



DLMS/COSEM Server Source Code Library
User Manual

SCL Revision Summary

SCL Version	Release Date
2.2.2.2	19-Mar-09

Table of Contents

Table of Contents.....	2
INTRODUCTION.....	3
OVERVIEW.....	3
IMPLEMENTATION OVERVIEW.....	3
IMPLEMENTATION PROCEDURE.....	5
1. EDIT THE KDEFS.H FILE.....	5
2. BUILD THE SCL ON YOUR TARGET PLATFORM.....	5
3. IMPLEMENT PLATFORM INTERFACE.....	6
Serial Port.....	7
Ethernet Port.....	7
4. TEST PLATFORM IMPLEMENTATION.....	8
5. EDIT CONFIGURATION INTERFACE.....	8
Protocol related information.....	12
6. TEST CONFIGURATION.....	14
7. IMPLEMENT DATA INTERFACE.....	14
8. TEST COMPLETE SERVER IMPLEMENTATION.....	16

Introduction

Overview

The DLMS/COSEM Server SCL provides a complete implementation of a 62056 COSEM/HDLC protocol stack to enable Meter-manufacturers and other OEMs to DLMS-enable their 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 protocol source is written in ANSI-C and can be built onto most platforms as-is without any changes. For the stack to function however, three specific interfaces are required to be implemented from the OEM's point of view

1. Platform Interface

The platform interface is used by the SCL to perform functions that can be only be provided in a platform-specific manner. These functions include millisecond timers and communication port handling.

The SCL at runtime makes calls to these functions and expects the function to perform the documented purpose of the function. The OEM must implement these functions

2. Configuration interface

There is a vast amount of configuration data that is required by the SCL. This includes

- Complete master-list of data to be served by the meter (with their OBIS codes, Short-Names, Interface-Classes etc.)
- Division of access to these data-objects into multiple association-views. The same data-object may be accessed under different association-views with different access-privileges
- Configuration of the associations
- Static-information about the data-objects (information that does not change at runtime, for eg. The Scaler and Unit of a Register object)
- Capture-object lists for Profile objects
- Inter-layer buffer-sizes
- More

This information is configured into the SCL by editing two header files (obis.h and config.h). Note that this information is specified at compile-time.

3. Data interface

The SCL (when built with the platform and configuration interfaces implemented) will already run and serve data to a DLMS/Client. Notably it can already serve all the static-information that is available for each object in the meter. However

when requested for dynamic-values (for eg. The value attribute of a Register object) it will return zeros or invalid data.

To provide the actual data from the meter or device, the OEM must implement the functions in the Data-interface

The SCL ships with a default Configuration containing many pre-configured objects from all supported Interface-Classes and with a number of association-views that exercise the LN and SN referencing with different levels of security. The OEM can use this sample configuration as a base for further development

The SCL also ships with a Windows implementation of the full SCL. This sample project also implements the platform interface, providing Windows-specific implementations of the millisecond-timer and COM port routines. The OEM can build this Windows workspace and start running the server on a Windows PC and test against a client to trace the flow and logic of the protocol-stack.

Preparation for Integration

At this stage OEM must ensure that they are ready for the integration exercise by confirming the below checklist

1. Build environment

Ensure that the proper cross-compiler is available for building the sources for the specific platform. Availability of a good debugger is recommended

2. Meter data

OEM must build a master-list of meter-data and map these to the standard DLMS Interface classes and OBIS codes. Examine the header file obis.h closely. The list_ALLOBIS[] array contains the default configuration and models various objects of several Interface Classes.

For example Energy values can be modeled by a Data (IC 1), Register (IC 3) or an ExtendedRegister (IC 4) class object. If you only require the value you can use a Data object. If you also want to model the scaler and unit of the value, you must use a Register object. Additionally, if you require to store the timestamp of the Energy-Value you can use an ExtendedRegister object. A full description of the Interface classes is beyond the scope of this document. Please refer to IEC-62056-62 for further details.

