Application Report
SLAA614–November 2013

Using TI's DLMS COSEM Library

Chander B Goel and Steve Underwood

ABSTRACT

This application report describes in detail the usage of DLMS COSEM library developed by Texas Instruments for customers who use TI's microcontrollers in metering applications. The library is provided as object coded with a configuration file for ease of use. The library can be obtained by contacting the regional sales and marketing offices. The customer will have to sign an SLA before getting access to the library.

Contents

List of Figures

1 Introduction

DLMS stands for "Device Language Message specification". It is a generalized concept for abstract modeling of communication entities which, along with the COSEM (COmpanion Specification for Energy Metering) specification, has been standardized as a set of rules for data exchange with energy meters. The DLMS specification is developed and maintained by the DLMS User Association and has been adopted by the IEC TC13 WG14 into the IEC 62056 series of standards.

Texas Instruments has developed a compact and easy to use library that can be integrated into a metering application on TI's MSP430 microcontrollers. Section 2 describes the various features of this library.

2 Features

The DLMS Library provided by Texas Instruments supports the following features:

- All COSEM classes are supported. The examples of a few commonly used classes are given.
- Supports three associations: No Security (NS), Low Security (LS) and High Security (HS). In the HS association, a four step AES128 based authentication mechanism is used. These three associations are also referred as Public Client (PC), Meter Reader (MR) and Utility Setting (US) associations, respectively.
- Supports long name addressing
- Supports one, two and four byte addressing
- Supports GET, SET, GET WITH BLOCK, SET WITH BLOCK, ACTION and Selective Access requests
- A special mechanism for handling Profile Generic objects where multiple data entries are required
- Supports selective access in Profile Generic Class Buffer
- Passes all DLMS CTT V2.0 tests
- Code Size - ~24KB
- RAM Size ~ 1.8KB
- Easy configuration and ease of use

3 Prerequisites

For using this library, it is required that you have a thorough understanding of DLMS Blue Book Classes and embedded C programming background.

For any development on DLMS library, you need to have DLMS Client software so that the meter can be tested. There are a few companies like Kalkitech, Gurux, and so forth that provide a semi-functional demo software, which can be downloaded from the internet. For developing a final product, one would need to buy a full version of the Kalkitech DLMS Client.

You also need to have a full version of IAR Embedded Workbench® for MSP430.

4 File Structure

The following files are available in the library:

- iec62056_demo.c: This is the user interface file for system initialization. All the peripherals are initialized and the HDLC state machine is initialized.
- uart_comms.c: This is the configuration file for the UART module. Depending upon the requirement, the available USCIs can be initialized.
- iec62056_link.r43: This file implements HDLC and Mac layer. This file is provided as object coded file.
- server_msgs.r43: This file implements COSEM application layer. This file is also provided as object coded file.
- config.c: This file is the main configuration file for the user's system. In this file, one can add or remove parameters from the object list, which is described later in the document
- app_server_msgs.c: This file is the one where most of the user code has to be written. Here, one has to write all the functions, which are used to extract data from the memory and provide it to the DLMS library.
- config.h: Prototype declarations and macro definitions for config.c
- cosem.h: Definition of all the constants used in COSEM application layer.
- iec_62056_link.h:Prototype declarations and macro definitions for HDLC layer.

5 Object List Tables

The first step in starting to use the DLMS library is to add, remove or update the list of objects in the object list table (see Figure 1). The object list table is defined in config.c file. They are defined as "object_desc_s" data type. The different rows in the object list table represent different objects that the meter supports. An object is the representation of a metering parameter or a data log.

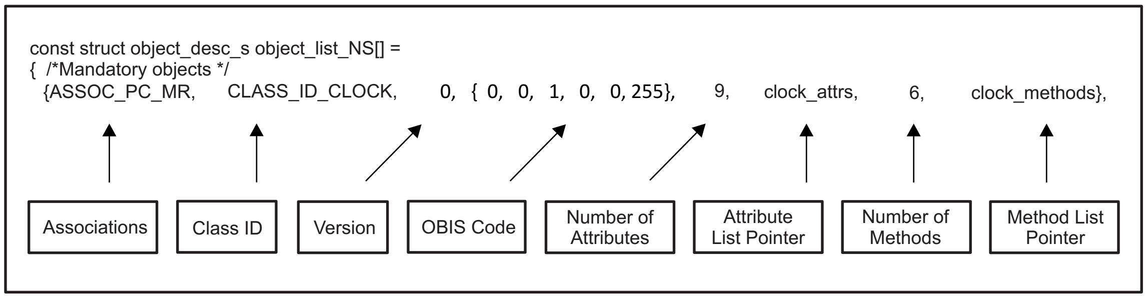

Figure 1. Object List Table

The following columns in the object list table appear in order:

- Associations: The associations where this object is visible.
- Class-ID: The class ID indicates which class does this object belongs to. This is an enumerated data type and various classes' names are available under macro-definition.
- Class Version Number: This entry specifies the version number of a specific class type as per the DLMS specification.
- OBIS Code: This entry is the six character code for the parameter, which is referred in this row. The OBIS codes for most of the common metering parameters are defined in the DLMS Blue Book 11th Edition (section 6).
- Number of Attributes: This entry indicates the number of attributes available for the parameter of this specific class.
- Attribute Table Pointer: Each parameter in the Object List Table has its own attribute list table. The attribute list table is further defined in Section 6. This entry gives a pointer to the attribute list table for the entry.
- Number of Methods: This entry indicates the number of methods implemented for the parameters of this specific class.
- Method Table Pointer: Each parameter in the Object List Table has its own method list table. The method list table is very similar to attribute list table and is further defined in Section 6. This entry gives a pointer to the method list table for the entry.

6 Attribute List Tables

After adding the entry in the Object List Table, the next step is to build an attribute table (see Figure 2). The attribute table has the following columns and the number of entries depends upon the type of class to which the parameter in question belongs to.

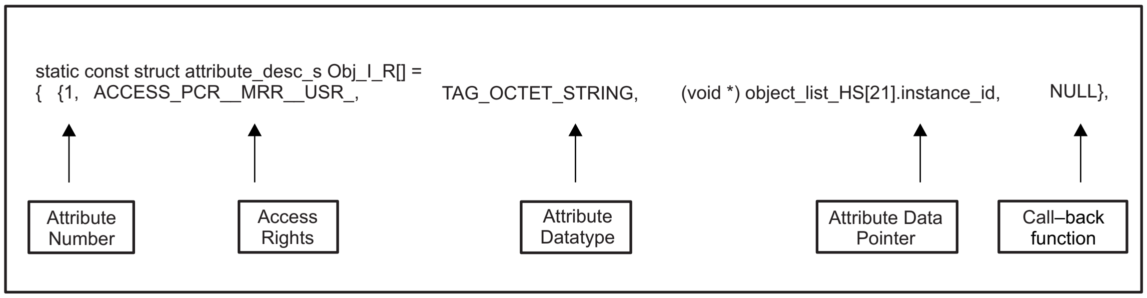
static const struct attribute_desc_s Obj_I_R[] =
{ {1, ACCESS_PCR__MRR__USR_, TAG_OCTET_STRING, (void *) object_list_HS[21].instance_id, NULL},
Attribute Number
Access Rights
Attribute Datatype
Attribute Data Pointer
Call–back function

Figure 2. Attribute List Table

- Attribute Number: The first entry in this table indicates the serial number of the attribute of this class.
- Access Rights: The second column provides the access rights for this attribute depending upon the type of association that is established between DLMS server and client. The syntax for providing the access rights is “PCXY_MRXY_USXY”, where PC, MR, US correspond to the type of association X is R if a read access is available and _ if no read access is available. Similarly, Y is W if a write access is available and _ if no write access is available.
- Data Type: This column defines a TAG for the data type to which this parameter belongs. The DLMS documentation (Blue Book, section 4.1.5) specifies a TAG for each data type. There are predefined macros available for each data type.
- Data Pointer: This value gives the pointer to the location where this value is stored.
- Call Back Function: If this value is not readily available in the RAM, then the library calls this call-back function that will execute some C code, which will fill up the actual value at the location specified in column 4.

