT Action-Request-Normal
    action-request-with-list [3] IMPLICIT Action-Request-With-List
```

Table 4 DLMS request messages

DLMS additionally defines options of messages which apply in the case when frames in the protocol of data link have limited maximum size. DCSAP operates in higher layer and uses streaming TCP protocol, which does not have such a restriction. On the other hand, in communication of a concentrator with meters, existence of the mentioned restriction is highly probable, therefore the concentrator must then "repack" the request to the meter so it can be sent to the used medium.
For data requests from the registers of Class 7 (profile_generic, data from attribute 2 - buffer) support of selective-access mechanism in full form is required, i.e.:

```
range_descriptor ::= structure
{
    restricting_object: capture_object_definition
    from_value: CHOICE
    to_value: CHOICE
    selected_values: array capture_object_definition
}

entry_descriptor ::= structure
{
    from_entry: double-long-unsigned
    to_entry: double-long-unsigned
    from_selected_value: long-unsigned
    to_selected_value: long-unsigned
}
```

Table 5 Range and entry descriptors

as described in documents [1] and [7].

The concentrator should handle data type date-time taking into account the time offset in accordance with the specified time zone and daylight saving time in force (summer or winter time). The concentrator should not assume that the received timestamps are in the same time zone as the concentrator or meter clock.

DLMS messages of commands in the field *invoke-id-and-priority* have the bit *priority*, which means a higher priority request. The concentrator, receiving a message with a bit *priority* set should handle the request contained in it before the previously unrealised requests that do not have this bit set.

The remaining bits in the field *invoke-id-and-priority* are not used by the acquisition system, therefore the concentrator, when necessary, may modify them freely – field *invoke-id-and-priority* in a response, does not have to be consistent with *invoke-id-and-priority* of the command.

The following are the examples of commands and a manner of their coding:

```
DCSAP-PDU
   00 00 00 01                                device-id (= 1)
   00 00 00 00 00 00 01 01                    message-id (= 257)
   00 00 00 0D                                data-size (= 13)
   xDLMS-APDU
      C0                                      get-request
      Get-Request
         01                                   get-request-normal
         Get-Request-Normal
            00                                invoke-id-and-priority (priority = 0)
            Cosem-Attribute-Descriptor        cosem-attribute-descriptor (= 3/1-
0:1.8.0*255/2)
               Cosem-Class-Id                 class-id (= 3)
                  00 03
               Cosem-Object-Instance-Id       instance-id (= 1-0:1.8.0*255)
                  01 00 01 08 00 FF
               Cosem-Object-Attribute-Id      attribute-id (= 2)
                  02
            Selective-Access-Descriptor       access-selection (= nothing)
               OPTIONAL
                  00                          (= not present)
```

Table 6 Request for A+ of the meter with identifier 1

```
DCSAP-PDU
   00 00 00 0B                            device-id (= 11)
   00 00 00 00 00 01 00 01                message-id (= 65537)
   00 00 00 12                            data-size (= 18)
   xDLMS-APDU
      C1                                  set-request
      Set-Request
         01                               set-request-normal
         Set-Request-Normal
            00                            invoke-id-and-priority (priority = 0)
            Cosem-Attribute-Descriptor    cosem-attribute-descriptor (= 7/1-
0:99.2.0*255/8)
               Cosem-Class-Id             class-id (= 7)
                  00 07
               Cosem-Object-Instance-Id   instance-id (= 1-0:99.2.0*255)
                  01 00 63 02 00 FF
               Cosem-Object-Attribute-Id  attribute-id (= 8)
                  08
            Selective-Access-Descriptor   access-selection (= nothing)
               OPTIONAL
                  00                      (= not present)
            Data                          value
               06                         double-long-unsigned (= 200)
                  00 00 00 C8
```

Table 7 The request for setting buffer size for measurements latched in the profile with the second period of latching to the meter with identifier 11

```
DCSAP-PDU
   00 00 00 0F                            device-id (= 15)
   00 00 00 00 00 00 01 02                message-id (= 258)
   00 00 00 0C                            data-size (= 12)
   xDLMS-APDU
      C3                                  action-request
      Action-Request
         01                               action-request-normal
         Action-Request-Normal
            80                            invoke-id-and-priority (priority = 1)
            Cosem-Method-Descriptor       cosem-method-descriptor (= 70/0-
0:96.3.10*255/1)
               Cosem-Class-Id             class-id (= 70)
                  00 46
               Cosem-Object-Instance-Id   instance-id (= 0-0:96.3.10*255)
                  00 00 60 03 0A FF
               Cosem-Object-Method-Id     method-id (= 1)
                  01
```

Table 8 Request for disconnection of the meter's relay with identifier 15

Responses

The following types and options of DLMS messages may be used in the responses to DCSAP commands:

```
get-response [196] IMPLICIT Get-Response
    get-response-normal [1] IMPLICIT Get-Response-Normal
    get-response-with-list [3] IMPLICIT Get-Response-With-List

set-response [197] IMPLICIT Set-Response
    set-response-normal [1] IMPLICIT Set-Response-Normal
    set-response-with-list [5] IMPLICIT Set-Response-With-List

action-response [199] IMPLICIT Action-Response
    action-response-normal [1] IMPLICIT Action-Response-Normal
    action-response-with-list [3] IMPLICIT Action-Response-With-List
```

Table 9 DLMS response messages

If a large response from a meter is split to many frames at the transmission of the concentrator, it should be "re-packed" into one answer transmitted to the acquisition system. For one message of the command sent by the acquisition system to a concentrator, there should always come one message with an answer of operation in a response.

The concentrator should provide the date-time data filling the field *deviation* according to the source's time zone if the field has the value UNSPECIFIED in the original data.

The examples of responses for the previously presented examples of commands and a manner of their coding is presented in the subsequent page of this sub-clause.

```
DCSAP-PDU
   00 00 00 01                               device-id (= 1)
   00 00 00 00 00 00 01 01                   message-id (= 257)
   00 00 00 0D                               data-size (= 13)
   xDLMS-APDU
      C4                                     get-response
      Get-Response
         01                                  get-response-normal
         Get-Response-Normal
            00                               invoke-id-and-priority
            Get-Data-Result                  result (= 54132)
               00                            data
               Data
                  15                         long64-unsigned
                  00 00 00 00 00 00 D3 74    (= 54132)
```

Table 10 Response to the request for A+ of the meter with identifier 1

```
DCSAP-PDU
   00 00 00 0B                                  device-id (= 11)
   00 00 00 00 00 01 00 01                      message-id (= 65537)
   00 00 00 04                                  data-size (= 4)
   xDLMS-APDU
      C5                                        set-response
      Set-Response
         01                                     set-response-normal
         Set-Response-Normal
            00                                  invoke-id-and-priority (priority = 0)
            Data-Access-Result                  result (= 3 - read-write-denied)
               03
```

Table 11 Answer to the request for setting buffer size for measurements latched in the profile with the second period of latching from the meter with identifier 11

```
DCSAP-PDU
   00 00 00 0F                                  device-id (= 15)
   00 00 00 00 00 00 01 02                      message-id (= 258)
   00 00 00 05                                  data-size (= 5)
   xDLMS-APDU
      C7                                        action-response
      Action-Response
         01                                     action-response-normal
         Action-Response-Normal
            80                                  invoke-id-and-priority
            Action-Response-With-Optional-Data    single-response
               Action-Result                    result (= 0 - success)
                  00
               Get-Data-Result                  return-parameters (= nothing)
                  OPTIONAL
                     00                         (= not present)
```

Table 12 Response to the request for disconnection of the meter's relay with identifier 15

Notifications

Notifications contain the following type of DLMS message:

```
event-notification-request[194] IMPLICIT EventNotificationRequest
```

Table 13 Event notification

The mechanism of notifications is used for asynchronous notification of the acquisition system about the events and changes in the concentrator and meters. This is an optional mechanism from the utilization point of view (and by default it is disabled in any new session), but its support by the concentrator is obligatory. If the notification mechanism is enabled in more than one session - identical copies of information about events

and changes shall be provided in each of those sessions. The acquisition system may alternatively periodically check the condition of relevant objects. Such check is much less efficient, but it may turn out to be the only solution for these concentrators with which communication proceeds through medium without the algorithms of avoiding and detecting collisions. In such a case the acquisition system must fully control the transmission in both directions. This can be guaranteed by performing single commands and waiting for answers to them and not activating the asynchronous notification transmission.

Example of notification and a manner of its coding is given below in this sub-clause.

```
DCSAP-PDU
   00 00 00 7F                          device-id (= 127)
   00 00 00 00 00 00 00 00              message-id (= 0)
   00 00 00 0C                          data-size (= 12)
   xDLMS-APDU
      C2                                event-notification-request
      Event-Notification-Request
         OCTET STRING                   time (= nothing)
            OPTIONAL
               00                       (= not present)
         Cosem-Attribute-Descriptor     cosem-attribute-descriptor (= 7/0-
0:99.98.0*255/2)
            Cosem-Class-Id              class-id (= 7)
               00 07
            Cosem-Object-Instance-Id    instance-id (= 0-0:99.98.0*255)
               00 00 63 62 00 FF
            Cosem-Object-Attribute-Id   attribute-id (= 2)
               02
         Data                           attribute-value
            FF                          dont-care
```

Table 14 Notification about the event registered by a meter with identifier 127 in the first event log.

5. COSEM objects

The concentrator, in order to perform its function must implement several basic mechanisms. Control as well as retrieval of action results of these mechanisms takes place through specific attributes and methods of relevant COSEM objects defined for a concentrator. These attributes and methods are generalized by means of classes of COSEM interfaces, associated with those objects. The acquisition system uses objects for the concentrator in the same manner as the objects stipulated for meters.

Specification of DCSAP protocol contains a minimum set of such objects and describes their semantics. Manufacturers of devices can present their specific mechanisms and the objects implementing an additional functionality related thereto. In this way, DCSAP protocol is easily expandable.

Specific COSEM objects of the concentrator have been divided into two main categories. The first one applies to global mechanisms, allowing to read and change the current state of a concentrator. The second group are session objects, related to the state of the current session. Their changes are not propagated between the sessions, both initiated one after another or maintained simultaneously. It can be said that set of temporary objects is related to each session that can be read and used to control this specific session.

Global objects of meters performed by the concentrator have been defined additionally. They are performed by the software of the concentrator, but they exist as independent instances for each registered meter. They are available from the level of the DCSAP protocol, except the standard specification of a COSEM method or

attribute descriptor in a relevant type of DLMS message, it is required to include the identifier of a device belonging to a meter (instead of the concentrator, so other than 0) in the header of this message.

All objects performed by the concentrator and defined as part of this documentation have the following characteristics:

1. The requests concerning them and operating on many objects (i.e. *Get-Request-With-List*, *Set-Request-With-List* or *Action-Request-With-List*) cannot refer in one and the same request to the objects which are not performed by the concentrator (e.g. objects implemented in meters).
2. Realisation of the requests referring to them is immediate and also synchronous, i.e. they are operated immediately (the only delays result from delays of access to the desired or changed objects) and without reaching to external resources (i.e. located beyond the concentrator or requiring communication with external devices), what additionally facilitates implementation.
3. The notifications concerning them are sent only after the end of a change or update to which they apply, rather than in the course of it or at the beginning of its execution.

Attention

Objects which are considered architecture-dependent thus optional are marked in blue.

Rules for numbering new classes and objects

To avoid identification chaos during defining new classes as well as specific objetcs using them, it has been decided to introduce several rules to govern this process.

Selection of new identifiers of interface classes (class_id)

For the purpose of interface classes of the objects implemented by the concentrator and those required during communication by means of DCSAP protocol, only range 40000–40199 will be used. This range is a part of a greater range 32768–65535 planned for user groups (*user group specific ICs* see chapter 4 in [1], [7]).

There are 2 sub-ranges defined as part of the chosen range:

- 40000-40099 – for interface classes of the objects dedicated only to concentrators,
- 40100-40199 – for interface classes of the objects for general purpose.

The numbers granted to class interfaces within this document are successive numbers from relevant sub-ranges, which have not been used yet.

In the case of external suppliers who want to use the scheme presented here, allocation of numbers should proceed from the end of these sub-ranges, which will allow avoiding collisions in the case of possible document revisions extending the normally required list of objects.

Selection of new OBIS codes

OBIS Codes of objects implemented by the concentrator and required during communication with DCSAP protocol are defined within the subgroup of codes for utility applications (*utility specific* see chapter 5 in [1]) where the group of B values must be from range 65–127.

The OBIS codes granted to objects as part of this document are subsequent, possibly the lowest, numbers from the pools defined in table below.

A separate pool of objects is planned for external suppliers wanting to use the scheme described here. It is recommended to systemize it internally in the same manner like it was performed with other pools.

Object pools	OBIS code					
	A	B	C	D	E	F
Global objects of the concentrator	0	100	0-31	d	e	255
Session objects of the concentrator	0	100	32-63	d	e	255
Global objects of a meter realized by the concentrator	0	100	64-95	d	e	255
Objects defined by suppliers	0	100	128-255	d	e	255

Table 15 Rules of selection of OBIS codes for new objects

d, e- any value not causing collision with pre-defined objects.

Interface Classes

For the needs of defining some structures performed by the concentrator there have been introduced specific, dedicated classes. They are mentioned in table below and described in the further part of this sub-clause.

Interface class name	Class number class_id	Cardinality in the concentrator	Cardinality in the meter
Meter list	40000	1	0
Event list	40001	1	0
Network statistics	40002	1	0
Initconfig	40051	1	0
Modem parameters	40052	1	0
Firmware control	40054	1	0
Emergency control	40055	1	0
Events dump control	40056	1	0
Cumulative statistics	40057	1	0
Profile special cpobj	40060	0	0…n
String list	40100	1…n	0

Device firmware	40101	1	1
Device basic information	40102	0…1	1
Device run information	40103	1	0
Profile config	40160	0	1
DLMS Security Setup	40199	0	1…n

Table 16 Interface Classes introduced for the needs of DCSAP protocol

The following subchapters describe all these specific interface classes.

Meter list (class_id=40000)

Meter list			**1**	**class_id = 40000, version = 1**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	meter_table	(dyn.)	array of meter_list_entry			
3	entires_in_use	(dyn.)	double-long-unsigned			
4	max_entries	(stat.)	double-long-unsigned			
5	extended_meter_list	(dyn.)	extended_meter_list_struct			
Specific methods:			***m/o***			

Attribute description	
meter_table	Table of structures (see Listing meter_list_entry) representing information about meters. Selective choice is possible (see Listing recent_entries_descriptor).
entires_in_use	Number of entries in table. When it reaches maximum, new entries overwrite the entries of the oldest-updated invisible meters.
max_entries	Maximum number of entries in table.
extended_meter_list	Table of structures (see Listing extended_meter_list_entry) representing information about meters. Selective choice is possible (see Listing recent_entries_descriptor).

```
meter_list_entry ::= structure
{
   last_change_seq_id   long64-unsigned
-- sequential number as part of the whole table of the last record change

   last_change_time date-time
-- time of last record change

   id   double-long-unsigned
-- numerical meter identifier assigned by the concentrator

   manufacturer octet-string
-- identifier of the meter's manufacturer (3 capital characters)

   name octet-string
-- meter name assigned by the manufacturer (max. 16 capital characters)

   present  boolean
-- meter is visible by the concentrator
}
-- network identifier of a meter constitutes combination of fields manufacturer and
name
```

Table 17 Structure of a single entry in attribute meter_table of class Meter list

```
-- required access-selector = 1, access-parameters assumes the following form:

recent_entries_descriptor ::= long64-unsigned

-- item selection, whose last_change_seq_id is greater than the value stated
```

Table 18 Descriptor of selective choice of entries from attribute meter_table of class Meter list

```
extended_meter_list_entry ::= structure

{
   seq_id                long64-unsigned
-- sequential number as part of the whole table of the last record change

   seq_time              double-long-unsigned
-- Time of last status change (UNIX time)

   id                    double-long-unsigned
-- device identifier assigned by DCU

   name                  octet-string[13]
-- COSEM Logical Device Name (LDN) of the meter

   conn_status           unsigned
-- The meter connection status to DCU: 0 -> not connected / 1 -> connected to the PLC
topology /
-- 2 -> DLMS connection possible

   prof_range            structure
              {
                     oldest                double-long-unsigned
                     newest                double-long-unsigned
                     capture-period        double-long-unsigned
                     error                 long
                     auto_readout          boolean
              }
-- The state of the profile cached by the concentrator,
-- either explicitly specified by access selector or the one with the lowest positive
capture-period.


   prof_counters         structure
              {
                     completed             unsigned
                     total                 unsigned
              }
-- How many profiles do have the full set of configuration data on the DCU,
-- and how many should these data have.

   slot_stats            structure
              {
                     time                  long64-unsigned
                     count                 double-long-unsigned
                     tx_bytes              double-long-unsigned
                     rx_bytes              double-long-unsigned
              }
-- How many milliseconds the concentrator used for DLMS communication with the meter,
-- how many DLMS queries were sent and how much raw bytes were transferred.

   got_met_conf          boolean
-- True if the concentrator already has the meter passport downloaded.

   topo           structure
              {
                     lnid                  long-unsigned
                     sid                   unsigned
                     lsid                  unsigned
                     mac                   octet-string[6]
              }
-- Current PRIME topology: lnid, sid, lsid and mac. If there is no connection, these
fields
-- are set to 0xffff. In the absence of information about the mac number, the returned
-- number is zero. Other connection means (eg. RS485) may reuse these fields with
different meaning.
```

```
   errors                  double-long-unsigned
-- Bit mask of communication errors with the meter detected by the concentrator
-- (Device basic information / error_flags)

   proto                   unsigned
-- Meter connection with DCU medium type: (0:unknown, 1:internal 2:PLC, 3:RS485)

   fwupdate                integer
-- The last known firmware update status (Device firmware / last_update_status)

   assoc_errors            array[4] of assoc_error
               {
                      client_id            unsigned
                      errors               double-long-unsigned
               }
-- per-association error flags (differentiable by association client id)
-- (DLMS Security Setup / error_flags)

   feature_flags           long-unsigned
-- meter feature flags (to be used by the client to prepare correct types of queries)

   fw_version              data
-- meter firmware version (Device firmware / version) or null if not available

   security                unsigned
-- current security setup of MGMT association - upper 4 bits (0xf0) encode
security_policy,
-- lower 4 bits encode authenticaction_mechanism_id

   conn_stats              structure
               {
                      conn_event_cnt        double-long-unsigned
                      connected_secs        double-long-unsigned
                      dcu_uptime            double-long-unsigned
               }
-- meter connection stability stats - numer of reconnections and total connected time
estimate
}
```

Table 19 Structure of a single entry in attribute extended_meter_list of class Meter list

```
-- required access-selector = 1, access-parameters assumes the following form:

recent_entries_descriptor ::= structure
{
   field_mask            long-unsigned -- or double-long-unsigned
-- Bit mask specifying which counter statistics to return (default: 0xffff)

   seq_id                long64-unsigned
-- Only meters with a higher seq_id will be returned

   max_cnt               double-long-unsigned
-- The maximum number of meters to be returned in response

   range_prof_obis       octet-string[6]
-- Profile OBIS code, according to which will return the "prof_range" field. By
default,
-- it is the profile with lowest positive capture-period.
}

-- available field bits in field_mask:
   meter_table_ex_field__meter_id          [1 << 0]
   meter_table_ex_field__meter_name        [1 << 1]
   meter_table_ex_field__seq_id            [1 << 2]
   meter_table_ex_field__seq_time          [1 << 3]
   meter_table_ex_field__connection_status [1 << 4]
   meter_table_ex_field__prof_range        [1 << 5]
   meter_table_ex_field__prof_counters     [1 << 6]
   meter_table_ex_field__slot_stats        [1 << 7]
   meter_table_ex_field__got_met_conf      [1 << 8]
   meter_table_ex_field__topo              [1 << 9]
   meter_table_ex_field__errors            [1 << 10]
   meter_table_ex_field__proto             [1 << 11]
   meter_table_ex_field__fwupdate          [1 << 12]
   meter_table_ex_field__assoc_errors      [1 << 13]
   meter_table_ex_field__feature_flags     [1 << 14]
   meter_table_ex_field__fw_version        [1 << 15]
   meter_table_ex_field__security          [1 << 16]
   meter_table_ex_field__conn_stats        [1 << 17]

-- The result is returned as a structure:

extended_meter_list_struct structure
{
   field_mask            double-long-unsigned
-- Bit mask specifying which meter statistics were returned. By default, this should be
-- the bit mask from the query access-selector, but may have fewer bits if
-- the DCU application does not support some types of information
-- (e.g. when it is an older version of the DCU software)

   extended_meter_list   array of extended_meter_list_entry
-- Meter statistics table, one entry per meter
}
```

Table 20 Descriptor of selective choice of entries from attribute extended_meter_list of class Meter list

Event list (class_id=40001)

Event list	**0...1**	**class_id = 40001, version = 1**		
Attributes	***Data type***	***Min.***	***Max.***	***Def.***

1	logical_name	(stat.)	octet-string			
2	event_log	(dyn.)	array of event_list_entry			
3	entires_in_use	(dyn.)	double-long-unsigned			
4	max_entries	(stat.)	double-long-unsigned			
5	event_mask	(dyn.)	array[2] of bit-string[256]			
Specific methods			***m/o***			
1	push(event)		m			

Attribute description	
event_log	Table of structures (see **Listing event_list_entry**) representing the information about events. Selective choice is possible (see **Listing recent_entries_descriptor**).
entires_in_use	Number of entries in table.
max_entries	Maximum number of entries in the event table.
event_mask	array[0] – system log event write mask: 0-no / 1-yes array[1] – event notification mask: 0-no / 1-yes
Method description	
push(event)	Causes adding a description of the event transferred in the argument to the table. The field *reason* is overwritten with the value 255 (*EV_PUSH*) , the field *seq_id* is overwritten with subsequent event number and the field *time* is overwritten with the current concentrator clock time. *event ::= event_list_entry*

```
event_list_entry ::= structure
{
   seq_id          long64-unsigned
-- event sequential number

   time            double-long-unsigned
-- event occurrence time (UNIX time)

   device_id       double-long-unsigned
-- numerical device identifier to which the event applies

   reason          unsigned
-- reason for occurrence or recording of the event

   status          integer
-- status code associated with the reason, 0 if not applicable

   recorded_data   data
-- additional data associated with the event

   comment         octet-string
-- additional comment

   device_name     octet-string
-- COSEM Logical Device Name (LDN, empty for DCU)
}
```

Table 21 Structure of a single entry in attribute event_log of class Event list

It is required that the event sequence number maintains the order of events occurrence regardless of changes in the concentrator clock settings (registered event with a lower number occurred no later than the event with a higher number).

