**DLMS Client Module Design**

This document lists the design of DLMS Client Module. The DLMS protocol related functions that accesses data from the meters are developed as a set of library functions. The document also lists the set of API functions of the library which the high level applications can call. This is the guideline for the development of both the DLMS library and the DLMS Client application. The data acquired is stored in redis server so that depending upon the requirements the data is packaged in various forms like csv files, cdf files, post to certain urls and sent to the processing software

# DLMS Module Architecture

Meter Data Files

**Remote Monitoring Software**

Redis Database

Serial/Ethernet Ports

DLMS APIs

**DLMS Library**

DLMS Meter

DLMS Meter

DLMS Meter

# API List

This document lists the APIs to be implemented by the DLMS library Module which can be accessed by the calling applications. The APIs to be developed are common to access data from both the serial DLMS meters and ethernet DLMS meters. The high level calling functions need to pass the required parameters to access the appropriate meters. The DLMS library will be developed in c language

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API Name** | **Input Parameters** | **Output Parameters** | **Return Value** | **Description** |
| **init\_comm** | Structure containing type of communication and the required parameters. Type identifying serial or ethernet | Serial fd or socket fd | 0 or appropriate error code | This API initializes the communication channel of DLMS devices. If the type is serial, initializes the serial port with the supplied parameters , if ethernet, initializes the tcp connection with the meter |
| **connect\_to\_meter** | Structure containing Commn fd, meter id , password , type of meter | Response packet | 0 or appropriate error code | Sends SNRM and AARQ packets to the meter and get the response |
| **get\_nameplate\_details** | Structure containing  Commn fd, meter id , | Response structure with all the nameplate details | 0 or appropriate error code | Get the nameplate information of the meter and return the decoded nameplate values in appropriate structure |
| **get\_obis\_codes** | Structure containing  Commn fd, meter id | Response structure | 0 or appropriate error code | This function does the following tasks   * gets values obis code of all the types of data . * get the scalar obis codes of all the types of data. * get the scalar values for each of the parameter of each type of data |
| **get\_inst\_values** | Structure containing  Commn fd, meter id , | Response structure with all the details | 0 or appropriate error code | Get the instantaneous data of the meter and return the decoded values of instantaneous data in appropriate structure |
| **get\_ls\_values\_block\_range** | Structure containing  Commn fd,meter id,timefrom , timeto | Response structure with all the details | 0 or appropriate error code | Get the load survey data of the meter for the given block range. The range should be such that it contains data of fewer blocks say maximum of 4 blocks. The decoded values of the blocks are returned in appropriate structure |
| **get\_ls\_values\_hour\_range** | Structure containing Commn fd,meter id,timefrom , timeto | Response structure with all the details | 0 or appropriate error code | Get the load survey data of the meter for the given range in hours. The range can be upto 24 hours. The decoded values are stored in the files for that day. Only the last block data is returned to the calling function |
| **get\_ls\_values\_day\_range** | Structure containing Commn fd,meter id,datefrom , dateto | Response structure with all the details | 0 or appropriate error code | Get the load survey data of the meter for the given range in days. The range can be upto 1 month. The decoded values are stored in the files for that day. Only the last block data is returned to the calling function |
| **get\_midnight\_data\_all** | Structure containing Commn fd,meter id | Response structure with all the details | 0 or appropriate error code | Get all the midnight data available in the meter. The decoded values are stored in files with the month as the name. Only the last midnight data is returned to the calling function |
| **get\_midnight\_data** | Structure containing Commn fd,meter id | Response structure with all the details | 0 or appropriate error code | Get the midnight data for the current month from the meter. The decoded values are stored in files with the month as the name. Only the last midnight data is returned to the calling function |
| **get\_event\_data\_all** | Structure containing Commn fd, meter id, event class | Returns the details of last event of the class | 0 or appropriate error code | Get all the events stored in the meter for the event class. The events read are stored in appropriate file in the standard format. Only the last event of the class is returned to the calling function |
| **get\_event\_data** | Structure containing Commn fd, meter id, event class | Returns the details of last event of the class | 0 or appropriate error code | Get the latest maximum 10 events stored in the meter for the event class. The events read are stored in appropriate file in the standard format. Only the last event of the class is returned to the calling function |
| **get\_billing\_info** | Structure containing Commn fd, meter id, | Returns the details of last bill info | 0 or appropriate error code | Get the billing information of the last month. Only the last information is returned to the calling function |
| **disconnect\_meter** | Structure containing Commn fd,meter id |  | 0 or appropriate error code | Disconnects the meter and closes the communication |
|  |  |  |  |  |