OBIS codes for each object follow a standard specification as described in IEC-62056-61. Note that the SCL code does not verify the OBIS codes against the standard and accepts all configured OBIS. If you are planning to use SN (ShortName) referencing, you must also map each object to a short Base-Name (2 bytes) which will be the attribute address of the first attribute of the object. The SCL will calculate the base-names of all other attributes of the object automatically.

3. DLMS Protocol features

- a. OEM must decide if they want to use LN (Logical Name) or SN (Short Name) referencing or both
- b. OEM must decide which subset of the Interface Classes they want to support. For example if the master-list of meter data objects does not contain any ExtendedRegister objects, they can exclude this Interface Class completely. This can save considerable code-space.

Implementation Procedure

1. Edit the kdefs.h File

You must edit the kdefs.h file, specifically the definitions of the primitive types to match the types available in your platform. The default definitions should suffice for most implementations

2. Build the SCL on your target platform

The first main activity is to build the SCL on your specific target-platform. To do this you must write a main() method (task entry point) and call the following functions as below

```
#include "..\KalkiSources\config.h"
#include "..\KalkiSources\cosem.h"
#include "..\KalkiSources\plaf.h"
#include "..\KalkiSources\hdlc.h"
#include "..\KalkiSources\wrapper.h"

extern KDEFS_UCHAR RECEIVE_PORT_TYPE;

/* Entry point function */
int main(int argc, char* argv[])
{

    #if COM_PROFILE != HDLC
        wrapper_Init();
    #endif
    #if COM_PROFILE != TCP_UDP
        hdlc_Init();
    #endif
    cosem_Init();
    while(1)
    {
        #if COM_PROFILE == DYNAMIC
            if((RECEIVE_PORT_TYPE == COM_RS232) ||
(RECEIVE_PORT_TYPE == COM_OPTICAL))hdlc_Process();
            else if(RECEIVE_PORT_TYPE == COM_IP)wrapper_Process();
            else
            {
                if(RECEIVE_PORT_TYPE ==
COM_INACTIVE)hdlc_Process();
                if(RECEIVE_PORT_TYPE ==
COM_INACTIVE)wrapper_Process();
            }
        #endif
        #if COM_PROFILE == HDLC
            hdlc_Process();
        #endif
        #if COM_PROFILE == TCP_UDP
            wrapper_Process();
        #endif
        cosem_Process();
    }
    return 0;
}
```

The source-files and header files are in the “KalkiSources” folder in your package. You must set the include-path and source-path to point to this folder.

The purpose of this build-step is to isolate any platform-specific code issues in the core SCL. Please inform pdssupport@kalkitech.in if you have any issues at this stage.

3. Implement Platform Interface

The next step is to implement the functions in the Platform interface. The functions are declared in `plaf.h` and defined with empty bodies in `plaf.c`

Note that the function definitions contain default implementations for Windows(bracketed in `#ifdef WIN32` and `#endif` statements) and Linux(bracketed in `#ifdef LINUX` and `#endif` statements) platforms. You can use the sample Windows/Linux implementations to help understand what is required in each function