```
-- required access-selector = 1, access-parameters assumes the following form:

recent_entries_descriptor ::= long64-unsigned

-- item selection, whose last_change_seq_id is greater than the value stated
-- access-selector enabling pagination and filtering,
-- required access-selector = 2, access-parameters assumes the following form:

recent_entries_descriptor ::= structure
{
   is_backward ::= boolean
-- sort order of the requested items

   max_count ::= double-long unsigned
-- pagination range, if is_backward, return this much from the end

   first_seq_id ::= long64-unsigned
-- starting seq_id

   device_id ::= double-long
-- device identifier filter; -1 to ignore

   event_reason ::= double-long
-- event reason filter; -1 to ignore
}
```

RC	Symbol	Description
0	*EV_START*	Device start-up. Fields *status* and *recorded_data* take then accordingly the values of attributes *last_start_status* and *start_count* of an object of class *Device run information*.
1	*EV_RESTART*	Device restart order. The field *status* takes then the value of the field *result* transferred in response to the customer calling the method *restart()* of an object of class *Device run information* (code *success* or error *temporary-failure*). Field *recorded_data* is completed *null-data*.
2	*EV_UPDATEINIT*	Device software update order. The field *status* takes then the value of the field *result* transferred in response to the customer calling the method *start_update()* of an object of class Device firmware (*success* code or *temporary-failure* error). The field *recorded_data* takes then the value of attribute *last_update_id* of an object of class *Device firmware* (the new, i.e.already incremented, in the case of success or the present in the case of an error).
3	*EV_UPDATEFINI*	End of updating software of a device. The fields *status* and *recorded_data* take then accordingly the values of attributes *last_update_status* and *last_update_id* of an object of class *Device firmware*.
4	*EV_METERSTAT*	Change in presence of a meter.

		The field *status* takes then a new value of the field *present* from the relevant record *meter_list_entry* of the list of meters. The field *recorded_data* is completed with structure (*structure*) containing the field *manufacturer* and *name* from the same record.
128	*EV_TIME_CHANGED*	Time was just changed to a new value The old time is stored in *recorded_data* field. Applies to manual date change (not NTP)
129	*EV_TIME_VALIDATION*	Time is now assumed valid/invalid (based on NTP synchronization status) Field *status* is equal to 1 if the time is assumed as valid, is equal to 0 when the time is assumed as invalid.
130	*EV_CONF_CHANGE*	DCU configuration change requested In the *status* field DLMS result is stored while *recorded data* stores changed value (module_name, diff or old value).
131	*EV_TAMPER_CHANGE*	Tamper status changed *status*: 1-cover closed (OK) / 0-cover opened (tampered) *device_name*: tamper cause (cable, main, accel, magnet) *data*: empty or X/Y/Z axis
132	*EV_TEMPERATURE_ALERT*	High temperature detected *status*: 1-temerature raised above threshold / 0-temperature dropped below threshold *data*: temperature threshold (in miliCelsius)
133	*EV_EMERGENCY_BROADCAST*	Emergency control activated *status*: 1-emergency broadcast started / 0-emergency broadcast cancelled *data*: emergency_profile sent
134	*EV_POWER_FAIL*	Power fail detected (inputted only externally) *status*: 1: power fail begin / 0: power fail end *device_name*: empty or power rail name if multiple power rails present
135	*EV_EXT_CTRL_RESTART*	External controller restart detected
136	*EV_FACTORY_RESET*	Reset to factory defaults *status*: 1: intentional reset (requested by user) / 0: involuntary reset (because of device state)
144	*EV_NEW_METER_SEEN*	A meter has been spotted for the first time
145	*EV_METER_DELETED*	An old meter has been marked for deletion from the database
146	*EV_METER_CLOCK_SET*	Meter clock synchronization was performed

		device_id: meter identifier, *device_name*: meter COSEM Logical Device Name (LDN) *data*: algorithm+parameter, where algorithm =“set” (DLMS SET on the attribute 2) or ”shift_time” (call of the shift_time method) parameter: previous clock value (for “set”) or shift value (shift_time parameter sent, for “shift_time”), *status*: DLMS result code
147	*EV_METER_PROFILE_DATA*	Meter profile data request completion *device_id*: meter identifier, *device_name*: meter COSEM Logical Device Name (LDN) *status*: positive value is DLMS result code; negative is related to profile request error *data*: OBIS without class and attribute), 'from' data as UNIX time in GMT, 'to' as UNIX time in GMT, rows count
148	*EV_METER_ERROR*	Meter error flags changed (either global or per-association) *device_id*: meter identivication, *dev_name*: meter COSEM Logical Device Name (LDN) *status*: 1-setting error flag / 0- clearing error flags *data*: DLMS client ID, error flags (if client ID is 0 then each bit corresponds to one of meter error flag, else to association error flag)
149	*EV_EXT_CTRL_UPDATE*	External controller firmware updated *data*: stores old version and new version identifiers
160	*EV_CONNECTION*	A connection has been established *device_name* : name of one of connection channels, *data*: IP, port and local_port *status*: 0-closed / 1-opened
161	*EV_LOGIN*	A login attempt has been made *device_name*: name of one of connection channels, *data*: user name, *status*: 0-failure / 1-success / -1-blocked
254	*EV_CUSTOM*	Custom event Field *dev_name* is the name of the source device while *recorded_data* is the comment
255	*EV_PUSH*	The event resulting from execution of the method *push()*.

Table 22 Codes of reasons for occurrence or writing of the events that may be assumed by the field reason of the structure event_list_entry and descriptions of associated fields

Network statistics (class_id=40002)

Network statistics			0…1	class_id = 40002, version = 0		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	sessions_opened	(dyn.)	long64-unsigned			
3	sessions_active	(dyn.)	long64-unsigned			
4	bytes_received	(dyn.)	long64-unsigned			
5	bytes_sent	(dyn.)	long64-unsigned			
6	messages_received	(dyn.)	long64-unsigned			
7	messages_sent	(dyn.)	long64-unsigned			
8	dcu_reqs_completed	(dyn.)	long64-unsigned			
9	meter_reqs_cempleted	(dyn.)	long64-unsigned			
Specific methods			***m/o***			

Attribute description	
sessions_opened	Number of all previously open sessions as part of the current start-up of a concentrator.
sessions_active	Number of currently kept sessions as part of the current start-up of a concentrator.
bytes_received	Number of bytes received in the application layer from all sessions as part of the current start-up of a concentrator.
bytes_sent	Number of bytes sent in the application layer from all sessions as part of the current start-up of a concentrator.
messages_received	Number of messages received in all sessions as part of the current start-up of a concentrator.
messages_sent	Number of messages sent in all sessions as part of the current start-up of a concentrator.
dcu_reqs_completed	Number of executed orders directed to the objects implemented by the concentrator in all sessions as part of the current start-up of a concentrator.
meter_reqs_cempleted	Number of executed orders directed to the objects realised by meters in all sessions as part of the current start-up of a concentrator.

Initconfig (class_id=40051)

Contains mainly static DCU application configuration properties for a given platform/client, also some of them can be dynamically changed (affect the operation of the DCU application).

Initconfig			**1**	**class_id = 40051, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
16	conn_slots	(dyn.)	double-long			
17	conn_slots_hard_limit	(dyn.)	double-long-unsigned			
18	max_conn_meters	(dyn.)	double-long-unsigned			
32	max_db_meters	(dyn.)	double-long-unsigned			
33	pagecache_kb	(dyn.)	double-long-unsigned			
34	heap_kb	(dyn.)	double-long-unsigned			
48	profile_max_capture_object_cnt	(dyn.)	double-long-unsigned			
49	profile_cnt_per_meter	(dyn.)	double-long-unsigned			
50	prof_conf_cache_cnt	(dyn.)	double-long-unsigned			
51	past_hours_gather_new	(dyn.)	double-long-unsigned			
52	keep_profile_data_for_secs	(dyn.)	double-long-unsigned			
53	keep_inactive_meters_for_secs	(dyn.)	double-long-unsigned			
54	cache_cleaner_rescan_every_secs	(dyn.)	double-long-unsigned			
55	keep_profile_data_min_entries	(dyn.)	double-long-unsigned			
56	keep_profile_data_max_entries	(dyn.)	double-long-unsigned			
57	keep_profile_data_max_entries_plc	(dyn.)	double-long-unsigned			
58	past_hours_gather_gap	(dyn.)	double-long			
59	auto_prof_readout_disabled	(dyn.)	boolean			

60	max_prof_rows_per_req	(dyn.)	double-long-unsigned			
64	channel_cnt	(dyn.)	double-long-unsigned			
65	remote_msg_size	(dyn.)	double-long-unsigned			
66	remote_msg_cnt	(dyn.)	double-long-unsigned			
67	local_msg_size	(dyn.)	double-long-unsigned			
68	local_msg_per_channel_cnt	(dyn.)	double-long-unsigned			
69	remote_query_timeout_sec	(dyn.)	double-long-unsigned			
80	dlms_max_resp_size	(dyn.)	double-long-unsigned			
96	device_identifier	(stat.)	octet-string			
97	device_identifier2	(stat.)	octet-string			
98	system_version	(dyn.)	octet-string			
112	band	(stat.)	double-long-unsigned			1
113	mac_bc	(dyn.)	boolean			true
114	phy_bc	(dyn.)	boolean			false
115	frame_len	(dyn.)	double-long-unsigned	0	3	1
Specific methods			***m/o***			

Attribute description	
conn_slots	how many concurrent active communications with meters can be handled (“connection slots”), value “-1” means that the actual number of connection slots will be decided dynamically by the DCU depending on current PLC network size (default: -1)
conn_slots_hard_limit	hard limit for the *conn_slots* (default: 16)
max_conn_meters	maximum number of concurrently open connections to the meter (eg. PRIME CON, default: 2048)
max_db_meters	maximum number of simultaneously supported meters (DB and RAM storage, default: 2048)
pagecache_kb	amount of RAM memory reserved for DB page cache (default: 4096 kB)

heap_kb	amount of RAM memory reserved for DB heap (default: 1024 kB)
profile_max_capture_object_cnt	maximum number of capture objects in any profile kept in DCU cache (default: 48)
profile_cnt_per_meter	*(depreciated, not used anymore)*
prof_conf_cache_cnt	number of distinct profile configurations (tuple: *capture_objects*, *capture_period*, *max_entries*) handled simultaneously by the DCU (default: 100)
past_hours_gather_new	amount of past profile data to be gathered - for newly connected meters (default: 168 hours)
keep_profile_data_for_secs	amount of time after which cached meter profile data will be deleted from the DCU (default: 5443200 seconds)
keep_inactive_meters_for_secs	amount of time after which inactive meters will be deleted from the DCU (default: 7776000 seconds)
cache_cleaner_rescan_every_secs	interval between consecutive cache cleaning operations (default: 21600 seconds)
keep_profile_data_min_entries	for profiles with capture_period=0, keep at least x profile entries regardless of the keep_profile_data_for_secs time constraint (default: x=250)
keep_profile_data_max_entries	for profiles with capture_period=0, keep at most x profile entries regardless of the keep_profile_data_for_secs time constraint (default: x=9100)
keep_profile_data_max_entries_plc	the same as above, but only for plc-connected meters (so: not internal/billing meters, default: 3000)
past_hours_gather_gap	amount of past profile data to be gathered - for reconnecting meters (default: -1 -> no gaps)
auto_prof_readout_disabled	disable automatic profile readout for all meters (false: auto readout enabled, true: auto readout disabled, default: false)
max_prof_rows_per_req	maximum numer of profile rows to request with single automatic request (default: 4)
channel_cnt	maximum number of concurrent sessions to the DCU (either local by Web-GUI or remote, default: 16/20)
remote_msg_size	maximum size of the “remote” DLMS message (“remote” means message to the meter (not handled by the DCU itself, but passed to the meter)) - this constraint is the same for request and response size (default: 4096 bytes)
remote_msg_cnt	maximum number of pending remote messages (total, not per channel, default: 160)

local_msg_size	maximum size of the “local” DLMS message (either message to DCU or to the meter but handled by the DCU, default: 204800 bytes)
local_msg_per_channel_cnt	maximum number of pending local messages (per channel, default: 1)
remote_query_timeout_sec	default timeout for remote queries (can be changed per-session by special session object, default: 1200 seconds)
dlms_max_resp_size	maximum DLMS response size (from the single meter) supported by the DCU (default: 16384 bytes)
device_identifier	unique device identifier (format 1)
device_identifier2	unique device identifier (format 2 if supported)
system_version	system_version (if separate from DCU firmware version), currently empty on all platforms
band	PRIME 1.4 only: FCC PHY band (set of channels) - each bit corresponds to one of the PRIME channels (LSB = bit 0 → channel 1; MSB = bit 7 → channel 8)
mac_bc	PRIME 1.4 only: MAC backward compatibility mode (PRIME macBCMode PIB)
phy_bc	PRIME 1.4 only: PHY backward compatibility mode - used only if mac_bc is false
frame_len	PRIME 1.4 only: MAC frame len (0-3) (PRIME macFrameLength PIB) - used only if mac_bc is false

Modem parameters (class_id=40052)

these are the custom properties of a given multimodem (modem interface of some medium that supports connecting many meters at once) - each attribute is of type double-long unsigned and only SET is available (GET returns READ_WRITE_DENIED), currently it is implemented only for PRIME Base Node multimodem. Some features might not be available on certain hardware targets.

Modem parameters			**1**	**class_id = 40052, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	pcap_cnt_limit	(dyn.)	double-long unsigned			
3	pcap_file_reset	(dyn.)	double-long unsigned			
Specific methods			***m/o***			

2	send_broadcast_packet()	m	

Attribute description	
pcap_cnt_limit	enable / disable dropping of PRIME packets into the PCAP file (the limit of packets dropped set to 0 – disables dropping)
pcap_file_reset	reset the PCAP file
Method description	
send broadcast packet()	sends transferred data (octet-string encoded) as a PRIME broadcast packet (in DLMS CL connection)

Firmware control (class_id=40054)

Class to manage the firmware upgrade mechanism of meters managed by the concentrator.

Description of operation:

- first all the rules tried to match to the meters, the one with the lowest priority are chosen,

- attempts are made at: changing the schedule, connecting the meter for which the current version of the schedule has not yet been checked, after the firmware upgrade

- matching checks: wildcard based on the name of the meter ('?' as any character), meter version (literal matching) or visible string (substring in the firmware version, for firmware file as structures containing subcomponents individual firmwares)

- *update_cmd* is a parameter for method 3 of class *Device Firmware* object (*start_conditional_update*) will perform firmware upgrade only if the device is no longer in the target version (second parameter) - first it will download the current version directly from the meter (needed if the meter was updated when it was connected to another concentrator as part of crosstalk).

Firmware control			**1**	**class_id = 40054, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	distinct_versions	(dyn.)	array of version_list_entry			
3	schedule	(dyn.)	array of schedule_list_entry			
4	fw_slots_current_state	(dyn.)	fw_slots_current_state_struct			
5	fw_unicast_slots_cnt	(dyn.)	double-long unsigned			

6	broadcast_min_receivers	(dyn.)	double-long unsigned			
7	broadcast_freq_ms	(dyn.)	double-long unsigned			
8	broadcast_done_perc	(dyn.)	double-long unsigned			
9	meter_broadcast_support_forced	(dyn.)	unsigned			
10	meter_broadcast_support_forbidden	(dyn.)	unsigned			
11	is_updater_paused	(dyn.)	unsigned			
12	disable_fw_version_check_before_upgrade	(dyn.)	unsigned			
Specific methods			***m/o***			

Attribute description	
distinct_versions	The table which presents the quantitative distribution of firmware versions of meters Table of structures (see **Listing version_list_entry**) representing information about firmware version cardinality
schedule	list of rules that allows to automatically (independently) start an FW meter upgrade (details below) Table of structures (see **Listing schedule_list_entry**) representing information for schedule (automatic start) definition of firmware meter upgrade via DCU
fw_slots_current_state	The state of the currently upgraded unicast counters (8 parallel unicast upgrade sessions are running as standard), the attribute has a type described in **Listing fw_slots_current_state_struct**
fw_unicast_slots_cnt	number of possible parallel unicast upgrade sessions (default: 8)
broadcast_min_receivers	the minimum number of currently upgrading meters to switch automatically to broadcast (default: 2)
broadcast_freq_ms	what how many ms to send another broadcast packet with upgrade (default: 6000ms)
broadcast_done_perc	how many packets (percentage of *active_fw_blocks_cnt*) must be received by broadcast for a meter (default: 200%)
meter_broadcast_support_forced	first to try broadcast firmware upgrade, even if the meters do not support it
meter_broadcast_support_forbidden	do not broadcast even if the meters support it (it is a must to enable it if the simultaneous update of different meters by different files is

	necessary - otherwise broadcast on one would destroy other firmware upgrade)
is_updater_paused	pause the entire firmware upgrade mechanism for meters (no new packets will be sent, but the current state and firmware upgrade will not be lost)
disable_fw_version_check_before_upgrade	do not download the current firmware version of the meter before starting the firmware upgrade

```
version_list_entry ::= structure
{
   count          double-long-unsigned
-- number of meters with particular firmware version

   version        data
-- firmware version label (typically octet-string or struct, depending on manufacturer)
}
-- particular firmware version cardinality
```

Table 23 Structure of a single entry in attribute distinct_versions of class Firmware control

```
schedule_list_entry ::= structure
{
   prio           double-long-unsigned
-- priority (lowest value is chosen to process)

   name_wildcard  octet-string[13]
-- wildcard by which the appropriate firmware is matched based on the name of the meter
-- ('?' as any character),

   curr_fw        data
-- wildcard by which the appropriate firmware is matched based on the
-- meter version (literal matching) or visible string (substring in the firmware
version,
-- for firmware file as structures containing subcomponents individual firmwares)

   update_cmd     data
-- parameter for method 3 of class device firmware object (start_conditional_update)
-- will perform firmware upgrade only if the device is no longer in the target version
-- (second parameter) - first it will download the current version directly
-- from the meter (needed if the meter was updated when it was connected to another
-- concentrator as part of crosstalk)
}
```

Table 24 Structure of a single entry in attribute schedule of class Firmware control

```
fw_slots_current_state_struct ::= structure
{
   device_logical_name    octet-string
-- COSEM Logical Device Name (LDN)

   time_on_slot          double-long-unsigned
-- number of seconds this meter occupies the slot

   progress              unsigned
-- percentage progress <0-100>

   state                 unsigned
-- internal status of meters firmware upgrade machine state in DCU
}
-- particular firmware version cardinality
```

Table 25 Structure of attribute fw_slots_current_state of class Firmware control

Emergency control (class_id=40055)

Class added to meet the requirements of emergency power off.

Emergency control			**1**	**class_id = 40055, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	broadcast_repeat_cnt	(dyn.)	double-long-unsigned			
3	broadcast_freq_ms	(dyn.)	double-long-unsigned			
Specific methods			***m/o***			
1	start(emergency_profile)		m			
2	cancel()		m			

Attribute description	
broadcast_repeat_cnt	emergency requests packets repeat count (default: 20)
broadcast_freq_ms	emergency requests packets frequency (default 2000 ms)
Method description	
start(emergency_profile)	starts broadcast packages sending (in the context of pre-estabilished association) with request Set-Request-Normal (71/0-0.17.0.1.255/8, *emergency_profile*), if emergency

	broadcast sending is in progress - returns DLMS error TEMPORARY_ERROR, if permission error - returns DLMS error READ_WRITE_DENIED
cancel()	if the emergency broadcast sending is in progress - it is interrupted (not sending new packets), NOTE: this does not cause *emergency_profile* to be turned off on the side of the meters that have already received it (to do this, call the *start()* method with a properly prepared *emergency_profile*, or (preferably) communicate by unicast with each of the meters)

Events dump control (class_id=40056)

Objects of this class are used to control the mechanism of DCU application events recording to an external CSV file, e.g. timestamp + data during registration / de-registration, connection / disconnection of each counter (allows analysis of PRIME topology development in time).

Events dump control			**1**	**class_id = 40056, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	state	(dyn.)	enum			
3	max_limits	(stat.)	max_limits_struct			
4	counts_curr	(dyn.)	counts_curr_struct			
5	timestamps	(dyn.)	timestamps_struct			
6	app_events_mask	(stat.)	double-long-unsigned			
Specific methods			***m/o***			
1	reset()		m			

Attribute description	
state	enum: • 0 not initialized • 1 running • 2 paused
max_limits	structure (see #max_limits_struct) defining limits for events storage (count and byte size)
counts_curr	structure (see #counts_curr_struct) showing current events collected counter and bytes size

timestamps	structure (see #timestamps_struct) showing the timestamps range of (first/last) event collected
app_events_mask	bitfield of event types to be captured (see #app_event_mask_values)
Method description	
reset()	reset the counters and range stored in *counts_curr* and timestamps attributes

```
max_limits_struct ::= structure
{
   max_events            double-long-unsigned
-- maximum number of events to store in CSV file

   max_bytes             double-long-unsigned
-- maximum bytes to store in CSV file
}
-- limits for events (number and size to store in CSV files)
```

Table 26 Structure of attribute max_limits of class Events dump control

```
counts_curr_struct ::= structure
{
   curr_events           double-long-unsigned
-- current events collected counter

   curr_bytes            double-long-unsigned
-- current events collected bytes size
}
-- current events collected counter and bytes size
```

Table 27 Structure of attribute counts_curr of class Events dump control

```
timestamps_struct ::= structure
{
   first_timestamp double-long-unsigned
-- timestamp of first event collected

   last_timestamp        double-long-unsigned
-- timestamp of last event collected
}
-- timestamps of first/last event collected
```

Table 28 Structure of attribute timestamps of class Events dump control

```
-- bit values for app_events_mask
   facility__prime      [1 << 0]    -- PRIME PLC
   facility__db         [1 << 1]    -- sqlite RAW VFS usage
   facility__dcu_conn   [1 << 2]    -- DCU meter connected/disconnected events
   facility__dcu_slots  [1 << 3]    -- DLMS slots usage stats
   facility__dcu_fwup   [1 << 4]    -- meter FW upgrade mechanism events
   facility__dcu_req    [1 << 5]    -- DCU DLMS request types/states
```

Table 29 Available event types for class Events dump control

Cumulative statistics (class_id=40057)

Statistics (number of meters in a given state: disconnected, registered, connected, per-medium type: unknown, internal, plc, rs485).

Cumulative statistics			**1**	**class_id = 40057, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	periodic_stats	(dyn.)	array of periodic_stats_list_entry			
3	current_stats	(dyn.)	current_stats_definition			
Specific methods			***m/o***			

Attribute description	
periodic_stats	Table of structures (see Listing periodic_stats_list_entry) representing information about number of meters in a given state, per-medium type at specific timestamp
current_stats	Structure (see Listing current_stats_definition) representing information about current number of meters in a given state, per-medium type

Medium is a type of communication over which the data is acquired (it can be internal communication methods, PLC PRIME but also PLC G3 or RS 485).