# Errors List

|  |  |  |
| --- | --- | --- |
| **Error Code** | **Error** | **Description** |
| 0 | RET\_SUCCESS | Function completed successfully |
| -1 | RET\_NO\_RESP | Meter didn’t respond |
| -2 | RET\_INCOMP\_RESP | Incomplete response received |
| -3 | RET\_CRC\_FAIL | CRC failed |
| -4 | RET\_AUTH\_FAIL | Authentication failed |
| -5 | RET\_PORT\_OPEN\_FAIL | Failed to open serial port |
| -6 | RET\_ETH\_CONN\_FAIL | Failed to connect to ethernet meter |
| -7 | RET\_DATA\_NOT\_AVAIL | Data not available for the interval requested |
| -8 | RET\_VAL\_OBIS\_NP\_FAIL | Failed to get obis code for nameplate values |
| -9 | RET\_VAL\_OBIS\_INST\_FAIL | Failed to get obis code for instantaneous values |
| -10 | RET\_VAL\_OBIS\_LS\_FAIL | Failed to get obis code for load survey values |
| -11 | RET\_VAL\_OBIS\_MN\_FAIL | Failed to get obis code for midnight data values |
| -12 | RET\_VAL\_OBIS\_BILL\_FAIL | Failed to get obis code for billing info values |
| -13 | RET\_VAL\_OBIS\_EVENT\_FAIL | Failed get obis code for event data values |
| -14 | RET\_SCAL\_OBIS\_NP\_FAIL | Failed to get obis code for nameplate scalar values |
| -15 | RET\_SCAL\_OBIS\_INST\_FAIL | Failed to get obis code for instantaneous scalar values |
| -16 | RET\_SCAL\_OBIS\_LS\_FAIL | Failed to get obis code for load survey scalar values |
| -17 | RET\_SCAL\_OBIS\_MN\_FAIL | Failed to get obis code for midnight data scalar values |
| -18 | RET\_SCAL\_OBIS\_BILL\_FAIL | Failed to get obis code for billing info scalar values |
| -19 | RET\_SCAL\_OBIS\_EVENT\_FAIL | Failed get obis code for event data scalar values |
| -20 | RET\_SCAL\_VAL\_INST\_FAIL | Failed to get the scalar values for instantaneous parameters |
| -21 | RET\_SCAL\_VAL\_LS\_FAIL | Failed to get the scalar values for load survey parameters |
| -22 | RET\_SCAL\_VAL\_MN\_FAIL | Failed to get the scalar values for midnight data parameters |
| -23 | RET\_SCAL\_VAL\_BILL\_FAIL | Failed to get the scalar values for billing data parameters |
| -24 | RET\_SCAL\_VAL\_EVENT\_FAIL | Failed to get the scalar values for event data parameters |

# Data Structures

## Common API structure

This is the common structure which is passed from the calling module to the library. This has all the required information for the library to poll the meter. The response is also filled in the appropriate field of the data structure as output parameters

typedef struct

{

char ser\_port[32];

uint8\_t baudrate; // (index)Baudrate of serial port

uint8\_t databits; // Num of databits

uint8\_t stopbits; // Num of stop bits

uint8\_t parity ; // Parity - ODD/EVEN/NONE

uint8\_t handshake; // Handshake - SW/HW/NONE

uint8\_t infmode; // RS232 /485

uint8\_t unused;

}serport\_params\_t;

typedef struct

{

char meter\_ipaddr[32];

uint32\_t port;

}eth\_params\_t;

typedef struct

{

uint8\_t day;

uint8\_t mon;

uint8\_t year ;

uint8\_t hour;

uint8\_t min;

uint8\_t second;

}date\_time\_t;

typedef struct

{

int32\_t fd;

int8\_t inf\_type;

int8\_t meter\_type;

int32\_t meter\_id;

int8\_t meter\_addr\_format;

data\_time\_t from;

date\_time\_t to;

int32\_t inf\_params\_len;

char \*interface\_params;

int\_32 other\_info1;

int\_32 other\_info2;

char filename[SIZE\_128];//file where the big data has to be stored

int\_32 resp\_len;

char \*meter\_response

}meter\_comm\_params\_t

typedef struct

{

Param list

}nameplate\_t

# DLMS Meter Polling

This section explains the DLMS Client Meter Polling process. The module calls the APIs of the library to start the polling process and collect data. The interface through which the meters are connected can be either through serial port or through ethernet. The process polls the meter data by calling library calls and stores the response in redis server database.