7 Handling of Complex Data Types Like Structures and Arrays Through ASN.1BER Encoding

The DLMS objects require a special encoding of data, which is called ASN.1BER coding. The format of ASN.1BER coding looks like this:

```
Tag_Int, Value
```

In case of character strings:

```
Tag_Octet_String, Length, char1, char2, char3……etc.
```

In case of arrays and structures:

```
Tag_Structure,
       Nb_of_Elements,
              Tag_Element1, Value,
              Tag_Element2, Value
```

Whenever, data is provided as constant, it has to be formatted in the same form as above. There is no need to put the starting tag as it is taken from the Attribute List Table. But instead, you need to put the total size of the definition string.

8 The Call Back Function

The call back function is needed when the MCU doesn’t have the data requested by the DLMS client in its RAM and this data has to be fetched from an external EEPROM or some other controller. If there is a call-back function written in the Attribute-List Table then the library will call this function first before returning the data pointed by the Data Pointer in the table. The function has to be defined in the following way:

*void function_name(void *data, int direction)*

where, data is the pointer to the location where data has to be put in case of read and from where data has to be taken in case of write.

Direction is 0 if data has to be read and 1 if data has to be written. In case of multiple entry objects like profile generic buffers, one needs to give further information in the call-back function. This will be explained in Section 9.

9 Handling of Multiple Entries in a Profile Generic Class Buffer

In a Profile Generic Class, the ‘Buffer’ attribute can have a very big data, which can be in the form of multiple entries in the meter. An example of this is the last six month billing history. In this case, there will be a number of parameters that the meter has to record for each month. This DLMS library supports a unique way of handling this data based upon templates. As mentioned earlier in the document, the data in DLMS is encoded in ASN.1, which adds certain overhead in the actual data. This extra data remains the same for multiple entries of the record. Therefore, this data is not needed to be stored in the Flash / EEEPROM multiple times. One can define a template for an entry where you put this redundant data like ASN.1 tags.

As an example, assume that one has to record the following data in your meter after every 15 minutes.

Parameter	Datatype
Date and Time	String
Current – R Phase	Unsigned Long
Current – Y Phase	Unsigned Long
Current – B Phase	Unsigned Long
Voltage – R Phase	Unsigned Integer
Voltage – Y Phase	Unsigned Integer
Voltage – B Phase	Unsigned Integer
Active Energy	Unsigned Long
Reactive Energy - Lag	Unsigned Long
Reactive Energy – Lead	Unsigned Long
Apparent Energy	Unsigned Long

Assume that all currents are 10 Amps, Voltages 240 V and Energies as 100 KWh, then a single entry has to be encoded in ASN.1 as follows:

```
    TAG_STRUCTURE, 0x0b,
      TAG_OCTET_STRING, 12, ITEM_TAG_DATETIME_LP, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,       /* Time
& Date */
      TAG_UINT32, 0x00,0x00,0x03,0xe8            /* Ir =10.00 A*/
      TAG_UINT32, 0x00,0x00,0x03,0xe8            /* Iy =10.00 A*/
      TAG_UINT32, 0x00,0x00,0x03,0xe8            /* Ib =10.00A */
      TAG_UINT16, 0x09,0x60,                     /* Vr=240.0V */
      TAG_UINT16 ,0x09,0x60,                     /* Vy=240.0V */
      TAG_UINT16, 0x09,0x60,                     /* Vb=240.0V */
      TAG_UINT32, 0x00,0x00,0x27,0x10            /* Active Energy =100.00KWh*/
      TAG_UINT320x00,0x00,0x27,0x10              /* Reactive Energy – Lag=100.00KVarh */
      TAG_UINT32, 0x00,0x00,0x27,0x10            /* Reactive Energy – Lead=100.00KVarh */
      TAG_UINT32, 0x00,0x00,0x27,0x10            /* Apparent Energy = 100KVAh*/
```

The ASN.1 encoding above indicates that it is a structure with 11 elements. The first element is an octet string with 12 characters, the second element is a 32-bit unsigned long and so on. When the above data is recorded after every 15 minutes for 10 days, there will be 960 entries of the data exemplified above. For compacting the space required for such data, and for handling multiple entries easily, a template-based mechanism was put into place. The following is an example of the template for the above data:

Example 1. Example of a Template

```
const uint8_t Load_Profile_Buffer_Template[] =
{
    STUFF_DATA | TAG_STRUCTURE, 11,
      STUFF_DATA | TAG_OCTET_STRING, 12, ITEM_TAG_DATETIME_LP, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /*
Time & Date */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_IR_LP),                       /* Ir */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_IY_LP),                       /* Iy */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_IB_LP),                       /* Ib */
      STUFF_DATA | TAG_UINT16, INJECT16 (ITEM_TAG_VR_LP),                       /* Vr */
      STUFF_DATA | TAG_UINT16, INJECT16 (ITEM_TAG_VY_LP),                       /* Vy */
      STUFF_DATA | TAG_UINT16, INJECT16 (ITEM_TAG_VB_LP),                       /* Vb */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_CUM_KWH_TOTAL_LP),            /* Active Energy */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_CUM_KVAR_LAGH_TOTAL_LP),      /* Reactive Energy -
 Lag */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_CUM_KVAR_LEADH_TOTAL_LP),     /* Reactive Energy -
 Lead */
      STUFF_DATA | TAG_UINT32, INJECT32 (ITEM_TAG_CUM_KVAH_TOTAL_LP),           /* Apparent Energy */
};
```

In the above template, “INJECT32” is a macro, which converts a single long number into byte-wise characters in the same endianness as per DLMS requirement. In place of the actual data, a user defined lable is input (ITEM_TAG_DATETIME above). This label is used as a pointer for the function Get_Numeric_Item, Get_String_Item, Set_Numeric_Item & Set_String_Item (defined in file app_server_msgs.c), which is explained in the subsequent section. Each separate parameter has to have a unique label. Once the template is defined, the next step is to write the call-back function. When the call back function is called from the library, it carries the “Selective Access” information in case the DLMS client wants to read the selective data only. The “Selective Access” is not supported in case of the SET command. This data is passed in the following library variables.

In case of Selective access by entry:

- SA_From_Entry: This carries the number of the first entry, which the client wants to read.
- SA_To_Entry: This carries the number of the last entry.

In case of selective access by range:

- SA_Range[0] : This will have the date and time of the first entry, which the client wants to read.
- SA_Range[1] : This will have the date and time of the last entry.

Based on the above selective access information and the predefined template, you need to initialize the following msg_info data structure to pass the relevant information to the library. The following structure is defined in the library, to which, you have to pass the values.

Example 2. The Msg_Info Data Structure

```
typedef struct
{
  const uint8_t *template;
  uint16_t sz_template;
  uint16_t start_entry;
  uint16_t num_entries;
  uint16_t entries_remaining;
  const uint16_t *column_szs;
} Msg_Info;
```

- msg_info.template : This element points to the template defined for that particular object.
- msg_info.sz_template: This is the size of the template that is explained earlier.
- msg_info.start_entry: As there can be a number of entries that have to be transferred, this value indicates the number of the entry from where to start.
- msg_info.num_entries: This element has to be filled with the total number of entries that has to be transferred.
- msg_info.entries_remaining: This element is automatically updated in the library, which tells how many entries data has been transferred.
- msg_info.column_szs: This is an array of cumulative count of bytes in each column of the template. This parameter is used by the library in case the client needs to apply selective access.

Once you have completed the above steps, then the last step is to insert some code in the function get_numeric_item(), get_string_item(), set_numeric_item() or set_string_item() in the file app_server_msgs.c.

These functions are called by the library when it is decoding the profile template and encounters STUFF_DATA tag. In case of GET command, depending upon the data type of the parameter, either of get_numeric item() or get_string_item() are called with the corresponding item tag as argument. In these functions, you have to add a switch case statement and provide the required data into the val variable.

The example below explains this in a better way.

```
int32_t get_numeric_item(int item)
{
    int64_t val;
    switch (item)
    {
        ___________________
        ____________________
        case ITEM_TAG_IR:
               entry_no = msg_info.start_entry+(msg_info.num_entries-msg_info.entries_remaining);
               val=read_EEPROM_param(Load_Param,entry_no,IR);
               break;
      ____________________
      ____________________
     }
}
```

In case of the SET command, again depending upon the data type of the parameter, either of set_numeric_item() or set_string_item will be called with the data pointer from where data have to be copied as an argument. The example below explains this in a better way.

```
void set_numeric_item(uint8_t item, uint8_t *data, uint16_t len)
{
    switch(item)
  {
    ________________________
    ________________________
    case ITEM_TAG_DAY_PROFILE_SCRIPT_ID1:
    entry_no = msg_info.start_entry+(msg_info.num_entries-msg_info.entries_remaining);
        EEPROM_Write(ADDRESS_DAY_ID+entry_no,data)
    break;
  __________________________
  __________________________
  }
}
```

10 Changing the PDU Size

If there is a huge data to be transferred from the meter (server) to the client, then this data is broken into small PDUs and then sent by a GET_NEXT_BLOCK mechanism. It is up to you to define the size of this PDU. There is an environment variable defined by name CHUNK_SIZE, which can be used to change the PDU size.