```
periodic_stats_list_entry ::= structure
{
   timestamp      double-long-unsigned
   {
      num_of_meters_in_conn_state_0_for_medium_type_0    long-unsigned
      num_of_meters_in_conn_state_1_for_medium_type_0    long-unsigned
      ...
   },
   {
      num_of_meters_in_conn_state_0_for_medium_type_1    long-unsigned
      num_of_meters_in_conn_state_1_for_medium_type_1    long-unsigned
      ...
   },
   ...
}
-- number of meters
-- in a given state (unknown, internal, plc, rs485)
-- per-medium type (disconnected, registered, connected)
-- at specific timestamp
```

Table 30 Structure of a single entry in attribute periodic_stats of class Cumulative statistics

```
current_stats_definition ::= structure
{
   {
      num_of_meters_in_conn_state_0_for_medium_type_0    long-unsigned
      num_of_meters_in_conn_state_1_for_medium_type_0    long-unsigned
      ...
   },
   {
      num_of_meters_in_conn_state_0_for_medium_type_1    long-unsigned
      num_of_meters_in_conn_state_1_for_medium_type_1    long-unsigned
      ...
   },
   ...
}
-- current number of meters
-- in a given state (disconnected, registered, connected)
-- per-medium type (0:unknown, 1:internal, 2:PLC, 3:RS485)
```

Table 31 Structure definition of attribute current_stats of class Cumulative statistics

Profile special cpobj (class_id=40060)

Used only with access-selectors in meter profiles (queries to the concentrator cache) - in addition to the data available in the capture objects of a given profile, the request can ask for the metadata stored on each line.

Profile special cpobj	**1**	**class_id = 40060, version = 0**		
Attributes	***Data type***	***Min.***	***Max.***	***Def.***

1	logical_name	(stat.)	octet-string			
2	prof_time	(dyn.)	double-long-unsigned			
3	recv_time	(dyn.)	double-long-unsigned			
4	row_idx	(dyn.)	double-long-unsigned			
5	complex_idx	(dyn.)	complex_idx_struct			
Specific methods			***m/o***			

Attribute description	
prof_time	UNIX timestamp calculated from DLMS date-time from the profile line (in GMT)
recv_time	UNIX timestamp from the moment this row is received from the meter (in GMT)
row_idx	incremental row index (independent per-counter, per-profile), never decreasing
complex_idx	`complex_idx_struct ::= structure { prof_time double-long-unsigned; recv_time double-long-unsigned; row_idx double-long-unsigned }` where prof_time, recv_time and row_idx are equal to the values of attributes 2, 3 and 4 respectively

String list (class_id=40100)

String list			**1**	**class_id = 40100, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	string_list	(stat.)	array of octet-string			
3	entires_in_use	(dyn.)	double-long-unsigned			
4	max_entries	(stat.)	double-long-unsigned			
Specific methods:			***m/o***			

Attribute description	
string_list	List of character strings
entries_in_use	Number of elements on the list.
max_entries	Maximum number of elements on the list.

Device firmware (class_id=40101)

Device firmware			**1**	**class_id = 40101, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	version	(dyn.)	octet-string			
3	checksum	(dyn.)	octet-string			
4	last_update_time	(dyn.)	date-time			
5	last_update_id	(stat.)	double-long-unsigned			
6	last_update_status	(dyn.)	integer			
7	last_update_progress	(dyn.)	unsigned			
Specific methods			***m/o***			
1	start_update(https_url)		m			
2	abort_update(update_id)		m			
3	start_conditional_update(data)		m			
4	update_fw_version()		m			

KSA	Symbol	Description
7	*DEVREBOOT*	Device is being restarted after uploading an image.
6	*DEVRELOAD*	Updated modules are being restarted. (Device is not restarted in this case)
5	*IMGWRITE*	Loading an image to a device is in progress.
4	*IMGBACKUP*	Creating backup of modules undergoing updating is in progress

3	*IMGVERIF*	Checking the image in terms of correctness or compliance with the device is in progress
2	*IMGDOWNLOAD*	Downloading image is in progress
1	*CERTVERIF*	Acquiring and checking the certificate returned by the HTTPS server at the address of the image is in progress
0	*SUCCESS*	Last update was completed successfully.
-1	*EFWABORT*	Update was cancelled upon request.
-2	*EMAINTAIN*	Update was cancelled due to previously initiated and unfinished maintenance operation, e.g. software update or device restart.
-3	*EWRONGCERT*	Update was cancelled because of an improper certificate, i.e. expired or signed by an unknown certification authority
-4	*EINVURL*	The update was cancelled because of a bad URL address, i.e. incorrect or not leading to the image file.
-5	*EINVCHKSUM*	Update was cancelled because of an unsuccessful verification of the image's checksum.
-6	*EUNAVAIL*	Update was cancelled because the device is unavailable.
-7	*EFWINVALID*	Update was cancelled because the image was chosen which is not compatible with the device.
-8	*EDEVABORT*	Update was cancelled by the device

Table 32 Status codes of software update that may be accepted by attribute last_update_status of class Device firmware

Attribute description		
2	**version**	The software version present on the device. In the case of many independently versioned modules, it should specify their particular versions.
3	**checksum**	Checksum of the current software on the device. In the case of multiple independent software modules the field should contain checksums of individual modules.
4	**last_update_time**	Time of last update start.
5	**last_update_id**	Sequential number of the last update.
6	**last_update_status**	Status of the last update. The positive values mean that updating is ongoing and they notify of its stage. 0 value means that the last update was successfully completed. Negative values

		mean that the last update was completed unsuccessfully and inform about the cause of this failure. The described codes of statuses are located in **Tab. 3**.
7	**last_update_progress**	Progress of the last update. It is assumes the values from the scope of 0-255, where 0 means that the last update was started, and 255 – indicates a successfully completed update. The intermediate values are assigned by the manufacturer according to his best knowledge, so as to reliably reflect the actual progress in acquisition, check and load of software to the device.
Method description		
1	**start_update(https_url)**	Order of device software update start. If the devices is currently undergoing another update or a restarting procedure is underway, an error *temporary-failure* is returned. Otherwise, the sequential number of the last update is increased and it is returned together with code *success*. Following this, proper updating starts. It is necessary to download the software image indicated in argument *https_url::=octet_string*, (`http://` and `local://` also supported, both for meters and DCU firmware upgrade actions) which contains the URL of the file available via the HTTPS protocol. It is preceded by verification of a certificate presented at establishing communication with a set server – whether it has expired or has been signed by an appropriate, well-known to the concentrator, certification authority. After downloading the image, a test of its integrity and compliance with the device is performed, after which the image is loaded to the memory of the device. During updating the attributes *last_update_status* and *last_update_progress* are being continuously updated.
2	**abort_update(update_id)**	It aborts, if it is possible at a given stage, updating software with the sequence number provided in the argument. *update_id::= double-long-unsigned* Use of sequential no. allows to avoid accidental cancellation of update that is different than the previously ordered (after its completion, another update could have been ordered, e.g. from the level of another session). In the case of successful interruption of updating, the *success* code is returned. In the case of duration of other update procedure than the specified by *update_id* parameter, the *object-unavailable* code is returned. If it is not possible to interrupt updating, the *hardware-fault* code is returned. Software update should be fully aborted before sending the *Action-Response* reply.

3	**start_conditional_update(data)**	`data ::= stucture`
		`{`
		`  https_url  octet-string`
		`  dest_fw_version  data`
		`}`

		method introduced to avoid unnecessary multiple updates, for some meters the firmware update order can be done by more than one concentrator at a time (in the case of crosstalks). Performs firmware update only if the current firmware version (attribute 2 - *version*) is different from *dest_fw_version*. The check will be performed after each connection of the meter to the DCU.
4	**update_fw_version()**	force meter FW version fetch as soon as possible

Device basic information (class_id=40102)

Device basic information			**0...1**	**class_id = 40102, version = 1**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	config_id	(dyn.)	double-long-unsigned			
3	passport	(dyn.)	octet-string			
4	error_flags	(dyn.)	double-long-unsigned			
5	meter_name	(dyn.)	octet-string[13]			
6	max_query_obises	(dyn.)	double-long-unsigned			
7	time_recv_quirks	(dyn.)	double-long-unsigned			
8	time_send_quirks	(dyn.)	double-long-unsigned			
9	max_response_bytes	(dyn.)	double-long-unsigned			
10	last_clock_diff	(dyn.)	double-long			
11	logic_quirks	(dyn.)	double-long-unsigned			
12	security_quirks	(dyn.)	double-long-unsigned			
Specific methods			***m/o***			
1	clear_error_flags(mask)		m			
2	delete()		m			
3	delete_profiles()		m			
4	disconnect_topo()		m			

5	disconnect_conn()	m	

Attribute description	
config_id	Identifier of full device configuration (changed at the time of any change in the device configuration).
passport	Manufacturer-specific sequence of data identifying the device parameters. Contains obligatorily the software model and version.
error_flags	Meter error bit mask (see Listing meter_error_flags)
meter_name	Meter name
max_query_obises	number of OBIS'es supported by meter in single request in case of multiple-references support
time_recv_quirks	coded clock handling errors for a given meter type (for time receive operations) - see Listing time_quirks
time_send_quirks	coded clock handling errors for a given meter type (for time set operations) - see Listing time_quirks
max_response_bytes	maximum size of full response to a single query (sum of all responses sizes in multi-block) - now 4kB
last_clock_diff	currently estimated difference between the time of the meter and the concentrator
logic_quirks	coded quirks for request logic errors for a given meter - see Listing logic_quirks
security_quirks	coded quirks for security-related errors for a given meter - see Listing security_quirks
Method description	
clear_error_flags(mask)	Clearing the meter error bits (only those indicated by mask)
delete()	Removing the meter from the DCU database (with all collected data)
delete_profiles()	Deleting meter profiles from the DCU database
disconnect_topo()	Meter full disconnection (deregistration) from the PLC topology
disconnect_conn()	Meter disconnection (closing the CON connection) in PRIME layer

One error can set more than one error bit. Errors are only stored until the next time the DCU application is started. In addition, they can be cleared for each meter.

Bit	Name	Description

0	dlms_format	DLMS response parsing error
1	dlms_exception	The meter sent a DLMS exception in response
2	dlms_multiblock	Error parse multiblock response
3	dlms_llc	LLC header error
4	dlms_flow	Unexpected response when DLMS session is already established
5	dlms_flow_aarq	Unexpected response when establishing DLMS session
6	-	<not used>
7	-	<not used>
8	-	<not used>
9	-	<not used>
10	time_inconsistent	Error handling time stamps in profile queries
11	profile_missing	Some profiles (cached by DCU) are not supported by the meter
12	-	<not used>
13	-	<not used>
14	-	<not used>
15	-	<not used>
16	dlms_transport	DLMS communication has been (temporarily) interrupted
17	dlms_rejected	The meter returned an error code in response to a DLMS query
18	-	<not used>
19	-	<not used>
20	-	<not used>
21	-	<not used>
22	fatal_profile	Some profiles collection has been discontinued
23	fatal_passport	Meter passport download has been discontinued
24	fatal_version	The meter software version download has been discontinued

25	fatal_clock	Fatal error while getting/setting clock
26	-	<not used>
27	-	<not used>
28	-	<not used>
29	-	<not used>
30	-	<not used>
31	fatal_generic	Generic fatal flag (automatic communication with this meter will not be continued)

Table 33 Description of the meter error bits

Bit	Name	Description
0	local	send: the clock value should be sent in DCU local time (if not set it would be sent as GMT time) receive: ignored
1	tzund	send: set deviation as undefined (0xffff) meaning "local timezone" receive: expect undefined deviation, treat it as DCU local timezone
2	no_dst_off	DST status is not included in devieation (deviation is the same during the whole year)
3	dt_dev_little_endian	deviation field incorrectly encoded (using LE instead of BE)

Table 34 Description of time quirks bits

Bit	Name	Description
0	meterconf_in_mgmt_assoc	meter passport is not available in PUBLIC association (note: this may make meter not encryptable)
1	round_as_end_time_to_capture_time	meter error: profile request return empty table if access selector "to_value" exceeds the newest captured row timestamp
2	fix_non_periodic_profile_capture_period	meter error: capture period of event logs is set to '1', change it to '0'
3	fix_datetime_in_cached_profiles	meter error: invalid datetime timezone/status in received profile rows -> change it to datetime in DCU timezone

Table 35 Description of logic quirks bits

Bit	Name	Description
0	no_initiate_enc	Do not encrypt/expect AARQ/AARE InitiateRequest/InitiateResponse fields to be encrypted

1	allow_bad_context_id	Allow to use LN-Plaintext instead LN-With-Ciphering context in AARE
2	allow_hls_rx_fc_mismatch	Allow RX FC in HLS response to be lower by 1 than the current RX FC
3	app_ctx_error_if_lls_invalid	meter returns AARE result "application-context-name not supported" for any error in AARQ (eg. LLS secret invalid)
4	initiate_auth_enc	InitiateRequest/InitiateResponse is authenticated & encrypted if security_policy != 0

Table 36 Description of security quirks bits

Device run information (class_id=40103)

Device run information			**0...1**	**class_id = 40103, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	start_count	(dyn.)	double-long-unsigned			
3	last_start_time	(dyn.)	date-time			
4	last_start_status	(dyn.)	integer			
5	curr_uptime_secs	(dyn.)	double-long-unsigned			
6	prev_start_time	(dyn.)	date-time			
7	prev_start_status	(dyn.)	integer			
8	prev_uptime_secs	(dyn.)	double-long-unsigned			
Specific methods:			***m/o***			
1	restart()		m			

Attribute description	
start_count	Number of previous device start-ups.
last_start_time	Time, when the last device start-up began.
last_start_status	Status of the last start-up. 0 means lack of problems.
curr_uptime_secs	How many seconds from the last device start-up.
prev_start_time	Time, when the previous device start-up began.

prev_start_status	Status of the previous start-up. 0 means lack of problems.
prev_uptime_secs	How many seconds did the device work after the previous start-up
Method description	
restart()	Device restart request. If the software updating procedure was started previously, which has still not completed, requesting restart is rejected and the *temporary-failure* error is returned. Otherwise, the *success* code is returned and the device restarting procedure begins.

Profile config (class_id=40160)

Returns an array of structures - profile metadata that are cached for this meter.

Profile config			**1**	**class_id = 40160, version = 0**		
Attributes			***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	(stat.)	octet-string			
2	profile_metadata	(dyn.)	array of profile_metadata_struct			
Specific methods			***m/o***			

Attribute description	
profile_metadata	array of structures (see Listing profile_metadata_struct) representing information about profile metadata that are cached for this meter

```
profile_metadata_struct ::= structure
{
   logical_name                  octet-string[8]
-- OBIS code of profile object (without class_id and attribute(s)

   capture_objects_array         array
                                 {
                                    capture_object_01     octet-string
                                    capture_object_02     octet-string
                                    ...
                                 },
-- array of capture_object (or null if empty)

   capture_period               double-long
-- value of the profile capture_period

   first_cached_timestamp       double-long-unsigned
-- UNIX timestamp of the first (oldest) row in cache (GMT)

   last_cached_timestamp        double-long-unsigned
-- UNIX timestamp of the last (newest) row in cache (GMT)

   row_cnt                      double-long
-- number of rows in the cache

   error_code                   long
-- error code when downloading this profile (positive value is DLMS error returned
-- by the meter, negative is logical error of returned data:
-- (-3: bad_timerange, -2: bad_format, -1: other)

