



**DLMS/COSEM Client Object Library**  
**User Manual**

**SCL Revision Summary**

SCL Version	Release Date
2.1.5.6	19-March-09

# Table of Contents

Table of Contents.....	2
<b>INTRODUCTION.....</b>	<b>3</b>
<u>OVERVIEW.....</u>	<u>3</u>
<u>IMPLEMENTATION OVERVIEW.....</u>	<u>3</u>
<b>CLIENT LIBRARY FUNCTIONS.....</b>	<b>3</b>
Signature.....	5
Return Value.....	5
Parameters.....	5
<b>CALL SEQUENCE.....</b>	<b>11</b>
<b>DLMS_UNION AND ACTION_UNION STRUCTURES.....</b>	<b>12</b>

# Introduction

## Overview

The DLMS/COSEM Client Object Library provides a complete implementation of a 62056 COSEM/HDLC protocol stack to enable interested OEMs to create DLMS Client Applications to read/write DLMS-enabled meters and devices.

The stack is based on the following specifications

IEC-62056-42: Physical Layer

IEC-62056-46: HDLC Datalink Layer

IEC-62056-47: Cosem transport layers for IPv4 networks

IEC-62056-53: COSEM Application Layer

IEC-62056-61: OBIS (OBject Identification System)

IEC-62056-62: Interface Classes

## Implementation Overview

The client Object Library is provided as a Windows DLL or as a Linux shared library. Other platform-implementations can also be provided on request.

The DLL is shipped with a sample application to demonstrate the implementation methodology.

The exported DLL functions provide a means to open and initialize a serial or IP port/, establish a HDLC connection (only for HDLC based communication profile), establish an application association, and then functions to read/write/execute attributes and methods in the connected DLMS server (meter). The client application can then call functions to disconnect from the meter, close the communication port and cleanup memory.

## Client Library Functions

The following are the exported functions in the Client Object Library.

### 1. InitClient

This function is to be used to initialize the client as the first step when using the library

#### Signature

```
KDEFS_UCHAR initClient( KDEFS_USHORT respTimeout,  
                        KDEFS_USHORT frmTimeout,  
                        KDEFS_USHORT LinkBufSz,  
                        KDEFS_ULONG cosemBufSz);
```

#### Return Value

SUCCESS if function succeeds,  
Else FAILURE.

#### Parameters

1. respTimeout

Response timeout used by the client, in milliseconds. This is the amount of time that the Client is to wait after sending a request without receiving a response before declaring that the request has timed-out.

2. frmTimeout

Frame timeout in milliseconds. This is the amount of time that the Client will wait after receiving the initial few bytes of a frame-header, for the rest of the frame to be received.

3. LinkBufSz

HDLC/wrapper(for IPv4) buffer size. Suggested value is 512 bytes. This allocates the size of the HDLC or TCP/IP Layer buffers for received and transmit data.

4. cosemBufSz

COSEM buffer size, This is the size of the COSEM Application Layer buffers that hold the received data when performing a Get/Read operation. This should be a large value to hold the largest possible attribute value that may be read from the meter. Certain array-type attributes (for example, the Buffer attribute of a Profile object) may contain several entries. This buffer should be sized appropriately to hold the entire response.

**2. InitPort**

This function initializes and opens a communication port.

**Signature**

```
KDEFS_UCHAR initPort(KDEFS_UCHAR commProfile,  
                    KDEFS_UCHAR* client_ipAddr,  
                    KDEFS_UCHAR* server_ipAddr,  
                    KDEFS_USHORT server_port,  
                    KDEFS_USHORT wClientAddr,  
                    KDEFS_USHORT wServerAddr,  
                    KDEFS_UCHAR* port,  
                    KDEFS_ULONG baud,  
                    KDEFS_UCHAR parity)
```

**Return Value**

SUCCESS if function succeeds,  
Else FAILURE. (Here SUCCESS is 0, and FAILURE is 1)

**Parameters**

1. commProfile

Four choices for communication profile are as follows

- 1 => Direct HDLC
- 2 => Mode E
- 3 => TCP IP
- 4 => UDP

2. client\_ipAddr (Required only for IP based communication profile)

IP address of client as string (e.g. "127.0.0.1")

3. server\_ipAddr (Required only for IP based communication profile)

IP address of server as string (e.g. "127.0.0.1")

4. server\_port (Required only for IP based communication profile)  
UDP/TCP port of server
5. wClientAddr  
Client wrapper port number
6. wServerAddr  
Server wrapper port number
7. port  
The serial port name as a string.
8. baud  
The baud rate for communication.
9. parity  
Specifies the parity scheme to be used, 0-4 = no, odd, even, mark, space

### 3. ModeE\_Init

This function performs the mode E (IEC 62056-21) handshaking before switching to COSEM/ HDLC.