11 Methods List Tables and Method Description

The method list table is similar to the attribute list table. Figure 3 shows the meanings of different columns in method list table.

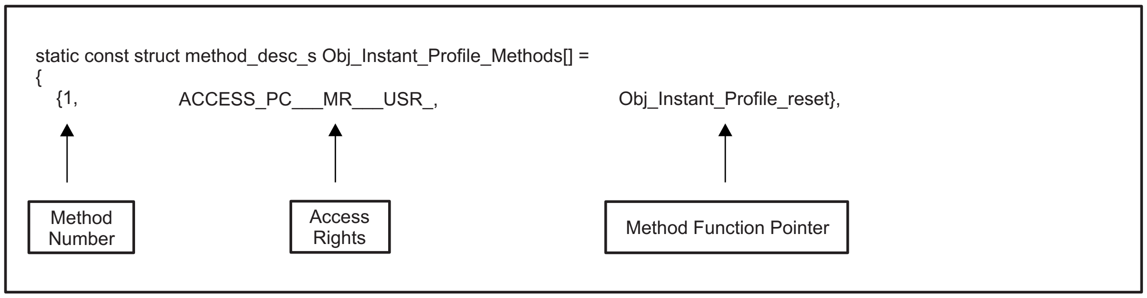
```
static const struct method_desc_s Obj_Instant_Profile_Methods[] =
{
   {1,            ACCESS_PC___MR___USR_,               Obj_Instant_Profile_reset},
```

Figure 3. Method List Table

- Method Number: The first entry in this table indicates the serial number of the method of this class.
- Access Rights: The second column provides the access rights for this method depending upon the type of association that is established between DLMS server and client. The syntax for providing the

access rights is “PCXY_MRXY_USXY”, where PC, MR, US correspond to the type of association X is R if a read access is available and _ if no read access is available. Similarly, Y is W if a write access is available and _ if no write access is available.

- Method Function Pointer: This value gives a pointer to the C function that you need to execute in the meter software. The functions have to be written with predefined arguments. Below is the syntax for the function:

```
void Obj_Instant_Profile_reset(uint8_t *data,uint16_t data_len,uint8_t
*response,uint16_t *response_len)
{
// Your code comes here.
}
```

In the above function definition, *data* is the pointer to the data (if any) sent by the DLMS Client as argument for this method, *data_len* carries the length of this data, response is the data that is to be returned to the client and *response_len* is the length of this returned data.

12 Handling Associations

The DLMS library is configured to handle three associations. You do not have access to change the properties of these associations, except the passwords. Table 1 describes the properties of these associations:

Table 1. Handling Associations

Feature	Public Client	Meter Reader	Utility Setting
SAP Address pair in format (Client,server)	(16,1)	(32,1)	(48,1)
Application Context – Basic Security	LN without ciphering	LN without ciphering	LN without ciphering
Application Context- Advanced Security	Not Applicable	LN - Ciphered	LN - Ciphered
Signon Authentication Mechanism – IEC 62056-53, Clause 7.3.7.2	Lowest Level	Low Level (LLS)	High Level (HLS)
Services in Conformance Block	Get, Get with Block transfer	Get, Get with Block transfer, Selective Access	Get, Set,Action, Get and Set with Block transfer, Selective Access
OBIS Codes	0.0.40.0.1.255	0.0.40.0.2.255	0.0.40.0.3.255

13 Adding a New Class

Adding a new class in the library is a simple process. You need to add an entry in the Object List table with the new Class ID, give all the details and create the Attribute list and method list table. You have to strictly follow the Class details given in the Blue Book.

14 Reference

DLMS UA 1000-1 ed.11, 2013 Blue book, COSEM Identification System and Interface Classes

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		