   automatic_readout             boolean
-- whether DCU automatic cache readings for this profile are enabled
}
```

Table 37 Structure of a single entry in attribute profile_metadata of class Profile config

DLMS Security Setup (class_id=40199)

The class defines DLMS communication security parameters related to the corresponding COSEM objects present in the meters:

DLMS Security Setup		**1..n**	**class_id = 40199, version=0**		
Attributies		***Data type***	***Min.***	***Max.***	***Def.***
1	logical_name	octet-string			
2	security_policy	enum	0	3	0
3	authentication_mechanism_id	unsigned	0	5	1
4	secret	octet-string			(empty)
5	global_unicast_encryption_key	octet-string			(empty)
6	global_broadcast_encryption_key	octet-string			(empty)

7	global_authentication_key	octet-string			(empty)
8	error_flags	double-long-unsigned			0
Specific methods		***m/o***			
1	clear_error_flags(mask)	m			
2	force_framecounter_sync()	m			

Attribute description	
security_policy	defines enabling encryption and/or transmission authentication mechanisms in the context of Security Suite ID = 0 (Galois/Counter Mode with AES-128 encryption algorithm): enum: 0 - none 1 - all messages are authenticated 2 - all messages are encrypted 3 - all messages are authenticated and encrypted
authentication_mechanism_id	identifier of the data access mechanism (authentication): unsigned: 0 - COSEM_lowest_level_security_mechanism_name 1 - COSEM_low_level_security_mechanism_name 5 - COSEM_High_Level_Security_Mechanism_Name_Using_GMAC
secret	password (secret) used in LLS authentication mode with meter
global_unicast_encryption_key	key value global unicast encryption key - GUEK
global_broadcast_encryption_key	key value global broadcast encryption key - GBEK
global_authentication_key	key value (global) authentication key – GAK
error_flags	error flags in the association context (see **Listing assoc_error_flags**)
Method description	
clear_error_flags(mask)	force to clear error flags (*error_flags* attribute)
force_framecounter_sync()	force to synchronise frame counters

Bit	**Name**	**Description**
0	fatal_aarq_rejected	AARQ frame rejected

1	fatal_aarq_context_name_invalid	AARQ frame invalid context name
2	fatal_aarq_sec_mechanism_not_supp	AARQ security mechanism not supported
3	fatal_aarq_lls_auth	LLS authentication mechanism error
4	fatal_aarq_hls_auth	HLS authentication mechanism error
5	fatal_aarq_flow	AARQ frame flow error
6	-	<not used>
7	-	<not used>
8	fatal_fc_sync_error	frame counter synchronisation error
9	fatal_fc_rx_too_low	received RX frame counter is lower than expected
10	fatal_fc_rx_too_big	received RX frame counter exceeds FRAME_COUNTER_SAFE_MAX
11	fatal_fc_tx_too_big	supposed TX frame counter exceeds FRAME_COUNTER_SAFE_MAX
12	broad_fc_tx_too_big	supposed broadcast TX frame counter exceeds FRAME_COUNTER_SAFE_MAX
13	-	<not used>
14	-	<not used>
15	-	<not used>
16	aarq_default_pass	informative: to trigger password change to the correct one
17	unexpected_encryption	informative: the transmission should not be encrypted (but it is)
18	-	<not used>
19	-	<not used>
20	-	<not used>
21	-	<not used>
22	-	<not used>
23	-	<not used>
24	-	<not used>
25	-	<not used>

26	-	<not used>
27	-	<not used>
28	-	<not used>
29	-	<not used>
30	-	<not used>
31	fatal	global permanent "fatal" flag for this association

Table 38 Description of the meter error bits in the association context

6. COSEM data model of the DCU

Each data object of the DCU data model is identified by an OBIS (Object Identification System) code. Coding objects using OBIS codes is in accordance with IEC 62056-61 [6].

The DCU data model contains 3 main types of data objects:

- global objects of the concentrator
- session objects of a concentrator
- global objects of meters realised by a concentrator

Global objects of the concentrator

All global objects of meters realised by a concentrator (except DCU network statistics) are persistent, i.e. concentrator restart does not cause loss of their contents. These objects are used mainly for DCU configuration, parametrisation or collect statistic/debug information.

COSEM logical device name

These objects define classical DLMS/COSEM device behaviour - COSEM Logical Device Name (LDN) objects of the DCU:

Object	COSEM class_ID	OBIS code
COSEM logical device name	1	0-0:42.0.0.255

DCU identification number

Object	COSEM class_ID	OBIS code
ID of the device	1	0-0:96.1.0.255

Clock

Object	COSEM class_ID	OBIS code
Clock	8	0-0:1.0.0.255

Meters clocks synchronisation

Time synchronisation is a crucial topic for CBP/AMI system, so set of specific objects and special mechanisms were implemented to perform this task.

Two clock synchronisation algorithms are supported:

- "use_set" - uses the DLMS SET operation on attribute 2 of the clock object (sets the absolute time of the meter)
- "use_shift_time" - uses the DLMS ACTION operation on method 6 of the clock object (shift_time) - shifts the current clock value by value stored in "*Maximum time shift*" (0-100:128.1.9.255) object

The parameters to manage the meters clocks are available in the concentrator's COSEM model and can be changed using DLMS protocol:

Object	COSEM class_ID	OBIS code
Meter clock synchronisation on/off	1	0-100:128.1.1.255
Minimum time between synchronisation attempts	3	0-100:128.1.3.255
Minimum time deviation	3	0-100:128.1.5.255
Maximum time between synchronisation attempts	3	0-100:128.1.6.255
Meter clock verification cycle	3	0-100:128.1.7.255
Synchronisation algorithm	1	0-100:128.1.8.255
Maximum time shift	3	0-100:128.1.9.255
Minimum cycle for time shift	3	0-100:128.1.10.255

Description of the algorithms

Algorithm 0 – "use_set":

1. When connecting the meter for the first time (after DCU restart):
    1.1. current clock value is collected -> (2)
2. Collection of the clock value: command DLMS GET CLOCK/2 is sent. Depending on the result:
    2.1. In the case of a DLMS meter error, the attempt is repeated to retrieve the clock value at "now" + "*Minimum time between synchronisation attempts*" object (0-100:128.1.3.255) value -> (2)

2.2. In case of success, the moment of the next clock value collection is set to "now" + "*Meter clock verification cycle*" object (0-100:128.1.7.255) value and checking of the value is performed -> (3)

3. Checking the value of the clock collected from the meter. If at least one of the conditions is true:

3.1. the absolute value of the time difference between the meter and the concentrator is greater than "*Minimum time deviation*" object (0-100:128.1.5.255) value

3.2. or the status byte of the returned clock contains the CLOCK_STATUS_DOUBTFUL_VALUE flag

3.3. or the status byte of the returned clock contains the CLOCK_STATUS_INVALID_VALUE flag

then: setting the meter clock is performed immediately -> (4)

4. Meter clock setting: DLMS SET CLOCK/2 request is sent. Depending on the result:

4.1. In the case of a DLMS meter error, the attempt to set the clock is repeated at "now" + "*Minimum time between synchronisation attempts*" object (0-100:128.1.3.255) value -> (4)

4.2. In case of success the algorithm parameters are set:

4.2.1.the moment of the next clock value collection for "now" + "*Meter clock verification cycle*" object (0-100:128.1.7.255) value

4.2.2.moment of the next forced clock setting for "now" + "*Maximum time between synchronisation attempts*" object (0-100:128.1.3.255) value

Algorithm 1 – "use_shift_time"

1. When connecting the meter for the first time (after DCU restart):

1.1. current clock value is collected -> (2)

2. Collection of the clock value: command DLMS GET CLOCK/2 is sent. Depending on the result:

2.1. In the case of a DLMS meter error, the attempt is repeated to retrieve the clock value at "now" + "*Minimum time between synchronisation attempts*" object (0-100:128.1.3.255) value -> (2)

2.2. In case of success, the moment of the next clock value collection is set to "now" + MAX("*Meter clock verification cycle*" object (0-100:128.1.7.255) value, "*Minimum cycle for time shift*" object (0-100:128.1.10.255) value) and checking of the value is performed -> (3)

3. Checking the value of the clock collected from the meter. If at least one of the conditions is true:

3.1. the absolute value of the time difference between the meter and the concentrator ("last clock difference") is greater than "*Minimum time deviation*" object (0-100:128.1.5.255) value

3.2. or the status byte of the returned clock contains the CLOCK_STATUS_DOUBTFUL_VALUE flag

3.3. or the status byte of the returned clock contains the CLOCK_STATUS_INVALID_VALUE flag

then: setting the meter clock is performed immediately -> (4)

4. Meter clock setting: DLMS ACTION shift_time(diff) request is sent with a "diff" value equal of "last clock difference", but not greater than "*Maximum time shift*" object (0-100:128.1.9.255) value. Depending on the result:

4.1. In the case of a DLMS meter error, the attempt to set the clock is repeated at "now" + "*Minimum time between synchronisation attempts*" object (0-100:128.1.3.255) value -> (4)

4.2. In case of success:

4.2.1. setting “last clock difference” -= “diff”

4.2.2. if the absolute value of the time difference between the meter and the concentrator (“last clock difference”) is greater than “*Minimum time deviation*” object (0-100:128.1.5.255) value - the next time of checking/setting the meter clock is set to “*Minimum time between synchronisation attempts*” object (0-100:128.1.3.255) value

4.2.3. otherwise - the next time of checking the meter clock is set to MAX(“*Meter clock verification cycle*” object (0-100:128.1.7.255) value, “*Minimum cycle for time shift*” object (0-100:128.1.10.255) value)

Additional remarks:

- during the first connection of the meter to DCU (no meter in the database), the use of CLOCK SET is allowed
- due to problems with access selector support in some meters (empty data returned if access selector is in the future) - profile data are not downloaded until the time of the given meter is synchronized (“last clock difference” >= “*Minimum time deviation*” object (0-100:128.1.5.255) value)
- the current “last clock difference” value can be retrieved for each meter (attribute 0x86 of the “*Meter basic information*” object (0-100:64.0.0.255))

Meter access

The most important function of the concentrator is detection and registering new meters and maintaining a list of meters that are currently operating in its range – this is the role of “Meter list” object – the fundamental one to manage the meters’ data acquisition. Other objects are supporting additional features and extending the basic functionalities.

Object	COSEM class_ID	OBIS code
Meter list	40000	0-100:0.0.0.255
DLMS client_id for data collection	1	0-100:128.2.1.255
Meter firmware control	40054	0-100:0.131.0.255
Meter emergency control	40055	0-100:0.132.0.255
Meter ACL	1	0-100:0.133.0.255

Object Meter list

The most important function of the concentrator is detection and registering new meters and maintaining a list of meters that are currently operating in its range. The DCSAP protocol defines the way in which the concentrator shares this list with the acquisition system.

We assume that the concentrator has a large enough buffer for the list of currently connected meters and additionally is able to store a certain number of records of removed (inactive) meters on this list, sufficient for correct operation of the acquisition system. This list has no more than one entry concerning a given meter. If the meter is disabled from the system, the current record of a meter becomes appropriately marked. Each

record is related to the time of the last update and a global sequential number of a change, thanks to which the acquisition system may obtain information about any change in this list by inquiring about the newer records than those recently received. Such an additive algorithm of reading a list of meters is correct under the condition that a concentrator maintains monotonicity of the sequential number of a change.

If all available records of the list are already taken by old entries concerning the removed meters, all those newly registered should overwrite the oldest ones (with the lowest sequential numbers). Assuming that the number of available records is sufficiently large and the frequency of supplementation of information by the acquisition system is large as compared to the frequency of appearance of completely new meters, the mechanism is able to ensure total synchronization of the list on the side of the acquisition system.