NOTE: This function is required only if opening mode is Mode E

#### Signature

```
KDEFS_UCHAR ModeE_Init(KDEFS_UCHAR *devAddr,
                       KDEFS_UCHAR *XXX,
                       KDEFS_UCHAR *Ident)
```

#### Return Value

SUCCESS if function succeeds,  
Else FAILURE. (Here SUCCESS is 0, and FAILURE is 1)

#### Parameters

1. devAddr  
Device address as string (optional field, manufacturer specific, 32 characters maximum)
2. XXX  
String (Manufacturer's identification comprising of three upper case letters)
3. Ident  
String (Identification, manufacturer specific, 16 printable characters maximum except for "/" and "!")

### 4. ClosePort

For closing the communication port.

#### Signature

```
KDEFS_UCHAR closePort();
```

#### Return Value

SUCCESS if function succeeds,

Else FAILURE.

## 5. SendSNRM

Function is used to establish a MAC connection by sending an SNRM (Set Normal Response Mode) frame. This function if successful establishes a link-layer connection to the Meter. The significant parameters are the client and server addresses which will be compared by the Meter against its configured list of Association client IDs and server logical-device ids.

This function usually also negotiates the HDLC parameters that define the Info-field size and the Window size for all ensuing transactions. The user is expected to send the required values, however the final values used in further communication may be different from these, since the meter will negotiate these parameters and set the minimum values that is acceptable to both client and server.

NOTE: This function is required only for HDLC based communication profile.

### Signature

```
KDEFS_UCHAR sendSNRM(KDEFS_UCHAR address_size,  
                    KDEFS_UCHAR clntID,  
                    KDEFS_USHORT serverID,  
                    KDEFS_USHORT dev_addr,  
                    KDEFS_BOOL negoParams,  
                    KDEFS_UCHAR windowRx,  
                    KDEFS_UCHAR windowTx,  
                    KDEFS_UCHAR infoFieldLenRx,  
                    KDEFS_UCHAR infoFieldLenTx );
```

### Return Value

SUCCESS if function succeeds

Else an error code indicating the reason for error

### Parameters

#### 1. address\_size

Number of bytes used for addressing the server, can be 1, 2 or 4.

When a value of 1 is specified, the destination address (server-address) contains only 1 byte which contains the Upper-HDLC address of the meter. This can be used for point-to-point communication.

In case of a multi-drop scenario (daisy-chained meters), the address must also specify the lower-HDLC or device-address to identify which physical device is being addressed. This can use a 2-byte or a 4-byte destination address. In case of a 2-byte address, the server address has 1-byte Upper-HDLC address and 1-byte Lower-HDLC address. In case of 4-byte addressing, the server-address contains 2-bytes Upper and 2-bytes Lower HDLC addresses.

#### 2. clntID

Client id for current association. This is the ID of the client driver. The client-ID is used by the server to determine which association in the addressed Logical-device is to be activated. A value of 16 (0x10) is pre-defined by the DLMS standard to specify a Public-Access client. Any DLMS server must provide at least one Public-Access association

#### 3. serverID

Upper HDLC address of the server. This identifies which Logical-device in the addressed-meter is being queried. Any DLMS-server will provide at least one Logical-device with an address of 1, which indicates the Management Logical device

#### 4. dev\_addr

Lower HDLC address of the server. This is the physical-device address of the meter. Please note that a physical device can contain multiple logical devices identified by unique Upper-HDLC addresses

In turn, each Logical device can support multiple associations (with different object-lists and access-rights) defined by unique client-IDs

#### 5. negoParams

Flag indicating, whether to negotiate the following parameters. As mentioned earlier, the SNRM/UA exchange can also be used to negotiate the following HDLC layer parameters

#### 6. windowRx

Number of windows receive. This identifies the number of segmented frames that can be received at a go by this client, before requiring to send a RR (Request-Ready) handshake. The DLMS default value is 1.

#### 7. windowTx

Number of windows transmit. This identifies the number of segmented frames that can be transmitted at a go by this client, before requiring a RR (Request-Ready) handshake from the server. The DLMS default value is 1.