## Algorithm

* Read the configuration
* Fill the communication structure with the interface type and other details
* Initialize the communication interface by calling the function **init\_comm**
* For each meter configured
  + Fill in the meter details
  + Call **connect\_to\_meter** function, handle the return value accordingly
  + Get the obis codes and scalar values by calling the function **get\_obis\_codes**
  + Get nameplate details by calling **get\_nameplate\_details** and hadle the return value accordingly
  + The nameplate details are available in the meter\_response output parameter
  + Store the nameplate details in appropriate redis server data structure
  + Get instantaneous values by calling the function **get\_inst\_values**. If the function is successful, the data is returned in the output parameter.
  + Store the instantaneous values in appropriate redis server data structure
  + Get the load survey data of past 45/30 days if not available by calling the function **get\_ls\_values\_day\_range**  If the function is successful, the data is stored in respective date files in the fixed directory
  + Update the last 4 blocks data in appropriate redis server data structure
  + Get the midnight data of the all available data by calling **get\_midnight\_data\_all**. If the function is successful, the data is stored in respective month’s file in fixed directory
  + Store the data in appropriate redis server data structure
  + Get the events of all the classes by calling **get\_event\_data\_all.**  If the function is successful, the data is stored in respective file in fixed directory
  + Get the billing information by calling **get\_billing\_info.**  If the function is successful, the data is stored in respective file in fixed directory
  + Depending upon the response, the respective functions can be retried
* For each meter configured
  + If it is the time to poll instantaneous values, get instantaneous values by calling the function **get\_inst\_values**. If the function is successful, the data is returned in the output parameter.Store the instantaneous values in appropriate redis server data structure
  + If it is time to poll load survey values, call **get\_ls\_values\_block\_range,** If the function is successful, the data is stored in respective date files in the fixed directory, Update the last 4 blocks data in appropriate redis server data structure
  + If it is time to poll midnight data, call **get\_midnight\_data** , If the function is successful, the data is stored in respective month’s file in fixed directory Store the data in appropriate redis server data structure
  + If it is time to get event data, call **get\_event\_data** for each event class . If the function is successful, the data is stored in respective file in fixed directory. Store the data in appropriate redis server data structure
  + If it is time to get billing data, call **get\_billing\_info**. If the function is successful, the data is stored in respective file in fixed directory. Store the data in appropriate redis server data structure
* Periodically update the status of communication of the meters in appropriate redis server data structure
* Update the redis server data structures of all the data types if not already updated by reading data from the files stored by the library
* Periodically cleanup the old files
* Update the log files for each operation
* If network debug logging is enables, send out the debug log messages over network to the configured debug logger

# Redis data structures

The hash to store the nameplate details of a meter

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **np\_details\_metserno** | Meter\_serial\_num |  |  |
| Meter\_id |  |  |
| Ct\_ratio |  |  |
| Pt\_ratio |  |  |
| Manufact\_det |  |  |
|  |  |  |
| updatetime |  |  |

The hash to store the instantaneous data of a meter

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **Inst\_data\_metserno** | Meter\_serial\_num |  |  |
| Meter\_id |  |  |
| Volt1 |  |  |
| Volt2 |  |  |
| Volt3 |  |  |
| Cur1 |  |  |
| Cur2 |  |  |
| Cur3 |  |  |
| pf |  |  |
|  |  |  |
|  |  |  |
| updatetime |  |  |

The hash to store the load survey data of a meter

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **ls\_data\_metserno** | Meter\_serial\_num |  |  |
| Meter\_id |  |  |
| Volt1 |  |  |
| Volt2 |  |  |
| Volt3 |  |  |
| Cur1 |  |  |
| Cur2 |  |  |
| Cur3 |  |  |
| pf |  |  |
|  |  |  |
|  |  |  |
| updatetime |  |  |

The hash to store the midnight data of a meter

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **midnight\_data\_metserno** | Meter\_serial\_num |  |  |
| Meter\_id |  |  |
| Volt1 |  |  |
| Volt2 |  |  |
| Volt3 |  |  |
| Cur1 |  |  |
| Cur2 |  |  |
| Cur3 |  |  |
| pf |  |  |
|  |  |  |
|  |  |  |
| updatetime |  |  |

The hash to store the event data of a meter

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **Event\_classno\_metserno** | Meter\_serial\_num |  |  |
| Meter\_id |  |  |
| Event code |  |  |
| Num\_events |  |  |
| Event\_class |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| updatetime |  |  |

The hash to store the billing info data of a meter

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **billinginfo\_metserno** | Meter\_serial\_num |  |  |
| Meter\_id |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| updatetime |  |  |

The hash to store the status of meters

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Field** | **Value** | **Comments** |
| **Meter\_status** | Num\_meters |  |  |
| Meter\_id |  |  |
| Status |  |  |
| Meter\_id |  |  |
| Status |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| updatetime |  |  |