Apart from the global sequential number of a change, time of the last update and a numerical identifier of a meter assigned by the concentrator, each record of the list contains a meter manufacturer's code, meter's name assigned by the manufacturer and a mark of presence of a meter in the system.

It should be emphasized that the changes in the object *Data concentrator meter list* must be carried out in a atomic manner. It is not acceptable to read a record concerning any meter, which has not been completed in full (in the case of a newly detected meter) or changed in full (in the case of a meter that is no longer visible or is visible again). The time of the last record change (or its adding) is updated along with the other fields of the structure *meter_list_entry*.

Security settings

Security object defines security mechanisms used to access the DCU and also the secure manner connections to the meters.

Object	COSEM class_ID	OBIS code
Permissions: webGUI	1	0-100:31.0.0.255
Permissions: remote access (no SSL)	1	0-100:31.0.1.255
Permissions: remote access over SSL - default	1	0-100:31.0.2.255
Permissions: remote access over SSL - group 3	1	0-100:31.0.3.255
Permissions: remote access over SSL - group 4	1	0-100:31.0.4.255
Permissions: remote access over SSL - group 5	1	0-100:31.0.5.255
Permissions: remote access over SSL - group 6	1	0-100:31.0.6.255

Permissions are defined as bit-string[256] where bits have the meaning:

Bit	Name	Description
0	ps_dcu_perm__invalid	invalid permission, always forbidden
1	ps_dcu_perm__conf_ip	configuring various DCU submodules

2	ps_dcu_perm__conf_ntp	
3	ps_dcu_perm__conf_ddns	
4	ps_dcu_perm__conf_services	
5	ps_dcu_perm__conf_events	
6	ps_dcu_perm__conf_perms	
7	ps_dcu_perm__conf_tzconf	configure timezone
8	ps_dcu_perm__set_clock	set clock
[…]		
10	ps_dcu_perm__conf_users	system security
11	ps_dcu_perm__conf_loginlimits	
12	ps_dcu_perm__conf_certnanny_config	
13	ps_dcu_perm__conf_ipsec_config	
14	ps_dcu_perm__conf_initconfig	
[…]		
16	ps_dcu_perm__restart	performing actions on DCU
17	ps_dcu_perm__fw_upgrade	
18	ps_dcu_perm__emergency_control	
[…]		
32	ps_dcu_perm__remote	all remote requests are forbidden
33	ps_dcu_perm__remote_mgmt	remote requests with management association are forbidden
34	ps_dcu_perm__remote_fw	remote requests with firmware update association are forbidden
[…]		
48	ps_dcu_perm__conf_display	configure display, if available
49	ps_dcu_perm__conf_dcuprof	configure predefined/disabled profiles list

Table 39 Meaning of bits in permissions definition

Other abstract objects

This group of objects define other aspects of the DCU behaviour, collect statistics necessary to monitor or debug the data concentrator diagnostic information, store events history etc.

Special role has “DCU firmware” object – controlling and performing firmware upgrade procedure of devices.

Object	class_ID COSEM	Kod OBIS
DCU firmware version	1	0-100:128.0.1.255
DCU passport	1	0-100:128.0.2.255
DCU status flags	3	0-100:0.99.0.255
DCU application statistics	3	0-100:0.100.0.255
Meters cumulative statistics	40057	0-100:0.102.0.255
DCU application events	40056	0-100:0.103.0.255
DCU firmware	40101	0-100:0.0.1.255
DCU run information	40103	0-100:0.0.2.255
DCU event list	40001	0-100:0.0.3.255
DCU NTP server list	40100	0-100:0.0.4.255
DCU network statistics	40002	0-100:0.0.5.255
DCU event list - important events	40001	0-100:0.1.3.255
DCU initconfig	40051	0-100:0.128.0.255
DCU modem parameters	40052	0-100:0.129.0.255
DCU debug parameters	1	0-100:0.130.0.255
DCU profile config cache	1	0-100:0.134.0.255
DCU predefined profiles	1	0-100:0.134.1.255
DCU disabled profiles	1	0-100:0.134.2.255
DCU system services	40100	0-100:170.0.1.255
DCU IP configuration (WAN)	40100	0-100:170.0.2.255
DCU DDNS configuration	40100	0-100:170.0.3.255

DCU users configuration	40100	0-100:170.0.4.255
DCU login limits configuration	40100	0-100:170.0.5.255
DCU certnanny configuration	40100	0-100:170.0.6.255
DCU IPSEC configuration	40100	0-100:170.0.7.255
DCU IP configuration (LAN)	40100	0-100:170.0.8.255
DCU meter types	1	0-100:180.0.1.255

Object DCU firmware

All concentrators must enable software update. For this reason, the DCSAP protocol provides the need of implementation of a dedicated object in the concentrator that will allow ordering software update by the acquisition system. We assume that the updating method passes an URL address indicating the binary image of the update. Responsibility for verification of correctness of the updating code, possible detection of compliance of software and equipment versions belongs to the concentrator.

Updating software of the concentrator initiated by method *start_update()* results in checking whether there are no other updates currently running or also whether the concentrator restarting was not initiated previously and if it is the case, the *temporary-failure* error is returned. Otherwise, the concentrator increases the sequential number and transfer it in a response, along with the code *success*.

If the concentrator does not support updating with maintaining a session, and the updated software module requires cessation of normal operation of the concentrator and its restart, all session are closed for the duration of the update. If the updated module does not require cessation of the device's normal operation, or maintaining a session is supported, then the attributes *last_update_status* and *last_update_progress* should be updated on the current basis in the course of the duration of the update.

The acquisition system after re-establishing connection with a concentrator or after obtaining a respective notification, if the session was not lost, will be able to read the attributes *last_update_id* and *last_update_status* to check if the update has been successful.

Object DCU run information

The concentrator must make available basic information on the current and previous start-up and the number of all start-ups. This is diagnostic information that with periodical checking, allow detection of problems with too frequent restarting of devices or their improper functioning. In addition, this object allows to restart the concentrator.

Object DCU event list

A concentrator, like meters, is a source of events, which should be stored in it with a possibility of future readout. During their logging, a notification should be generated to all sessions with an activated notification of events.

A concentrator logs the following events: concentrator start-up, order of its restart, order of software update, completion of software updating and change in a meter's presence (e.g. connection to a new, disconnection of an old as well as reconnection of a previously disconnected). Also supported is logging the events generated by the client calling method *push()* of this object, to which a description of the event consistent with structure

event_list_entry is transferred. In order to prevent creating untrue events, field *reason* of this structure is changed to value *EV_PUSH* in the case of the events generated in such a way.

If all available records of the list are already taken, new events should overwrite the oldest on the list (with the lowest sequential numbers). Every new event gets a new (incremented) sequential number, that the events log is in fact a cyclic buffer.

Object DCU NTP server list

Concentrators must ensure setting a list of NTP server addresses, so that it is possible to use the same time servers in the whole infrastructure. This object allows defining these servers in the form of a table of character strings representing domain names or IP addresses.

There is additional method available - `sync_now()` which triggers immediate retry to synchronize time with NTP servers.

Object DCU network statistics

A concentrator must collect basic network statistics from the communication realised by DCSAP protocol. These statistics are not persistent and are collected from scratch after each start-up of a concentrator.

The following values are subject to tracking: number of all previously open sessions, number of currently maintained sessions, number of bytes received in the application layer (from all sessions in total), number of bytes sent (from all sessions in total), number of messages received (in all sessions in total), number of messages sent (in all sessions in total), number of executed orders directed to the objects implemented by a concentrator (in all sessions in total) and the number of executed orders directed to the objects implemented by meters (in all sessions in total).

Object DCU status flags

DCU status flags, double-long-unsigned with bit values:

Bit	Name	Description
0	disk_failure	DCU underlying storage failure
1	db_failure	Database integrity error
2	plc_modem_failure	PLC modem unresponsive / failed
3	imeter_failure	Internal meter unresponsive / failed
4	-	<not used>
5	-	<not used>
6	-	<not used>
7	-	<not used>
8	memory_failure	DCU memory allocation failure
9	internal_error	DCU internal logic failure

10	time_invalid	System time is not synchronized
11	dcsap_invalid	Invalid DCSAP message received
12	-	<not used>
13	-	<not used>
14	-	<not used>
15	-	<not used>
16	dlms_unhandled_msg	unknown/inconsistient DLMS message encountered
17	dlms_error	DLMS parsing/flow error

Table 40 Description of the DCU status flags

Session objects of a concentrator

All session objects of a concentrator are temporary, established along with establishment of each new session and removed after its completion. Session objects in various sessions are independent from one another.

DCU session

Object	COSEM class_ID	OBIS code
Session caching	1	0-100:32.0.0.255
Session notifications	1	0-100:32.0.1.255
Session origin	1	0-100:32.1.0.255
Session common name	1	0-100:32.1.1.255
Session permission group	1	0-100:32.1.2.255
Session client id	1	0-100:63.0.0.255
Session remote timeout	1	0-100:63.0.1.255
Session requests sequential	1	0-100:63.0.2.255

Object Session caching

One of the basic functions of a concentrator is acceleration of access to the profiles of consumption of electric energy latched in meters. It is implemented by automatic background collection of subsequent, latched in meters, values and their storage in a concentrator. At the command of meter's profiles objects (class_id=7)

readout, a concentrator immediately returns data from its own buffer, not communicating with a meter. It accelerates execution of this type of commands and additionally reduces the load of communication medium between a concentrator and a meter because the data which may be collected from a concentrator repeatedly, are transmitted with this medium only once. This also permits effective use of connection between a concentrator and meters during the absence of communication on the side of the acquisition system.

The mechanisms of acceleration are a specific solution depending on the concentrator's manufacturer. In the simplest perspective, a concentrator can only mediate in collecting all data, also profile-related, from electricity meters. In most cases, efficiency of the communication channel between a concentrator and meters will not allow such trivial implementation. In order to meet the requirements of the system's user, it is then necessary to consider mechanisms of acceleration, at least in connection with profile data collection. For this purpose, the concentrator's manufacturer should introduce objects intended for control of such a mechanism. Setting these objects may indicate elements of the profile that a concentrator will collect from meters with the purpose of buffering. It is also possible to start-up buffering after recognition of the first request concerning this type of data by the concentrator. In such a case a good solution is disabling buffering after some time, when the acquisition system does not collect anymore a given profile element. In the latter case, the time of expiry of buffering should be configurable by means of a dedicated object.

In some cases, for example for diagnostic purposes, there is a need for direct access to meters. At that time, there must exist a mechanism deactivating buffering of responses of meters. For this purpose, a concentrator must implement an object, whose appropriate setting forces direct access to meters within a given session.