#### 8. infofieldLenRx

Information field length receive. This defines the size of the Info-field part of a received DLMS frame. The info-field is the carrier of all COSEM data destined to be processed by the upper Layers. The DLMS default value is 128

#### 9. infoFieldLenTx

Information field length transmit. This defines the size of the Info-field part of a transmitted DLMS frame. The info-field is the carrier of all COSEM data destined to be processed by the upper Layers. The DLMS default value is 128

### 6. SendDISC

This function can be used to send the disconnect frame which disconnects both the Link-layer as well as the Application layers. A DISC frame can be sent after a successful SNRM and also after a successful Association.

If a DISC frame is not sent after either case above and if there is no further activity from the client, the DLMS server will automatically go to disconnected mode (DM) after a configured Inactivity timeout.

NOTE: This function is required only for HDLC based communication profile.

#### Signature

```
KDEFS_UCHAR sendDISC( KDEFS_UCHAR clientID,  
                      KDEFS_USHORT serverID,
```

KDEFS\_USHORT dev\_addr);

**Return Value**

0 in case of success  
error code in case of failure

**Parameters**

1. clientID  
The client id of the current association. Same as above in the SNRM frame.
2. serverID  
Upper HDLC address of the server. Same as above in the SNRM frame.
3. dev\_addr  
Lower HDLC address of the server. Same as above in the SNRM frame.

**7. SendAARQ**

This function is used to establish an application association which connects the COSEM layers of the client and server. This can only be called after a successful SNRM which connects the lower layers. The parameters of the Association must match the settings in the meter for the Association that was identified in the SNRM by the client and server ids.

**Signature**

KDEFS\_UCHAR sendAARQ(KDEFS\_UCHAR app\_ctxt,  
KDEFS\_UCHAR auth\_mech,  
KDEFS\_UCHAR\* passwd,  
KDEFS\_UCHAR passwd\_len,  
KDEFS\_ULONG conformance,  
KDEFS\_USHORT max\_apdu\_size\_recv);

**Return Value**

0 in case of success  
otherwise an error code.

**Parameters**

1. app\_ctxt  
Application context. Values can be 1, 2, 3 or 4. The application context defines the referencing method to be used (LN or SN)  
1 for LN Association with No ciphering  
2 for SN Association with No ciphering  
3 for LN Association with ciphering  
4 for SN Association with ciphering.  
NOTE: Ciphering contexts are not supported by this client library.
2. auth\_mech  
Authentication mechanism used. Values can be 0, 1 or 2.  
0 for Lowest Level Security (without password)  
1 for Low Level Security (with static password)  
2 for High Level Security (with coded password)  
NOTE: High Level security is not supported by this client library.
3. \*passwd  
A pointer to the password string. NULL if password is not used (Lowest-level-security specified above).
4. passwd\_len

Length of the password. Zero if Lowest-level security specified above.

#### 5. conformance

This is the conformance block that specifies which functional components of the server are required by the client. This is a number formed by enabling the appropriate bits corresponding to specified functionalities.

For example a value of 0x180000 indicates Read and Write functions for SN and a value of 0x001818 indicates Get, Set, Block Transfer with Get and Block Transfer with Set functions. The supplied conformance block will be negotiated by the server by 'AND'ing with its own conformance block to return a negotiated value that contains the functions that are common to both.

#### 6. max\_apdu\_size\_rcv

This is the maximum apdu size that the client can receive. This is used only in LN referencing contexts that support Block transfer. In SN contexts a value of 0xFFFF can be specified.

#### 8. sendRLRQ

Release an already established application association.

NOTE: This function is required only for IP based communication profile.

#### Signature

```
KDEFS_UCHAR sendRLRQ(KDEFS_INT reason);
```

#### Return Value

0 in case of success  
error code in case of failure

#### 9. GetData

This function can be used to get values from the meter. The function returns the pointer to a structure in which the read data is filled. The definition for this structure can be found in the ICStructs.h header file.

The function automatically checks which context is being used and sends either a Get (for LN contexts) or a Read (for SN contexts). The returned structure pointer points to a structure that defines all attributes of all Interface classes supported. The user of the library is required to only read or process the attribute of the object that was requested. Attempts to read other attributes or other class objects may result in NULL errors.

#### Signature

```
DLMS_UNION* getData(OBISCODE *obis,  
                    KDEFS_USHORT base,  
                    KDEFS_USHORT ic,  
                    KDEFS_UCHAR attrIndex,  
                    SELACCESSPARAMS *selParams);
```

#### Return Value

On success a pointer to the DLMS\_UNION structure is returned

The first element of the DLMS\_UNION structure is a “result” value that indicates success or failure. The values in the proper fields of the structure are valid only if the ‘result’ field is 0 indicating success.