Attribute *value* of this object is of *boolean* type and by default it assumes value *true*.

Object Session notifications

The asynchronous mechanism of notifications allows increase in efficiency of communication between the acquisition system and a concentrator. It consists in asynchronous notification of any occurring events and change in the state of a concentrator or meters. Asynchronous notifications are based on a dedicated DLMS message – more details about it can be found in source documents.

The mechanism of asynchronous notifications cannot, however, be used if the acquisition system is forced to fully control data flow on the communication medium connecting it with a concentrator. For this reason, this mechanism is by default disabled after establishing a session and if the acquisition system wants to use it, it must accordingly control a specific object of a concentrator. Setting this mechanism applies only to a given session and does not affect other parallel or those succeeding it.

By definition, the acquisition system uses a single communication session with a concentrator, but nothing is in the way of handling asynchronous notifications for example in a dedicated session.

Attribute *value* of this object is of *boolean* type and by default it assumes value *false*.

Global objects of meters realised by a concentrator

Global objects of meters performed by the concentrator have been defined additionally. They are performed by the software of the data concentrator, but they exist as independent instances for each registered meter. They are available by the DCSAP protocol (meter identified by *device_id*) and using the standard specification of a COSEM method or attribute descriptor in a relevant type of DLMS message.

All global objects of meters realised by a concentrator are persistent, i.e. concentrator restart does not cause loss of their contents.

It is forbidden to ask for meter objects realised by a concentrator and meter itself in a single *-with-list* query.

Meters

Object	COSEM class_ID	OBIS code
Meter profile special cpobj	40060	0-100:0.255.255.255
Meter basic information	40102	0-100:64.0.0.255
Meter firmware	40101	0-100:64.0.1.255
Meter profile configuration	40160	0-100:160.0.1.255
Meter DLMS Security Setup - Reading association	40199	0-100:65.0.2.255
Meter DLMS Security Setup - Management association	40199	0-100:65.0.3.255
Meter DLMS Security Setup - Firmware Update association	40199	0-100:65.0.4.255
Meter DLMS Security Setup - Preestablished association	40199	0-100:65.0.6.255
Meter TX frame counter - Reading association - global unicast	1	0-100:66.0.2.255
Meter RX frame counter - Reading association - global unicast	1	0-100:66.128.2.255
Meter TX frame counter - Management association - global unicast	1	0-100:66.0.3.255
Meter RX frame counter - Management association - global unicast	1	0-100:66.128.3.255
Meter TX frame counter - Firmware Update association - global unicast	1	0-100:66.0.4.255
Meter RX frame counter - Firmware Update association - global unicast	1	0-100:66.128.4.255
Meter frame counter - Firmware Update association - global broadcast	1	0-100:66.1.4.255
Meter frame counter - Preestablished association - global broadcast	1	0-100:66.1.6.255

Object Meter basic information

An object storing an identifier of the meter's full configuration and a field containing the information concerning a type of a meter and its properties, the so-called passport. The structure of a passport is defined by each manufacturer independently. The following information must form compulsory elements of this structure:

- meter model,
- meter software version.

Object Meter firmware

All meters must enable software update. Concentrators must realise the objects addressed with the identifiers of registered meters, which will allow ordering software update by the acquisition system. We assume that the updating method provides an URL address indicating a binary image of the update. Responsibility for verification of correctness of the updating code, possible detection of compliance of software and equipment versions belongs to the concentrator and meters.

Software of the devices often consists of many modules, independently versioned. Any possible support of partial updates (i.e. covering only selected modules) also belongs to the concentrator and meters, i.e. the used format of an image should contain the information on the modules that are planned in it for the update. It is then recommended to make the *version* attribute a combination of version numbers of all modules separated with a selected separator, e.g. a semicolon.

If the acquisition system orders many subsequent commands of software update addressed to different meters, but using the same URL address, they can be handled with a software image collected and buffered once. This should slightly speed up mass updates of meters. The concentrator can execute such aggregated upgrade command by sending the firmware to the meters in broadcast mode.

7. Use of DCSAP protocol

This chapter presents the most typical examples of use DCSAP protocol in the process of realising acquisition.

Starting DCSAP session with a concentrator

The acquisition system during its work opens DCSAP sessions with all concentrators. Starting a session consists of establishing TCP connection with a concentrator. After establishing a connection, the acquisition system collects a list of meters and reads their configuration. If it is a subsequent session for a given concentrator, the acquisition system may supplement incrementally a list of meters only on these positions where a change occurred. If the acquisition system wants to use the notification mechanism during a given session, it should enable this mechanism immediately after establishing connection.

If commands are not sent to a concentrator for 5 minutes, the acquisition system should send an empty message and wait for a response. In this way, any possible connection loss will be detected, caused by unplanned hardware restart of a concentrator or, in the case of long-term lack of a response, suspension of a connection caused by problems of the telecommunication network. No response to an empty message for 5 minutes causes to closing a connection. After closing or breaking a connection, the acquisition system attempts to establish new one, in 3-minute intervals.

A concentrator should detect a long time of idleness of the session (lack of messages from the acquisition system for 10 minutes or longer) and close it releasing resources.

All of the above are shown in details in the folowing diagram sequences.

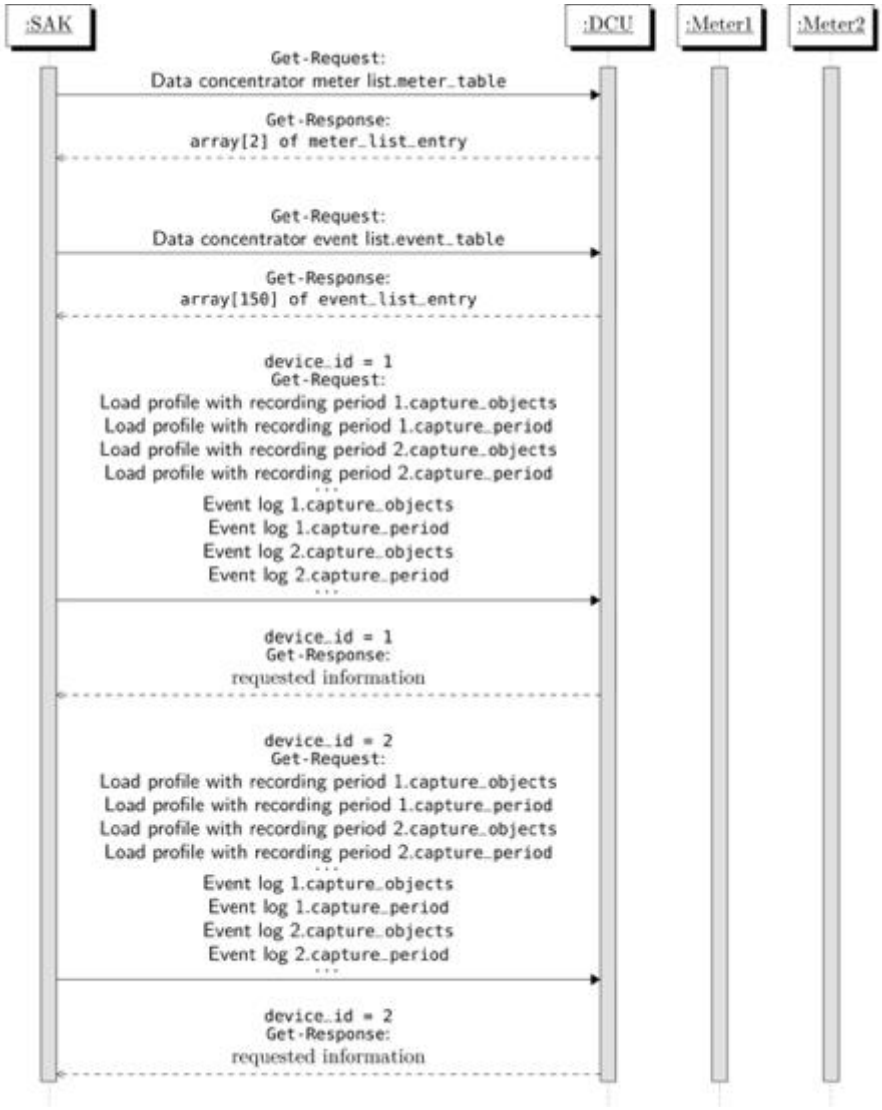

Figure 3 Sequence diagram of operations performed by the acquisition system after the start of the first DCSAP session with a concentrator

It has been assumed that a concentrator supports acceleration of access to configuration of meters and that the acquisition system does not use the notification mechanism.

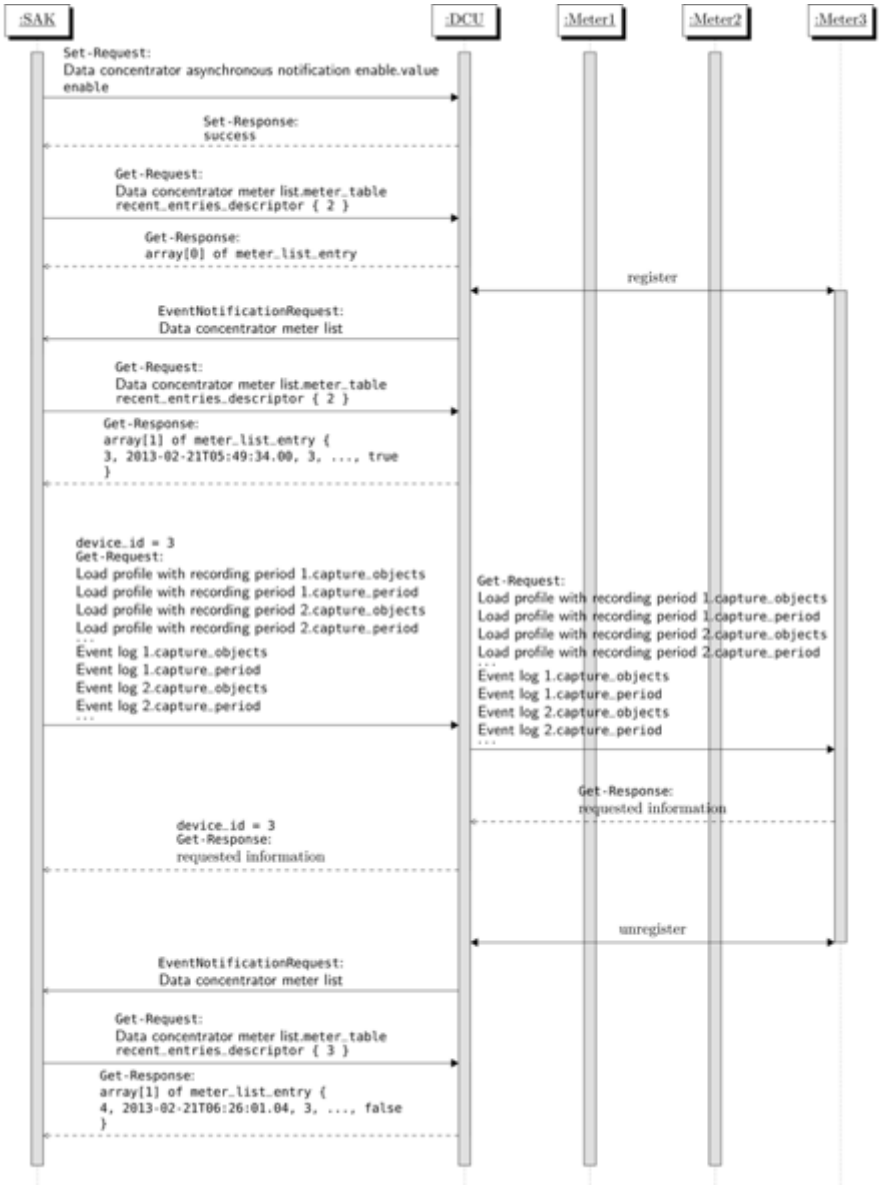

Figure 4 Sequence diagram of operations performed by the acquisition system after the start of DCSAP session with a concentrator

It has been assumed that a concentrator communicates with meters in the application layer by means of the DLMS protocol.

Figure 5 Sequence diagram of maintaining connection with a concentrator

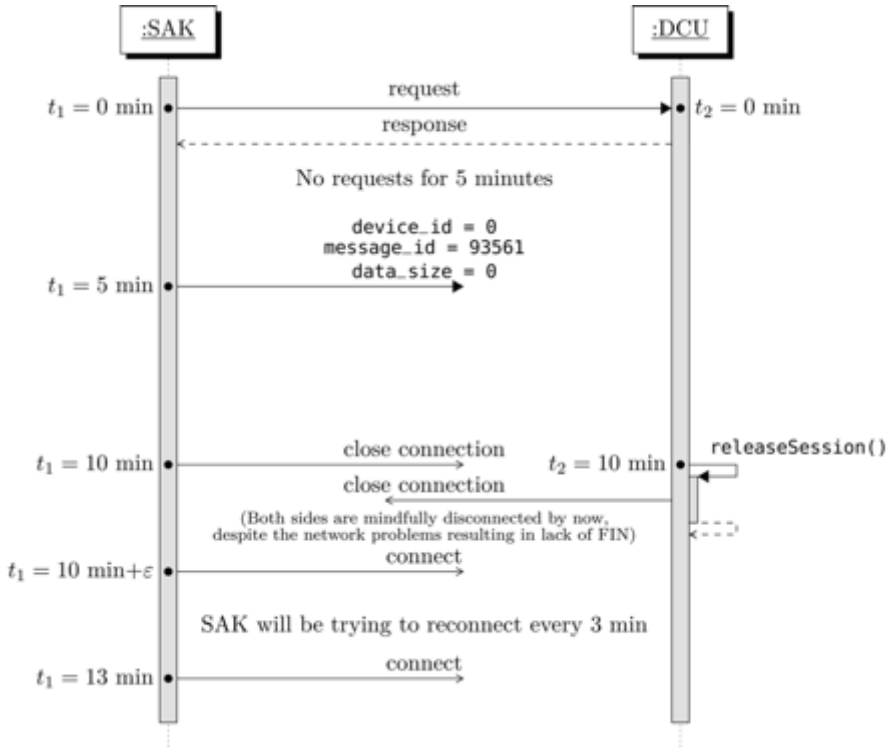

Figure 6 Sequence diagram of an attempt of maintaining connection with a concentrator and of idleness of the session in the case of problems with connection

It has been assumed that during the absence of communication, the parameters of communication have been deteriorated to the level preventing communication.

Closing DCSAP session

Closing a session follows closing the TCP connection.

Topology synchronization

During a session, the acquisition system synchronizes topology by sending a command to download a list of meters. In order to minimize the transmitted data, the acquisition system applies a selective choice stating the highest known sequential number of a record change. As a result, a concentrator will reply only with such a sub-set of records that have changed (i.e. the records with a sequential number of the last change greater than the set one).

There are two methods of topology updating activation. The acquisition system may periodically check whether there are updated records or respond to asynchronous notification.

Meter register readout

After establishing a session with a concentrator and downloading a list of meters, the acquisition system may direct commands to the meters connected to a particular concentrator. It prepares a message of the command for this purpose. It sets a meter identifier read from the list, a unique identifier of the message and completes the DLMS data with command *Get-Request* indicating a specific register of the unit. A message prepared in such a manner is sent to a concentrator and then it awaits an answer. Below relevant sequence diagrams are shown to explain the concept further.

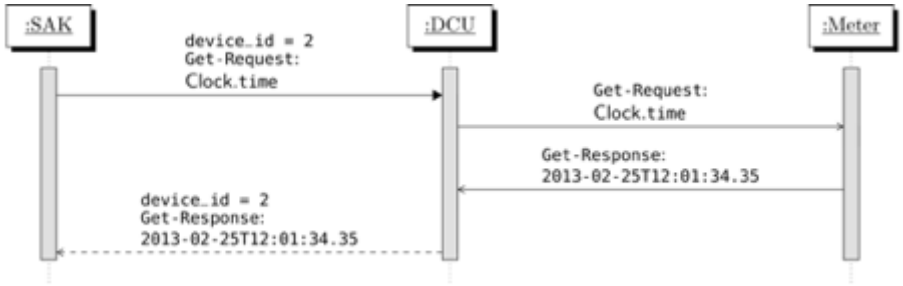

Figure 7 Sequence diagram of readout from the meter – successful option

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

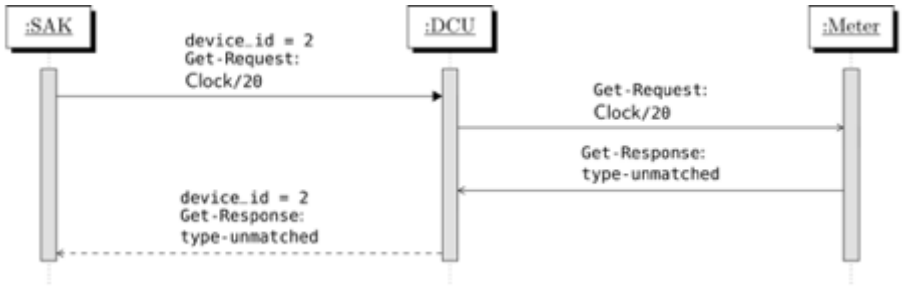

Figure 8 Sequence diagram of readout from the meter – option of incorrect attribute

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

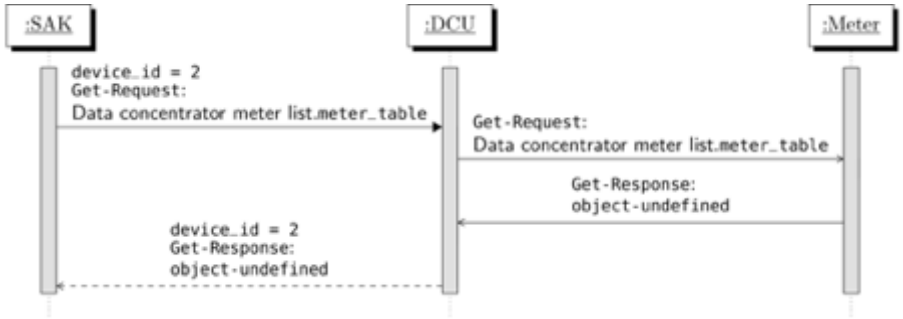

Figure 9 Sequence diagram of readout from the meter – option of a non-existing object

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

Figure 10 Sequence diagram of readout from the meter – option of a unknown identifier of a meter

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

Meter register writing

Writing is conducted similarly to readout. A message is sent with command *Set-Request* in the DLMS data. The sent response will confirm execution or non-execution of the operation.

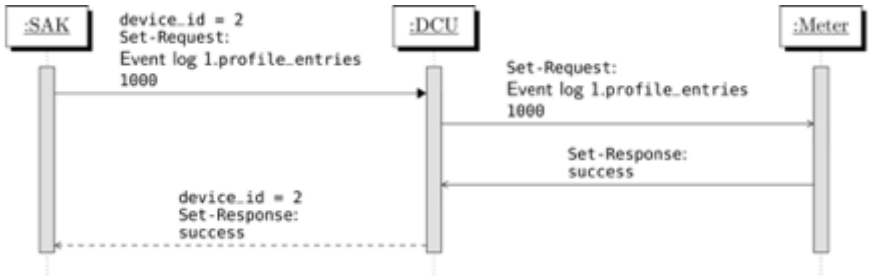

Figure 11 Sequence diagram of writing to a meter – successful option

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

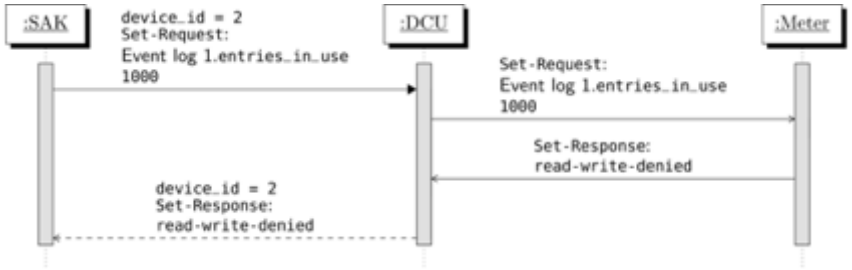

Figure 12 Sequence diagram of writing to a meter – unsuccessful option

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

Meter configuration

The meter configuration consists in sending a set of commands for writing a complete set of registers of a given unit. In the case of an incomplete confirmation or selective failure, the whole set of an operation can be repeated or the operations can be excluded from this set, for which confirmation have been received, for the purpose of optimization.

Downloading meter configuration parameters

Downloading configuration parameters consists in calling a sequence of readouts of subsequent registers constituting configurations elements of the unit.

Direct communication with the meter

Despite application of various techniques of buffering responses from the meters in concentrators, it is possible to force (e.g. for diagnostic purposes) direct communication with a meter, skipping these mechanisms. For this purpose, use command *Set-Request* to switch attribute *value* of object *Data concentrator meter data caching enable* to *false*, thus switching off buffering of meter responses. The responses to commands addressed to specific meters will come now from meters. Switching off of buffering is valid only under this session, until its completion or re-switching off of buffering.

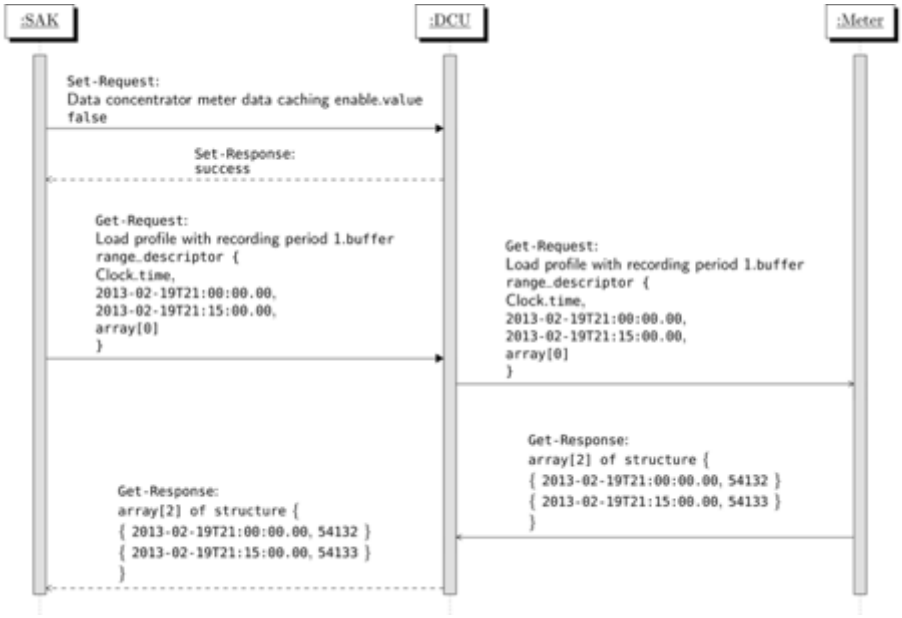

Figure 13 Sequence diagram of writing to the concentrator and direct readout from the meter

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol, and attribute capture_objects of object Load profiles with recording period 1 contains only time and value A+.

Concentrator restart

The concentrator's restart procedure is initiated by calling method *restart()* of object *Data concentrator run information*. This command returns an error if a concentrator or a meter are currently undergoing updating. Otherwise, the success code is returned, and a concentrator releases all sessions and restarts.

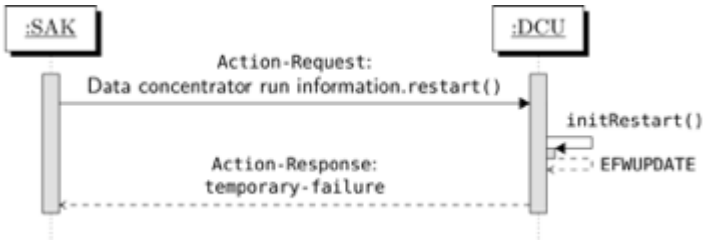

Figure 14 Sequence diagram of restarting a concentrator – unsuccessful option

It has been assumed that a software update was ordered, which is still not completed, before the restart of a concentrator.

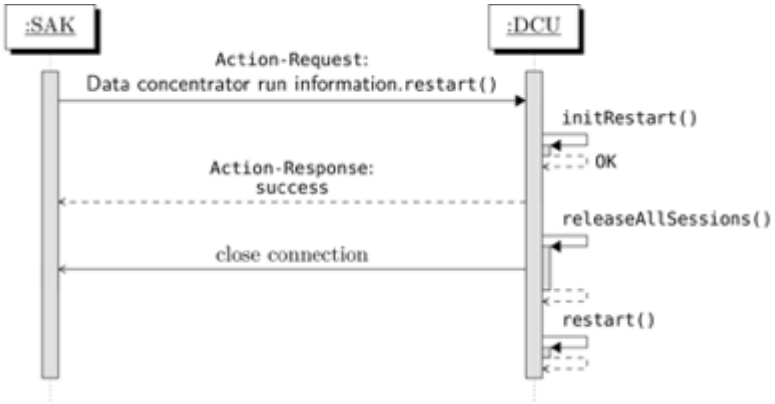

Figure 15 Sequence diagram of restarting a concentrator – successful option

Concentrator software updating

In order to update software on a concentrator, the acquisition system sends a message with a command for execution of method *start_update()* of object *Data concentrator firmware*, providing an URL address from which a concentrator downloads an update image with the HTTPS protocol. After downloading the image, a concentrator checks its correctness. Following this, it starts updating and afterwards – restarts. The acquisition system, after receiving a message about a successful initiation of updating, detects closing of a connection and then opens a new session, from the level of which it checks the date of the last update and its status. The entire process and possible failures are illustrated by sequence diagrams presented below.

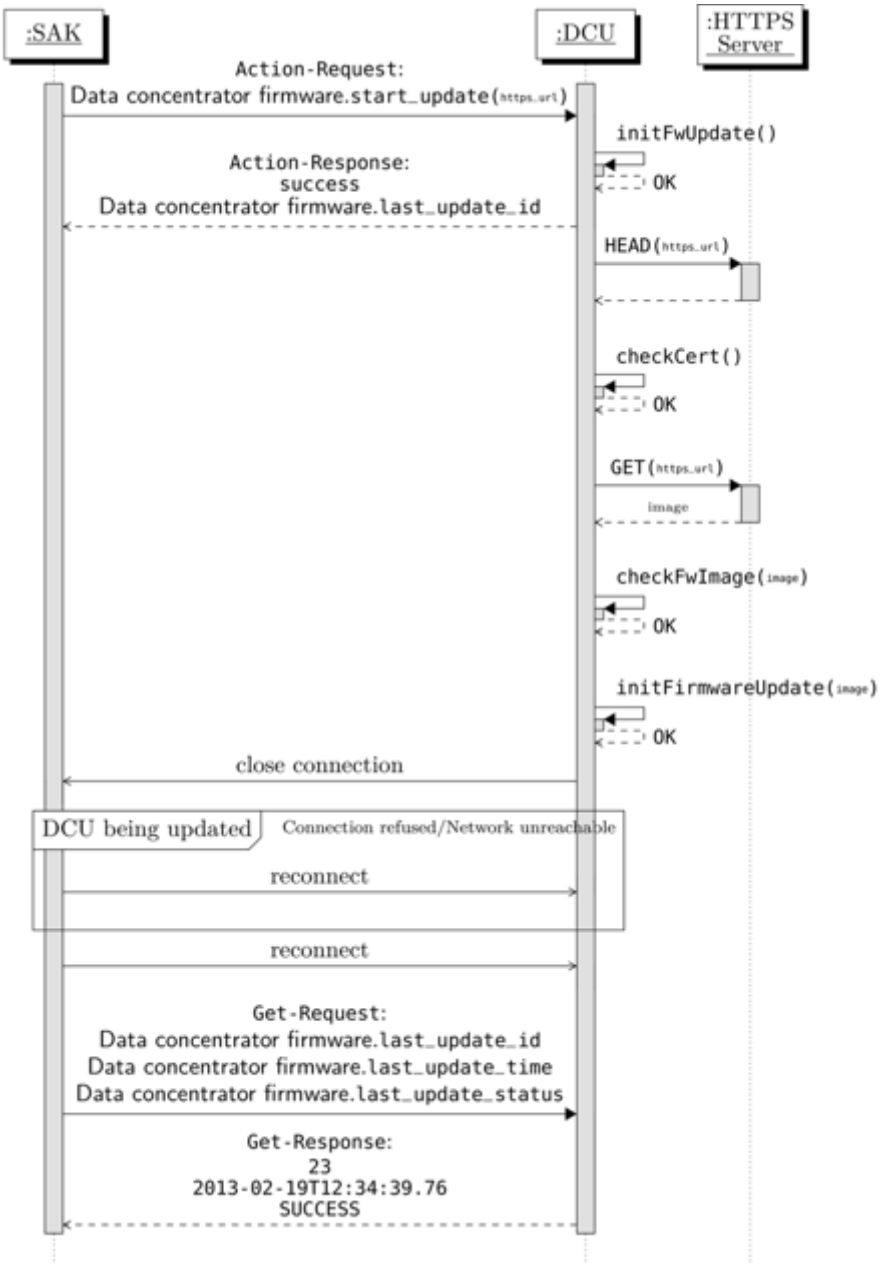

Figure 16 Sequence diagram of concentrator software update – successful option

It has been assumed that a concentrator does not support sustaining a session in the course of updating and that all steps were successful.

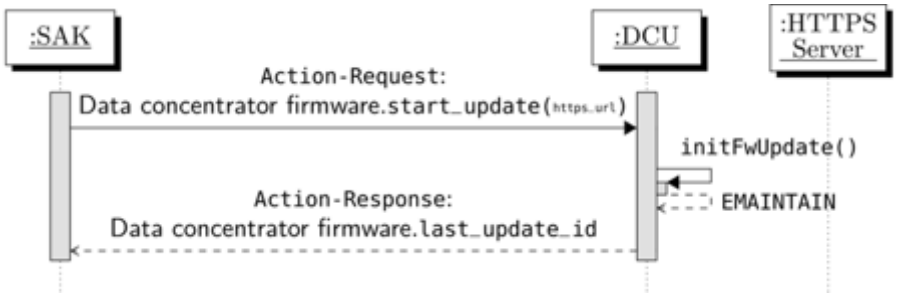

Figure 17 Sequence diagram of concentrator software update – 1, unsuccessful option

It has been assumed that another updating was started earlier, which is still not completed.

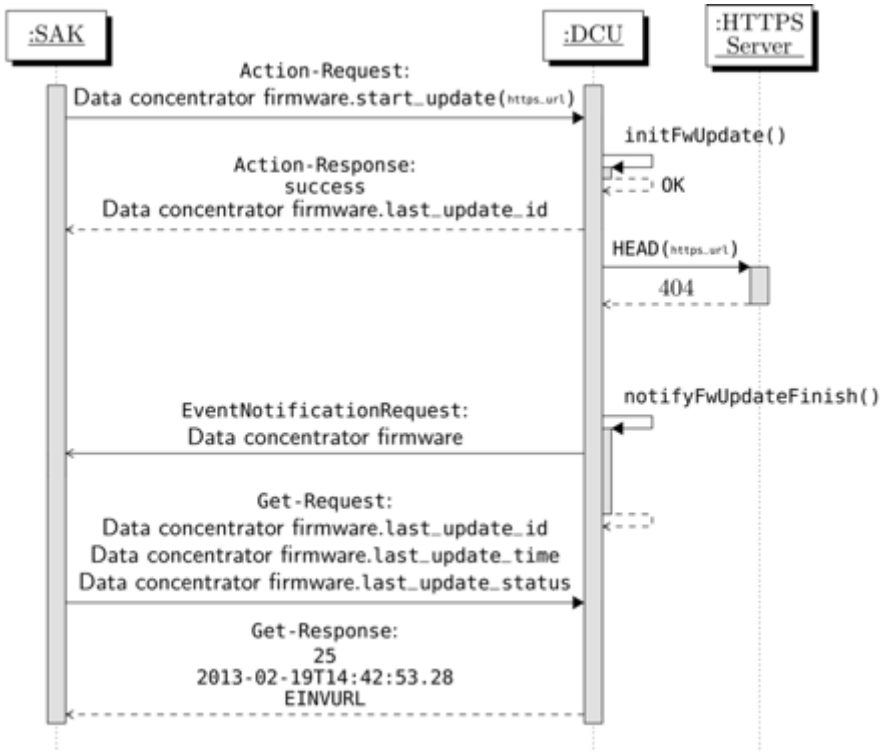

Figure 18 Sequence diagram of concentrator software update – 2, unsuccessful option

It has been assumed that the address of an image is incorrect.

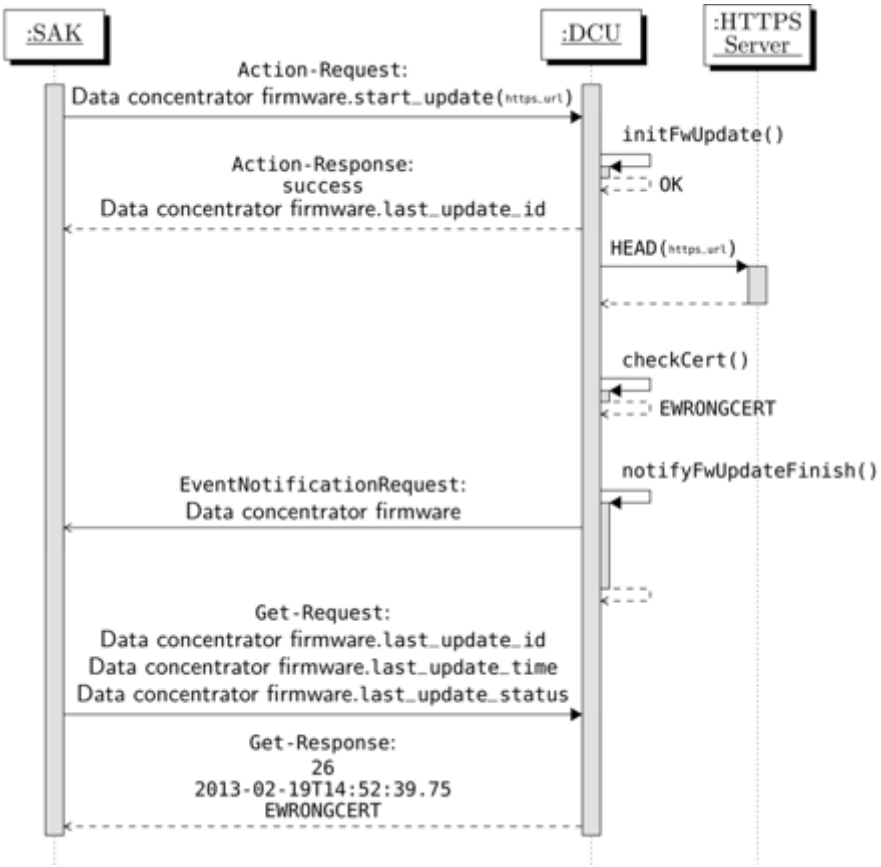

Figure 19 Sequence diagram of concentrator software update – 3, unsuccessful option

It has been assumed that a certificate that the HTTPS server returns, is incorrect.

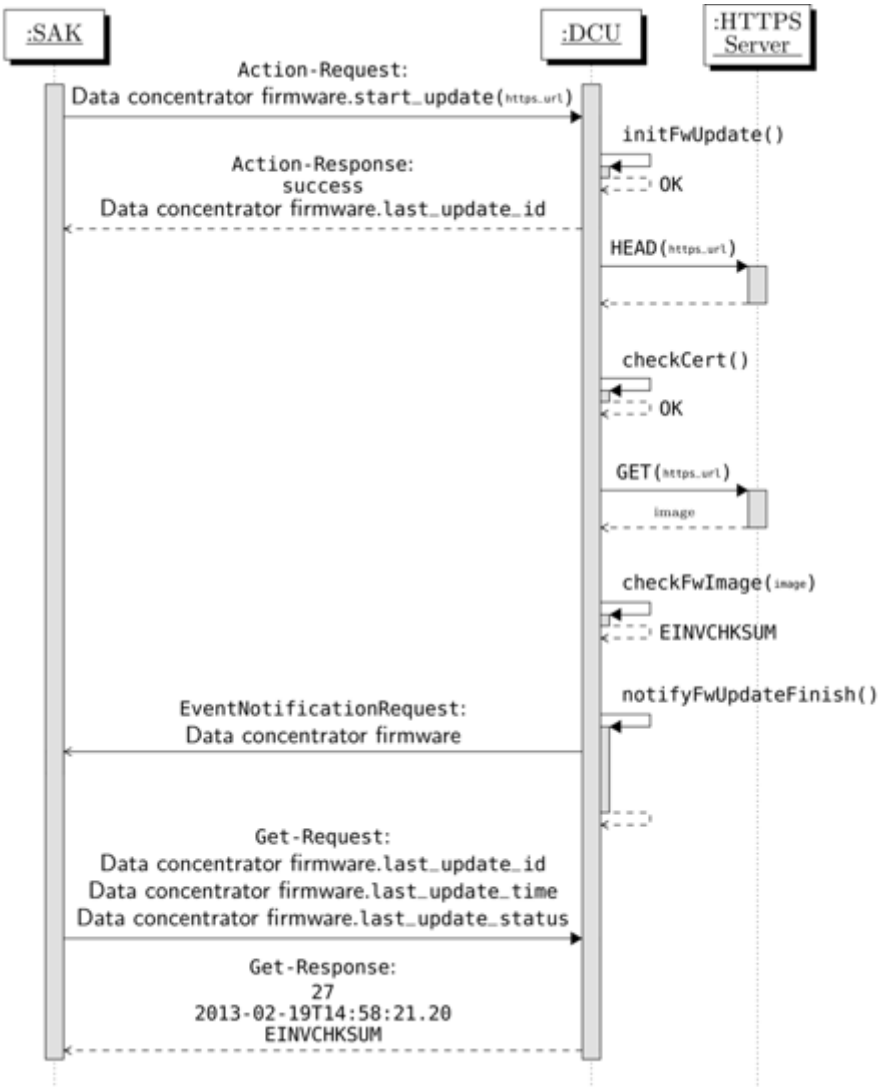

Figure 20 Sequence diagram of concentrator software update – 4, unsuccessful option

It has been assumed that an image file collected from the HTTPS server is damaged (its checksum is improper).

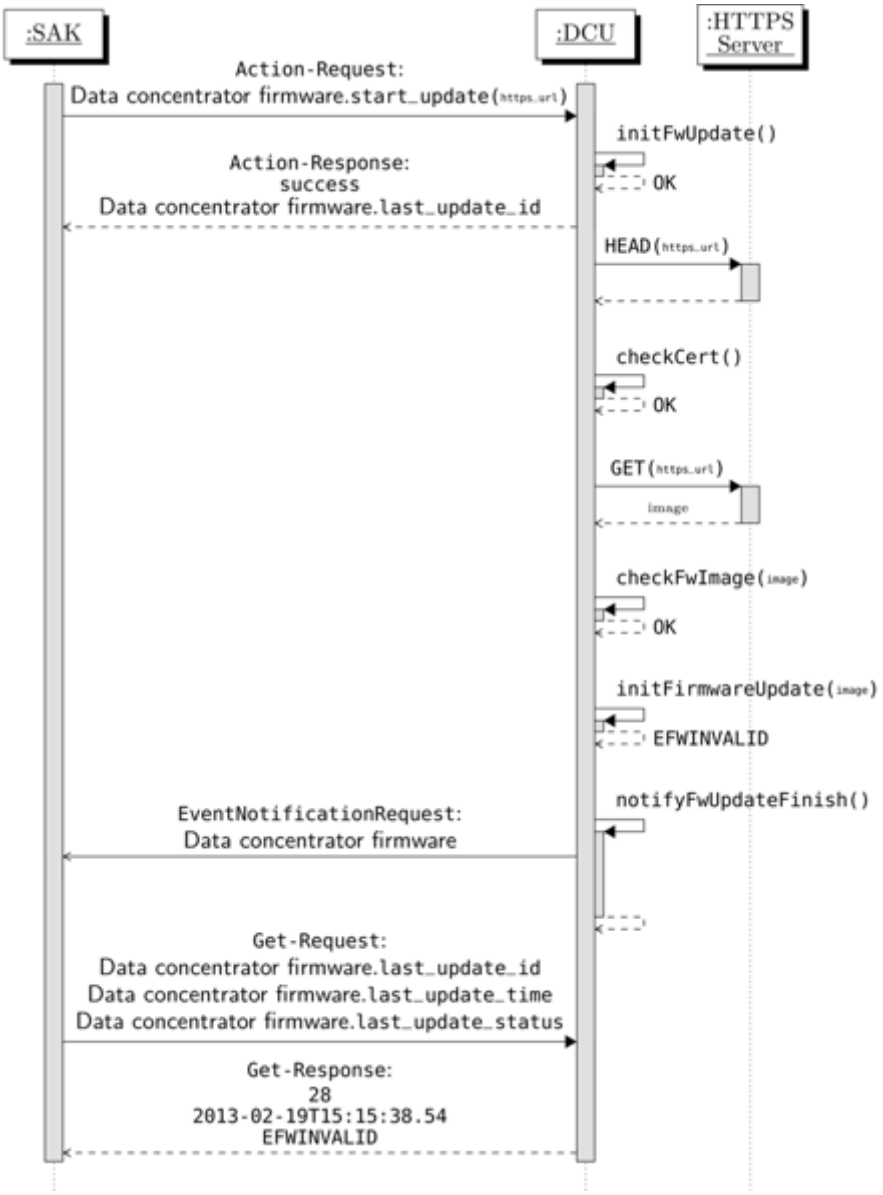

Figure 21 Sequence diagram of concentrator software update – 5, unsuccessful option

It has been assumed that an indicated image is not serviced by the concentrator.

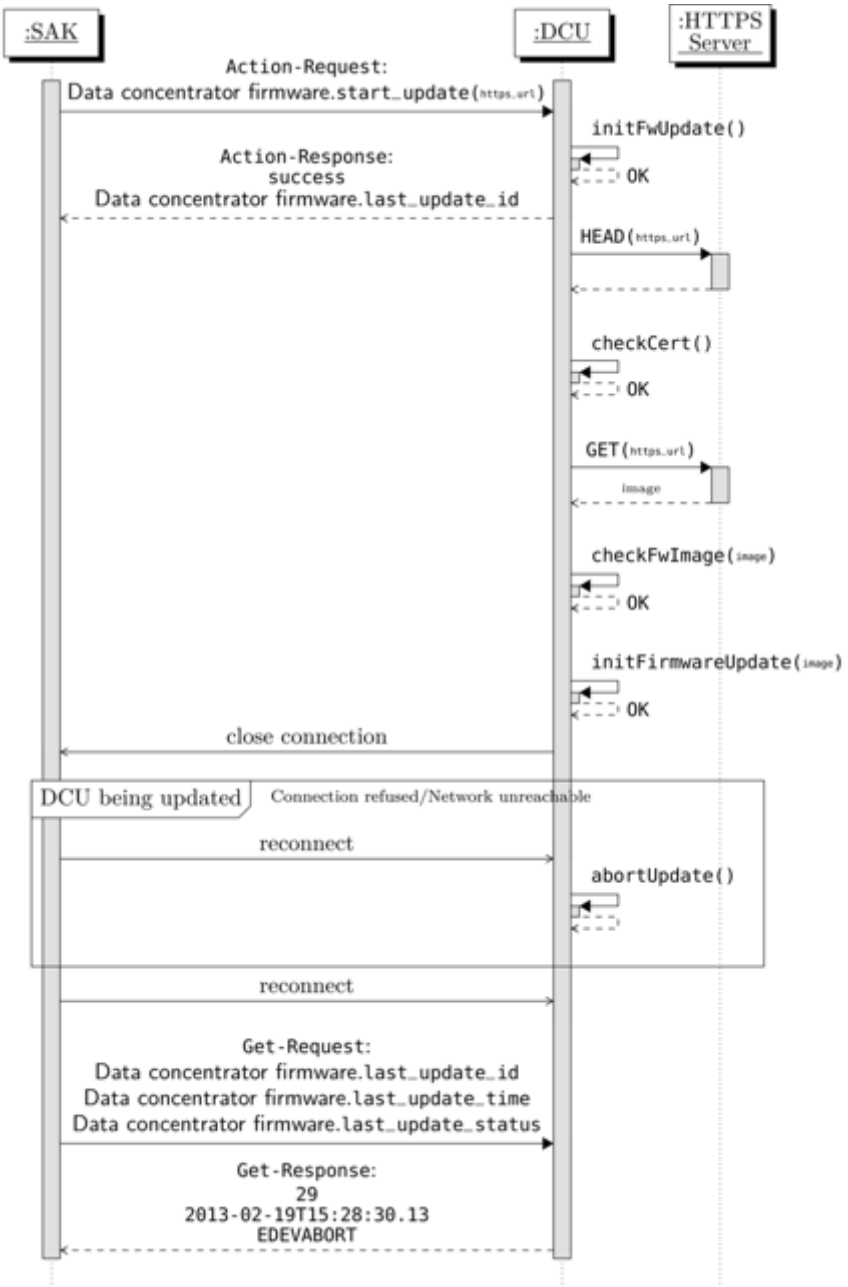

Figure 22 Sequence diagram of concentrator software update – 6, unsuccessful option

It has been assumed that a concentrator does not support sustaining a session in the course of updating and that it interrupted updating.

Updating meter software

Updating meter software takes place likewise to software update of a concentrator, with the exception that this command of updating is addressed to meter and object *Meter firmware*. An update image must in such a case match the indicated meter. The task of a concentrator is responding to the acquisition system with success

and a new (already incremented) identifier of the last (namely the currently lasting) updating, if no other updating is ongoing. Following this, downloading of this image proceeds, followed by verification of its correctness (if the manufacturer of the concentrator is able to carry it out), transfer of an image to a meter and relevant release of an updating process. At the end, a notification is sent to the acquisition system informing about termination of updating, after receipt of which the acquisition system checks the status of the last update.

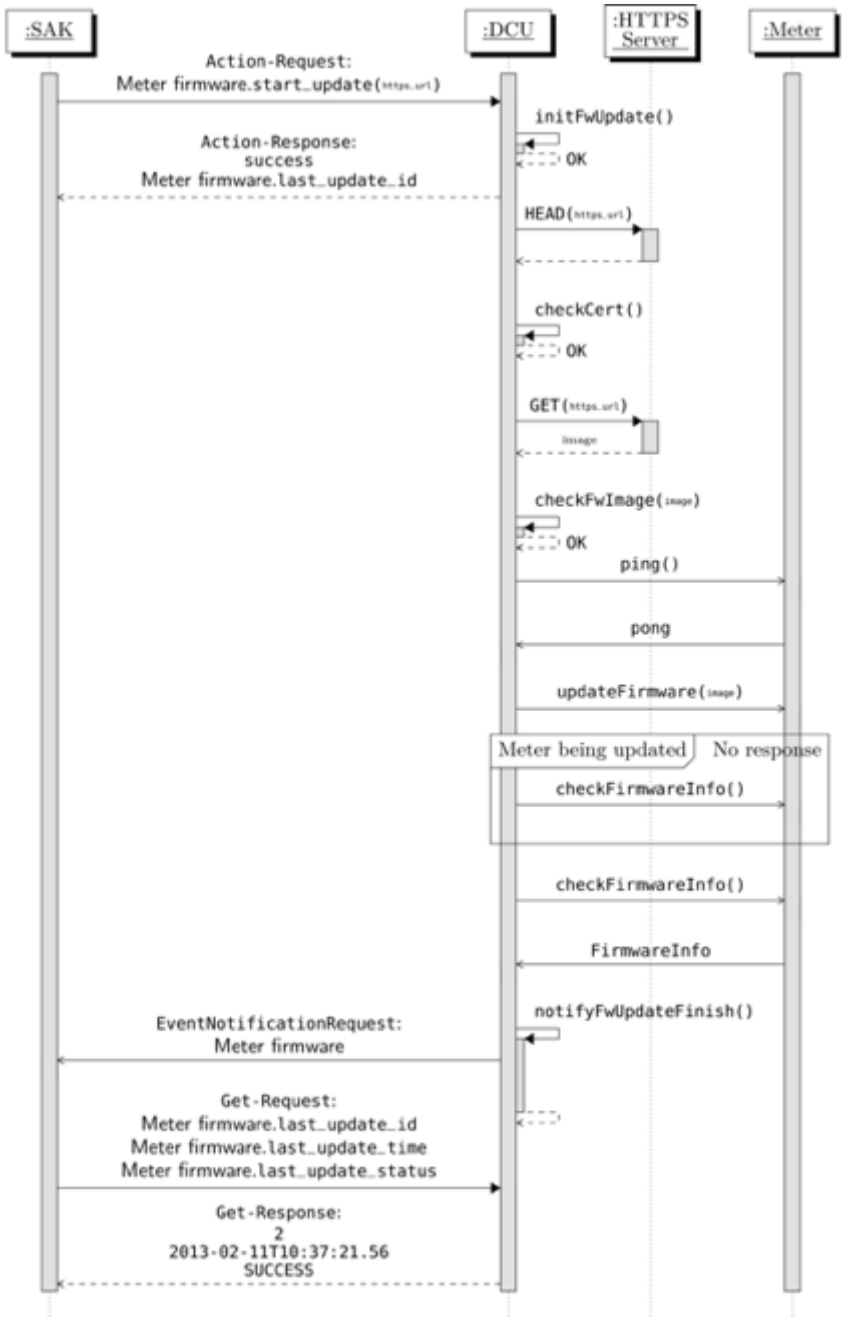

Figure 23 Sequence diagram of meter software update – successful option

It has been assumed that all steps were successful.

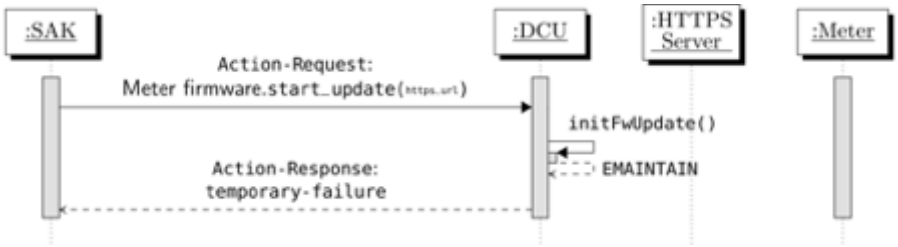

Figure 24 Sequence diagram of meter software update – 1, unsuccessful option

It has been assumed that another updating was started earlier, which is still not completed.

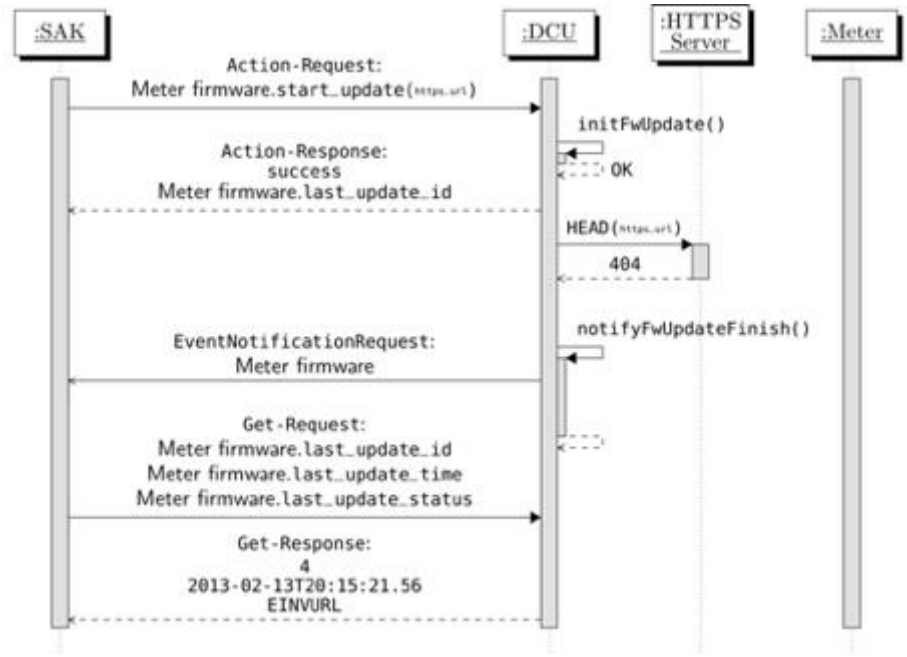

Figure 25 Sequence diagram of meter software update – 2, unsuccessful option

It has been assumed that the address of an image is incorrect.

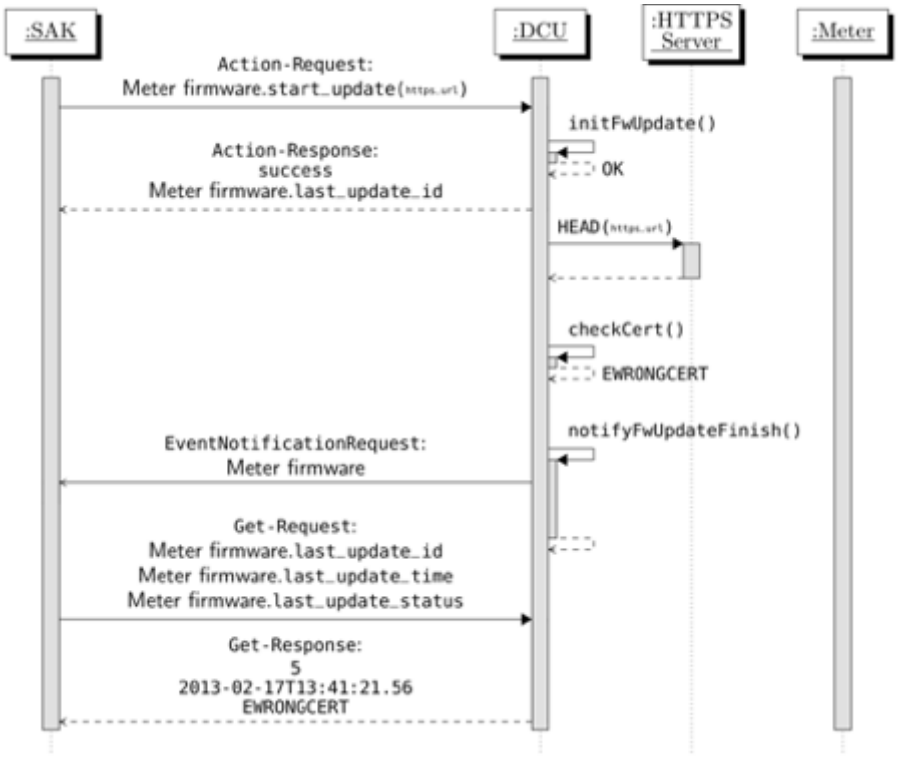

Figure 26 Sequence diagram of meter software update – 3, unsuccessful option

It has been assumed that a certificate that the HTTPS server returns, is incorrect.

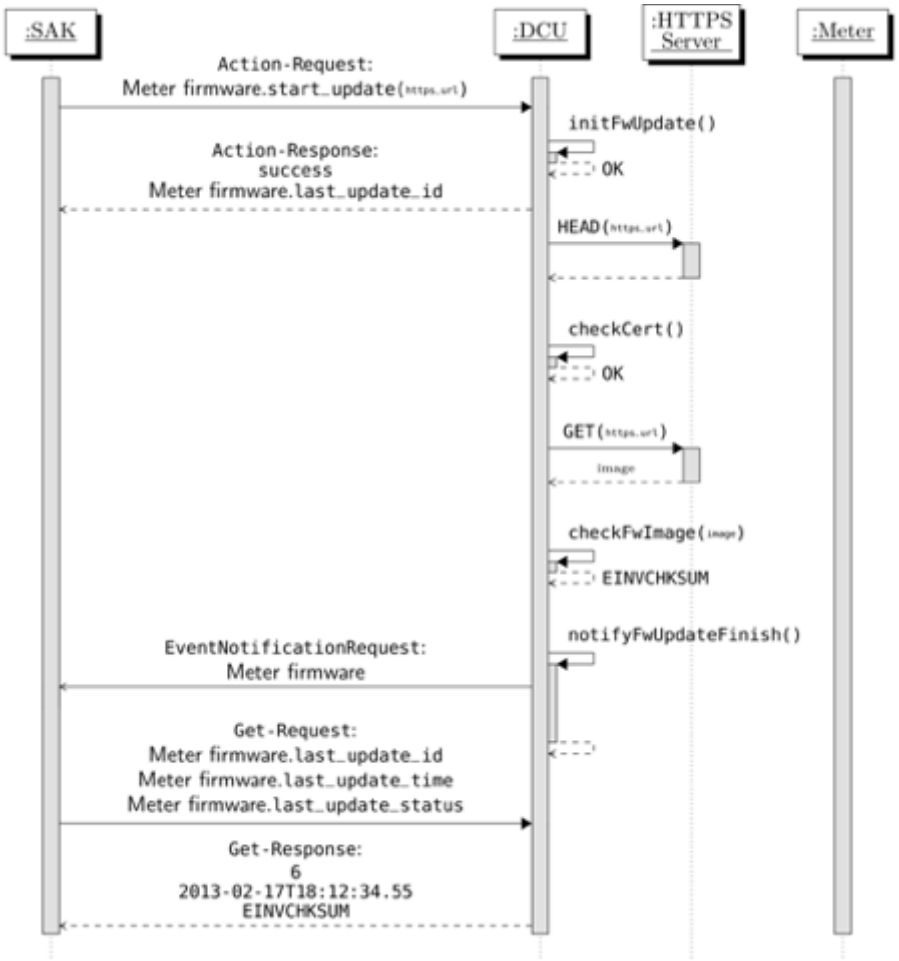

Figure 27 Sequence diagram of meter software update – 4, unsuccessful option

It has been assumed that an image file collected from the HTTPS server is damaged (its checksum is improper).

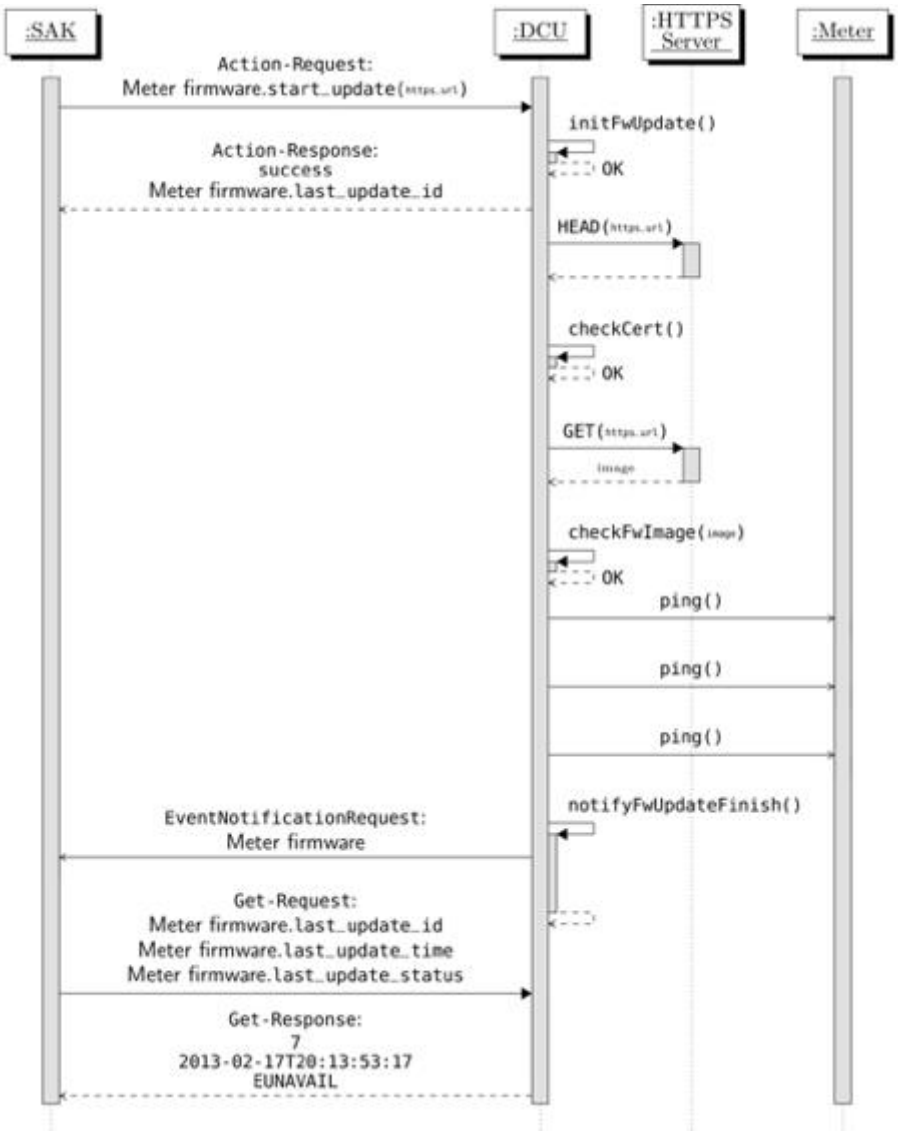

Figure 28 Sequence diagram of meter software update – 5, unsuccessful option

It has been assumed that a meter is not available when trying to update.

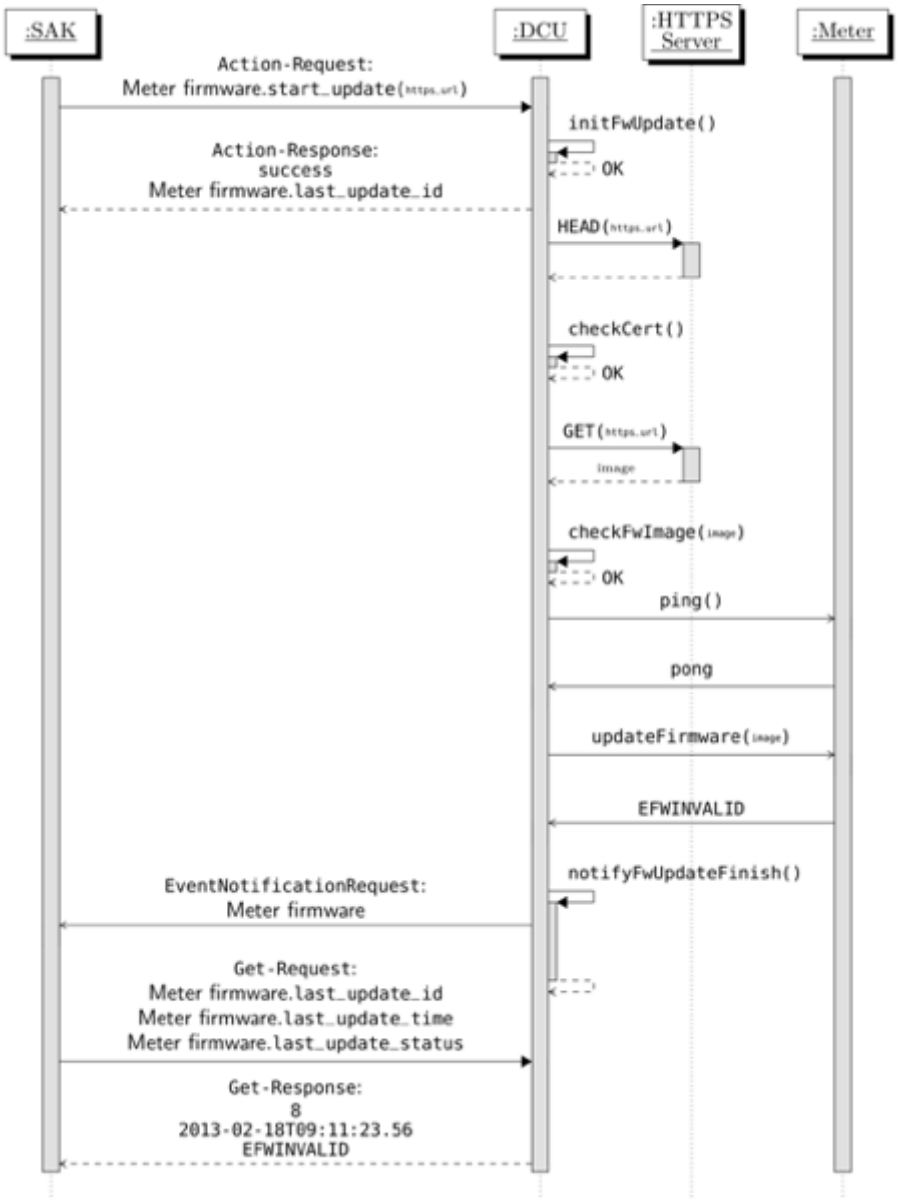

Figure 29 Sequence diagram of meter software update – 6, unsuccessful option

It has been assumed that an image transferred to a meter is not compatible.

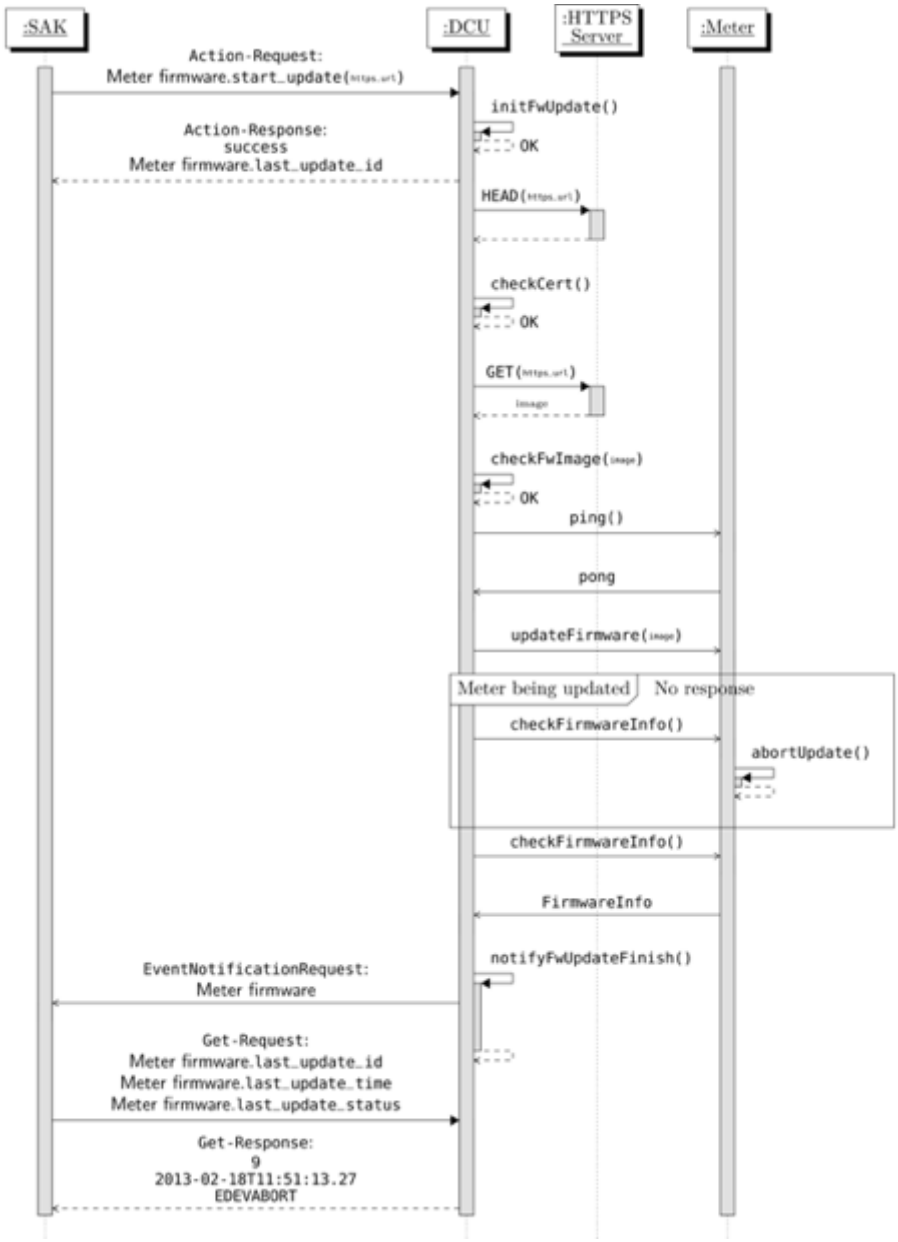

Figure 30 Sequence diagram of meter software update – 7, unsuccessful option

It has been assumed that a meter interrupted updating of an image.

Relay disconnecting

Relay disconnecting in a meter consists in sending command *Action-Request* calling method *remote_disconnect()* of object *Disconnect control* of an indicated meter. The completed operation is confirmed to the acquisition system.

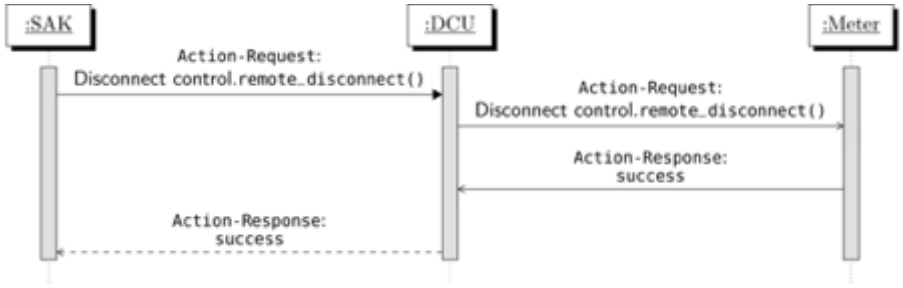

Figure 31 Sequence diagram of relay's disconnection in a meter

It has been assumed that a concentrator communicates with a meter in the application layer by means of the DLMS protocol.

Receiving meter events

The events such as opening a meter's cover and other are logged in an event log (there may be multiple event logs). The acquisition system read a buffer of such a profile object, using selective choice. A range descriptor is used here (structure *range_descriptor*; presented in [1] and [7] controlled with time (in field *restricting_object* specify descriptor COSEM 2. of the attribute of the timer's object representing the time – the events are always logged with the time of their occurrence). Unfortunately the range expressed by this descriptor (fields *from_value* and *to_value*) is limited on both sides, therefore the acquisition system states, as the lower limit, the time from the recently received record from a given events log increased by epsilon, which, in the case of type *data-time* means incrementation by one hundredth of a second. Upper limit is current time. As a result, as a response from a concentrator there are sent only the latest entries of the log and we avoid downloading the records that are already stored.

The acquisition system may periodically ask a concentrator about new records of the events log or may wait for release with a notification message. In the case of using the mechanism of notifications, enable this mechanism before requesting returning of the records that are newer than the last previously received, because this way we will not lose any notification. In the worst case scenario, we will get a notification when waiting for records or during their loading. It may apply to an event contained in the ordered range or not. For this reason, upon receipt of such a notification, the acquisition system must again order the concentrator to return the last records (this time with a new range), and it may receive an empty table as a response.

Figure 32 Sequence diagram of receiving new events from a meter

It has been assumed that in a concentrator there are active acceleration of access to data from meters and notification as well as that attribute *capture_objects* of object *Event log 1* contains the time and *Event code object #1*.

8. Recommendations for implementation of server of DCSAP protocol

TCP port

A recommended number for the port of DCSAP protocol is port number 16000. For DCSAP + SSL - 16443.

Number of handled, parallel DCSAP sessions

In the protocol there have not been defined limitations for a quantity of communication sessions. A recommended number of sessions opened by the acquisition system at multi-session communication with a concentrator consists of 3 – a constantly maintained session for the stream of instructions, constantly maintained session for receiving events as well as a diagnostic session appointed where necessary.

In most cases, communication with DCU will be operated by means of a single DCSAP session.

Meter list size

An acceptable number of entries in the table storing a list of meters (represented by attribute *max_entries* of object *Data concentrator meter list*) with which a concentrator has or had communicated should be a couple of times greater than the number of meters at stations. Recommended minimum value is 2048.

Events log size

The size of the events log, implemented by an object *Data concentrator event list* should be large enough so that in practice, after a possible loss of communication with the concentrator, allow for retrieving the history of the event. The recommended minimum value is 16384.

DCSAP session safety

It is recommended to use SSL with AES-CBS enciphering and authentication by means of a digital certificate (and DSA algorithm).

Time synchronization

DCSAP protocol assumes synchronization of time between DCU and the acquisition system. The mechanism of time synchronization is, however, outside the protocol's specification. A recommended method of time synchronization is NTP protocol [9] and use of the same time servers as the CBP/AMI application. The list of NTP servers, implemented by the object of the concentrator *Data concentrator NTP server list* should allow for defining at least 3 servers.

Throughput of a single DCSAP session

For the needs of implementation, it should be assumed that the average bandwidth of the connection to a single concentrator is 64 kbit/s and the data counted collectively as part of all sessions will be transferred with such speed. Notifications are light messages and diagnostic sessions will be established incidentally, therefore at multi-session communication with a concentrator, it can be assumed that simultaneous use of the connection by many sessions, although must be supported, will occur rarely enough so that the particular sessions will be able to use the available bandwidth in full.

DCSAP server model

Due to the ease of testing and implementation the preferred model of DCSAP server implementation is a parallel server with the use of separate threads for each session. An advantage of a parallel server is a possibility to implement sequential service of the acquisition' instructions.

Flow control in TCP connection

A concentrator receives commands sent by the acquisition system. It may happen that in the short period of time it will not be able to process the sent commands due to limited resources (CPU, memory). In such a case a concentrator may rely on cessation of receiving subsequent bytes from a socket, which will result in suspension of a TCP connection towards a concentrator. Such a solution is unfavourable due to the lack of possibilities of sending commands with high priority in this situation. However, it is assumed that for acquisition to be effective, a concentrator must be able to keep up with execution of commands, and short-term suspensions of flow, caused by temporary overload of a concentrator, are acceptable. If practice shows the need of application of an additional mechanism, it will be introduced in subsequent versions of a protocol as an optional mechanism.

9. Description of reference implementation

Reference implementation of DCSAP protocol is an example of its realisation in C language. Its purpose is to demonstrate work of a protocol, show details of interactions between the parties as well as suggest a simple and clear decomposition. There have been introduced a division into subsystems responsible for a specific part of functionality of the whole protocol.

Presented below is the description of each subsystem and their functions.

A-XDR coding

This is a set of functions allowing coding and decoding of messages written according to A-XDR standard. All functions transform the values from a variable, which is a calling argument, to a serial form in the allocated buffer or in reverse. The arguments of calling are always a double reference to a buffer as well as the indicator for its size. As a result of correct execution of operation, both reference to a buffer or its size are updated in such a way so as to reference to next element. The returned values inform about an error or, in the case of a positive value, about the required minimum buffer size, necessary for correct execution of an operation.