#### **Parameters**

1. \*obis

This is a pointer to the obis code of the object to be read. This is used if the context is LN and will be ignored otherwise

2. base

Base name of the object for reading. This is used if the context is SN and will be ignored otherwise

3. ic

Interface class id of the object to be read

4. attrIndex

Attribute index of the attribute within the object to be read.

5. selParams

This is a pointer to the structure variable, which has to be filled if selective access is required. NULL is to be passed if selective access is not required.

#### **10. CleanupMemory**

This function must be called after the getData function call, to free the memory allocated by that function. getData returns a DLMS\_UNION pointer that has various elements allocated dynamically. Before calling another getData() call, this function must be called to enable the library to clean-up the memory used.

#### **Signature**

```
void cleanupMemory();
```

#### **11. SetData**

This function can be used to write data to the meter. Before invoking this function, the write data must be filled within a DLMS\_UNION structure variable. Its address has to be passed through this function.

The user of the library is expected to allocate and fill in a DLMS\_UNION structure with the appropriate attribute data to be set. After the call completes the user is also expected to cleanup and release the memory used. The client library does not clean-up Set-Data.

#### **Signature**

```
KDEFS_UCHAR setData(DLMS_UNION* setData,  
                    OBISCODE *obis,  
                    KDEFS_USHORT base,  
                    KDEFS_USHORT ic,  
                    KDEFS_UCHAR attrIndex);
```

#### **Return Value**

0 if function succeeds  
otherwise an error code.

#### **Parameters**

1. \*setData

- This is a pointer to the structure variable in which the set data is properly filled.
2. \*obis  
This is a pointer to the obis code of the object to be written into.
  2. base  
Base name of the object to be written into.
  3. ic  
Interface class id of the object to be written into.
  4. attrIndex  
Attribute index of the object to be written into.

## 12. ExecuteAction

This function can be used to execute methods of objects. The method arguments have to be filled in a structure variable before invoking the function. The definitions of the ACTION\_UNION structure can be found in ICStructs.h file.

### Signature

```
KDEFS_UCHAR executeAction(ACTION_UNION* actnData,
                           OBISCODE *obis,
                           KDEFS_USHORT base,
                           KDEFS_USHORT ic,
                           KDEFS_UCHAR methIndex);
```

### Return Value

0 if function succeeds  
else an error code

### Parameters

1. \*actnData  
This is a pointer to a structure variable, in which the arguments for the method has to be passed in correctly.
2. \*obis  
This is a pointer to the obis code of the object of which the method is to be executed.
3. base  
Base name of the object.
4. ic  
Interface class id of the object.
5. methIndex  
Method index of the method to be executed.

## Call Sequence

The typical sequence of calling the library functions is as below

- **initPort()**
- **initClient()**
- **ModeE\_Init()** only for HDLC based communication profile, and opening mode is Mode E

- **sendSNRM()** only for HDLC based communication profile
- **sendAARQ()**  
Followed by any number of the following three in any order
- **getData()** (always followed by **cleanupMemory()** after processing data)
- **setData()**
- **executeAction()**  
Finally
- **sendDISC()** only for HDLC based communication profile
- **sendRLRQ** only for IP based communication profile
- **closePort()**

Please note that you can go back to sendSNRM after doing a sendDISC and continue polling another meter (or another association/logical-device in the same meter). HDLC functions (ModeE\_Init (), sendSNRM (), sendDISC ()) are not used for TCP/ UDP based communication profile.

## **DLMS\_UNION and ACTION\_UNION Structures**

The DLMS\_UNION structure is not a UNION, but a structure with all supported Interface-Class object structures within. Please refer to the ICStructs.h header file for the complete definitions of the structures.

This structure is the means to pass data between the user-application and the library. In case of Get/Read operations, the structure is allocated, filled-in and passed to the user-application from the library. In case of Set/Write operations, the user-application must allocate, fill-in and pass the structure to the library.

In case of Set/Write the user-application must also handle the freeing of memory allocated to the structure (In Get/Read, simply calling the cleanupMemory() function in the library will do)

The ACTION\_UNION structure is similar in procedure to the Set/Write operation. (In fact when using SN referencing the executeAction() method actually sends a Write command). When calling executeAction(), the user-application must allocate, fill-in and pass this structure to the library.