```
int32 axdr_read_choice(uint8 **buff, uint32 *bufflen, uint8 *val);

int32 axdr_read_enum(uint8 **buff, uint32 *bufflen, uint8 *val);
```

Reading element *CHOICE* or *ENUMERATED* from a buffer. One byte from the buffer is utilized. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a buffer with encoded form of data.
- *bufflen* – reference to the buffer's size.
- *val* - reference to a variable completed with the decoded value.

Result:

- Value 0 meaning correct execution.
- Value 1 meaning too low a buffer and determines its minimum size.

```
int32 axdr_read_optional(uint8 **buff, uint32 *bufflen, int *val);

int32 axdr_read_bool(uint8 **buff, uint32 *bufflen, int *val);
```

Reading element *OPTIONAL* or *BOOLEAN* from a buffer. One byte from the buffer is utilized.

The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a buffer with encoded form of data.
- *bufflen* – reference to the buffer's size.
- *val* – reference to a variable completed with a decoded value.

Result:

- Value 0 meaning correct execution.
- Value 1 meaning too low a buffer and determines its minimum size.

```
int32 axdr_read_int32(uint8 **buff, uint32 *bufflen, uint32 len, int32 *val);
int32 axdr_read_uint32(uint8 **buff, uint32 *bufflen, uint32 len, uint32 *val);
int32 axdr_read_int64(uint8 **buff, uint32 *bufflen, uint32 len, int64 *val);
int32 axdr_read_uint64(uint8 **buff, uint32 *bufflen, uint32 len, uint64 *val);
```

Reading element of value *INTEGER*, 32 or 64 bit with sign or without from the buffer. The indicated number of bytes from the buffer is utilized. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a buffer with encoded form of data.
- *bufflen* - reference to the buffer's size.
- *len* - size of encoded form (value ranging from 1–4 for 32-bit variants or 1–8 for 64-bit variants).
- *val* – reference to a variable completed with the decoded value.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.
- <0 incorrect argument *len* value.

```
int32 axdr_read_int32var(uint8 **buff, uint32 *bufflen, int32 *val);
int32 axdr_read_uint32var(uint8 **buff, uint32 *bufflen, uint32 *val);
int32 axdr_read_int64var(uint8 **buff, uint32 *bufflen, int64 *val);
int32 axdr_read_uint64var(uint8 **buff, uint32 *bufflen, uint64 *val);
```

Reading element of value *INTEGER*, 32 or 64 bit with sign or without from the buffer, encoded in the form of variable length. If a buffer does not contain the whole element, the operation will not be executed, and the returned value will indicate minimum needed size of a completed buffer. In the case of success, the buffer's reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a buffer with encoded form of data.
- *bufflen* - reference to the buffer's size.
- *val* - reference to a variable completed with the decoded value.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.
- <0 value beyond range of 32 or 64 bits.

```
int32 axdr_read_len(uint8 **buff, uint32 *bufflen, uint32 *val);
```

Reading length of the next element from a buffer. The length itself is encoded in the form of variable length. If a buffer does not contain the whole element, the operation will not be executed, and the returned value will indicate minimum needed size of a completed buffer. In the case of success, the buffer's reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a buffer with encoded form of data.
- *bufflen* - reference to the buffer's size.
- *val* - reference to a variable completed with the decoded value.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.
- <0 value outside the range of 32 bits.

```
int32 axdr_read_string(uint8 **buff, uint32 *bufflen, uint32 len, void *val);

int32 axdr_read_bitstring(uint8 **buff, uint32 *bufflen, uint32 len, void *val);
```

Reading sequence of bytes or bits with specified length from a buffer. The indicated number of bytes from the buffer is utilized. In the case of reading bits, the utilized is a minimum number of bytes containing the indicated number of bits. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a buffer with encoded form of data.
- *bufflen* - reference to the buffer's size.
- *len* - size of sequence of bytes or bits. Must be larger than 0.
- *val* - reference to target place, of readout sequence.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.
- <0 incorrect sequence size.

```
int32 axdr_write_choice(uint8 **buff, uint32 *bufflen, uint8 val);

int32 axdr_write_enum(uint8 **buff, uint32 *bufflen, uint8 val);
```

Writing of element *CHOICE* or *ENUMERATED* to a buffer. One byte of the buffer is utilized. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a target buffer.
- *bufflen* - reference to the buffer's size.
- *val* - encoded value.

Result:

- Value 0 meaning correct execution.
- Value 1 meaning too low a buffer and determines its minimum size.

```
int32 axdr_write_optional(uint8 **buff, uint32 *bufflen, int val);

int32 axdr_write_bool(uint8 **buff, uint32 *bufflen, int val);
```

Writing of element *OPTIONAL* or *BOOLEAN* to a buffer. One byte of the buffer is utilized. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a target buffer.
- *bufflen* - reference to the buffer's size.
- *val* - encoded value.

Result:

- Value 0 meaning correct execution.
- Value 1 meaning too low a buffer and determines its minimum size.

```
int32 axdr_write_int32(uint8 **buff, uint32 *bufflen, uint32 len, int32 val);
int32 axdr_write_uint32(uint8 **buff, uint32 *bufflen, uint32 len, uint32 val);
int32 axdr_write_int64(uint8 **buff, uint32 *bufflen, uint32 len, int64 val);
int32 axdr_write_uint64(uint8 **buff, uint32 *bufflen, uint32 len, uint64 val);
```

Writing element of value *INTEGER*, 32 or 64 bit with sign or without to the buffer. The indicated number of bytes of the buffer is utilized. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a target buffer.
- *bufflen* - reference to the buffer's size.
- *len* - size of encoded form (value ranging from 1–4 for 32-bit variants or 1–8 for 64-bit variants).
- *val* - encoded value.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.
- <0 incorrect argument value *len*.

```
int32 axdr_write_int32var(uint8 **buff, uint32 *bufflen, int32 val);
int32 axdr_write_uint32var(uint8 **buff, uint32 *bufflen, uint32 val);
int32 axdr_write_int64var(uint8 **buff, uint32 *bufflen, int64 val);
int32 axdr_write_uint64var(uint8 **buff, uint32 *bufflen, uint64 val);
```

Writing element of value *INTEGER*, 32 or 64 bit with sign or without to the buffer, encoded in the form of variable length. If a buffer does not carry the whole element, the operation will not be executed, and the

returned value will indicate the minimum needed size of the buffer. In the case of success, the buffer's reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a target buffer.
- *bufflen* - reference to the buffer's size.
- *val* - encoded value.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.

```
int32 axdr_write_len(uint8 **buff, uint32 *bufflen, uint32 val);
```

Writing length of the next element to a buffer. The length itself is encoded in the form of variable length. If a buffer does not carry the whole element, the operation will not be executed, and the returned value will indicate the minimum needed size of the buffer. In the case of success, the buffer's reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a target buffer.
- *bufflen* - reference to the buffer's size.
- *val* - encoded value.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.

```
int32 axdr_write_string(uint8 **buff, uint32 *bufflen, uint32 len, void *val);

int32 axdr_write_bitstring(uint8 **buff, uint32 *bufflen, uint32 len, void *val);
```

Writing a sequence of bytes or bits with specified length to a buffer. The indicated number of bytes is saved to the buffer. In the case of saving bits, utilized is a minimum number of bytes containing the indicated number of bits. The buffer reference is moved and its size reduced.

Arguments:

- *buff* - double reference to a target buffer.
- *bufflen* - reference to the buffer's size.

- *len* - size of sequence of bytes or bits. Must be larger than 0.
- *val* - reference to a sequence of bytes or bits.

Result:

- Value 0 meaning correct execution.
- > 0 meaning too low a buffer and determines its minimum size.
- <0 incorrect sequence size.

Use of TCP sockets

A set functions packing TCP sockets in a non-blocking mode. It allows easy creation of the client and server sockets, establishing connections or their accepting as well as sending and receiving data.

```
int tcpsocket_open(char *host, uint16 port, int32 timeout);
```

Creates a client socket as well as establishes connection with the server. If operation fails in the preset time, an error is returned.

Arguments:

- *host* – server domain name or its IP address.
- *port* – server port.
- *timeout* – time for performing the operation in milliseconds.

Result:

- > = 0 socket descriptor.
- <0 failure.

```
int tcpsocket_close(int sockfd);
```

Closes connection and the client's socket.

Arguments:

- *sockfd* - socket's descriptor.

Result:

- 0 success

```
int tcpsocket_server(uint16 port);
```

Creates a server socket listening on an indicated port.

Arguments:

- *port* – server port.

Result:

- > = 0 socket descriptor.
- <0 failure.

```
int tcpsocket_accept(int sockfd);
```

Approves connection from a client. Socket's descriptor handling a connection with the client is returned.

Arguments:

- *sockfd* - server socket's descriptor.

Result:

- > = 0 socket's descriptor handling connection with a client.
- <0 failure.

```
int tcpsocket_send(int sockfd, uint8 *buff, uint32 len, int32 timeout);
```

Sends data through an open socket. If operation fails in the preset time, an error is returned.

Arguments:

- *sockfd* - socket's descriptor.
- *buff* – a buffer with data for sending.
- *len* – quantity of data for sending.
- *timeout* – time for performing the operation in milliseconds.

Result:

- 0 data was sent.
- <0 failure.

```
int tcpsocket_receive(int sockfd, uint8 *buff, uint32 len, int32 timeout);
```

Receives data from an open socket. If operation fails in the preset time, an error is returned.

Arguments:

- *sockfd* - socket's descriptor.
- *buff* - buffer for data.
- *len* – quantity of data for receiving.
- *timeout* – time for performing the operation in milliseconds..

Result:

- 0 data received.
- <0 failure.

DCSAP server

Simple implementation of a listening server, accepting connections from clients. The server activates a separate thread waiting for new connections. For each new client, there is created a dedicated connection supported in a subsystem of DCSAP connections.

```
int dcsapserver_start(uint16 port, int32 timeout, int32 idle, int connmax);
```

It activates a monitoring server on all interfaces on the indicated port.

Arguments:

- *port* - server port.
- *timeout* – time in milliseconds for execution of operations of sending and receiving of data in the established connection with the client.
- *idle* – time in milliseconds of maximum idleness of a client, after which its disconnection will take place.
- *connmax* – maximum number of simultaneously operated clients.

Result:

- 0 server launched.
- <0 failure.

```
static void *dcsapserver_thread(void *data);
```

Internal function executing the code of DCSAP server thread.

Subsystem of DCSAP connections

This subsystem supports independent connections from clients. For the needs of each of them a structure is created describing a connection as well as a dedicated thread is activated. It is responsible for receiving messages as well as their initial parsing. Empty messages are immediately sent back to the client in accordance with the protocol requirements. As part of a connection, session objects are implemented controlling notifications as well as possible buffering of meters' responses. The commands are forwarded to another subsystem of DCSAP objects. Transfer of a message to another subsystem is connected with increasing a counter of references of a given connection what prevents removal of the structure with its description. It results from the fact that the reference to the structure of a connection cannot be cancelled until the time when a response is sent back to all transferred commands.

A subsystem of connections is also responsible for distribution of messages of notifications to appropriate clients. They are received only by those clients who activated the mechanism of asynchronous notifications by means of a suitable session object.

```
int dcsapconn_create(int sockfd, int32 timeout, int32 idle, int connmax);
```

Activates operation of a new connection initiated by the client. If maximum number of clients is already operated, the function does not work. During correct execution, a structure is created describing a connection, and then it is added to a list of operated connections. Session objects are created. A thread handling receipt of messages is activated. This function is triggered by a thread of DCSAP server.

Arguments:

- *sockfd* - socket's descriptor of a connection.
- *timeout* – time in milliseconds for execution of operations of sending and receiving data in the connection established with the client.
- *idle* – time in milliseconds of maximum idleness of a client, after which its disconnection will take place.
- *connmax* - maximum number of simultaneously operated clients.

Result:

- 0 operation of a connection was launched.
- <0 failure or exceeded maximum number of simultaneously operated clients.

```
int dcsapconn_response(dcsapconn_data_t *conn, uint32 deviceid, uint64 messageid, int32
datasize, uint8 *dlmsdata);
```

Sends back a response to a received command if disconnection did not take place. Reduced is a counter of references signalling release of resources which is the structure describing a connection. In the case of a disconnection, this structure must stay in memory until the subsequent subsystems return response to all commands transferred to them received as part of this connection. Synchronizing sending of many answers simultaneously is executed with dedicated critical section, independent for each connection.

Arguments:

- *conn* – reference to the structure connection.
- *deviceid* – device identifier.
- *messageid* – message identifier.
- *datasize* - DLMS data size. Negative value means an error or no data available.
- *dlmsdata* - DLMS data, only in the case of positive value of argument *datasize*.

Result:

- 0 response was sent.
- <0 failure. Connection will be closed, do not repeat.

```
int dcsapconn_notify(uint32 deviceid, int32 datasize, uint8 *dlmsdata);
```

Sends notification. Connections are searched with an activated mechanism of notifications. A counter of references is increased for each of them and then, already beyond the critical section protecting a list of connections, function *dcsapconn_response* is called and a notification sent.

Arguments:

- *deviceid* – device identifier.
- *datasize* – DLMS data size. Must be positive.
- *dlmsdata* - DLMS data.

Result:

- 0 notification was sent.
- <0 failure or incorrect datasize.

```
static void *dcsapconn_thread(void *data);
```

Internal function performing the thread code of DCSAP connection.

Subsystem of DCSAP objects

Subsystem of objects enables dynamic registering and removing global and session COSEM objects. The objects may be bound with an address of the concentrator or any meter. Subsystem stores the list of all registered objects, and for each of them a class, instance identifier, reference to data structure and reference to the functions processing simple commands for object (*get*, *set* as well as *action*). This subsystem performs parsing DLMS messages and transferring to be executed in the context of the indicated instance of an object. The commands that will not be served by the subsystem of DCSAP objects get to the subsystem of meters.

```
int dcsapobjects_register(dcsapconn_data_t *conn, uint32 deviceid, uint16 classid,
uint64 instanceid, void *data, get_function get, set_function set, action_function
action);
```

It registers the COSEM object with global or session character in the case of giving an reference to the connection's structure. The objects may be bound with an address of the concentrator or any meter. What is transferred is class, instance identifier, reference to data structure as well as references to the functions processing simple commands for object (*get*, *set* as well as *action*).

Arguments:

- *conn* – reference to the connection structure or NULL for global objects.
- *deviceid* – device identifier.
- *classid* – object class.
- *instanceid* – object instance.
- *data* - reference to object data.
- *get* - reference to the function processing *get* commands.
- *set* - reference to the function processing *set* commands.
- *action* - reference to the function processing *action* commands.

Result:

- 0 object has been registered.
- <0 failure or object exists.

```
int dcsapobjects_delete(dcsapconn_data_t *conn, uint32 deviceid, uint16 classid, uint64
instanceid);
```

Removes the COSEM object from the subsystem list. A critical section protects against removal of an object, which participates in execution of a command.

Arguments:

- *conn* - reference to the connection structure or NULL for global objects.
- *deviceid* – device identifier.
- *classid* – object class.
- *instanceid* – object instance.

Result:

- 0 object has been removed from the list.
- <0 object exists.

```
int dcsapobjects_request(dcsapconn_data_t *conn, uint32 deviceid, uint64 messageid,
int32 datasize, uint8*dlmsdata);
```

A command is performed addressed to the registered objects. Address of the object data and the function handling it are selected from the list of the registered objects. If an address is given to a meter object which is not reflected in this subsystem, the command will be transferred to the subsystem of meters.

Arguments:

- *conn* - reference to the connection's structure.
- *deviceid* - device identifier.
- *messageid* - message identifier.
- *datasize* - DLMS data size. Must be positive.
- *dlmsdata* - DLMS data.

Result:

- 0 a command has been processed or transferred to the subsystem of meters.
- <0 failure or incorrect *datasize*.

Subsystem of meters

The commands that a concentrator should transfer for implementation to real meters are directed here. They are put into a queue from which they may be chosen in any order. A separate queue handles priority commands. This subsystem may implement buffering of some meter responses.

```
int dcsapmeters_request(dcsapconn_data_t *conn, uint32 deviceid, uint64 messageid,
int32 datasize, uint8*dlmsdata);
```

It accepts for execution a command addressed to real meters. The message is copied to local queue. Queue of priority commands is operated separately. Control is returned without awaiting execution of a command. The subsystem is responsible for passing responses to a subsystem of DCSAP connections.

Arguments:

- *conn* – reference to the connection's structure.
- *deviceid* – device identifier.
- *messageid* – message identifier.
- *datasize* - DLMS data size. Must be positive.
- *dlmsdata* - DLMS data.

Result:

- 0 a command has been accepted for processing.
- <0 failure or incorrect *datasize.*

```
int dcsapmeters_cancel(dcsapconn_data_t *conn);
```

The function cancels the non-executed commands accepted as part of the indicated connection. It is called from the subsystem of DCSAP connections. It returns the quantity of cancelled operations that will not return any result.

Arguments:

- *conn* - reference to the connection's structure.

Result:

- Number of cancelled commands.

```
int dcsapmeters_start();
```

The function activates a thread of the subsystem of meters.

Result:

- 0 success.

static void *dcsapmeters_thread(void *data)

An internal function performing the thread code of the subsystem of meters.

10. Appendix list

Appendix 1: COSEM objects definitions

11. Source references

references to the source materials:

1. Blue Book 12th edition, TECHNICAL REPORT, Companion Specification for Energy Metering, COSEM Identification System and Interface Classes, DLMS User Association. Reference number: DLMS UA 1000-1, Ed. 12.0, 2014-09-10.
2. Green Book 8th edition, TECHNICAL REPORT, Companion Specification for Energy Metering, DLMS/COSEM Architecture and Protocols, DLMS User Association. Reference number: DLMS UA 1000-2, Ed. 8.0, 2014-07-07.
3. IEC 61334-6:2000, Distribution automation using distribution line carrier systems – Part 6: A-XDR encoding rule, 1st ed., 2000-06.
4. IEC 62056-21:2002, Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange, 1st ed., 2002-05.
5. IEC 62056-53:2006, Electricity metering – Data exchange for meter reading, tariff and load control – Part 53: COSEM application layer, 2nd ed., 2006-12.
6. IEC 62056-61:2006, Electricity metering – Data exchange for meter reading, tariff and load control – Part 61: Object identification system (OBIS), 2nd ed., 2006-11.
7. IEC 62056-62:2006, Electricity metering – Data exchange for meter reading, tariff and load control – Part 62: Interface classes, 2nd ed., 2006-11.
8. Draft Standard for PowerLine Intelligent Metering Evolution, version R1.3.6, PRIME Alliance TWG.
9. Mills et al., RFC5905, Network Time Protocol Version 4: Protocol and Algorithms Specification, 2010-06.
10. Feuerhahn et al., „Comparison of the communication protocols DLMS/COSEM, SML and IEC 61850 for smart metering applications”, in: IEEE International Conference on Smart Grid Communications, 2011, pp. 410–415.