- 1. void set_CommBaud(KDEFS_UCHAR baud)**
This function is called by the SCL when it requires to change the baud rate of Serial Communication Interface. It sends as argument the new baud rate. The OEM must change baud rate to new value and return.
- 2. void set_CommParityByteSize(KDEFS_UCHAR parity, KDEFS_UCHAR Data_Bits)**
This function is called by the SCL when it requires to change the parity and data bits(Byte Size) of Serial Communication Interface. It sends as argument the new parity and new data bits. OEM must change parity and data bits and return.
- 3. KDEFS_BOOL plaf_tcpAccept(void)**
This function should accept any connect request from peer TCP socket and return the result(SUCCESS/FAILURE).
- 4. void plaf_close_eth_port()**
Platform interface should gracefully release `tcp_socket`(connected) and disable receiving of data from the closed socket.
- 5. void plaf_Transmit(KDEFS_UCHAR* buffer, KDEFS_USHORT len)**
This function is called by the SCL when it requires to transmit a response to the client. It sends as arguments a pointer to the buffer containing the response and a length variable containing the number of bytes to be transmitted.
- 6. void plaf_GetMSCounter(void* timer)**
This function is expected to write a millisecond counter value to the supplied pointer. The SCL allocates `SIZEOF_TIMER_VALUE` number of bytes to this pointer before making the call.
If the OEM wishes to use a time-structure that require 20 bytes, OEM must configure `SIZEOF_TIMER_VALUE` to 20. The Windows implementation writes a `time_t` value to this pointer and configures `SIZEOF_TIMER_VALUE` to 10 bytes. The OEM must not attempt to free this memory in any way. The SCL does not attempt to process this value in any way and is not concerned with the actual type or structure that is written into this pointer.
- 7. KDEFS_BOOL plaf_ElapsedMSCounter(void* initialVal, KDEFS_ULONG timeout)**

This function is called by the SCL to check for expiry of timeouts. It sends as an argument the same timer-value pointer that was filled-in in the previous function and also sends a timeout value

The OEM code must compare the present timer-value against the initial value and check if the supplied timeout has expired. It must return a KDEFS_BOOL value of KDEFS_TRUE if the timeout has expired. The OEM code must handle timer-rollovers correctly.

8. void plaf_Sleep(KDEFS_USHORT millis)

OEM must implement code for the program to sleep for specified period of time, which is passed as function argument in milli seconds.

In addition to these functions, the OEM code must perform the following functions.

9. Initialize Comm Ports

It must initialize all communication ports with the desired settings. This must be done as an initialization procedure from the main function before anything else.

This includes the following functions

- SCI_Init
- TCPUDP_Init

10. Read Ports and store received bytes

It must provide a means of reading the initialized ports and storing the received bytes to appropriate receive_buffer(hdlc_InBuf/wInBuf). The Windows and Linux implementation uses separate thread to monitor the port and store received data. In the OEM implementation, this may not be required to run as a separate thread. Instead the OEM can utilize a port interrupt for received-chars. Received bytes are always written to the SCL buffer one byte at a time and must utilize the circular-buffer handling as described below

Serial Port

```
hdlc_InBuf[hdlc_In] = Byte;
hdlc_IncrIndex(&hdlc_In);
if(hdlc_In == hdlc_Out)
{
    hdlc_IncrIndex(&hdlc_Out);
}
```

Ethernet Port

```
wInBuf[wCosem_In] = Byte;
wIncrIndex(&wCosem_In);
if(wCosem_In == wCosem_Out)
{
    wIncrIndex(&wCosem_Out);
}
```

4. Test Platform implementation

At this stage the SCL-build contains a default configuration and a fully-implemented platform-interface. It is now possible to test the SCL-build with a DLMS client program or DLMS/COSEM OPC server.

The default configuration contains multiple associations including the standard “Public-Client” Association. This association is defined by a Client-ID of 16 (0x10). The Server-ID is set as 1 in the default configuration. If the OEM has not modified the Associations in the default configuration, this Association is set to use Logical Name referencing and uses Lowest level of security (No-password).

Using these details OEM can establish a Link-Layer connection (SNRM/UA exchange for HDLC based profiles or necessary functions to connect IP (TCP/ UDP) stack) and associate to the server (AARQ/AARE exchange). OEM can then proceed to read/write parameters to the server. A useful test at this stage is to read the Object-list of the association. The default configuration configures all objects in the meter with full-access to this association.

Please note however that several attributes of several objects are by default not writable. For example the LN (Logical Name) attribute of every object is not writable. Even though the association grants full read-write access, an attempt to write to this value will return a “READ_WRITE_DENIED” error

5. Edit Configuration Interface

At this stage, the server implementation has been running with a default configuration. OEM must edit the configuration in the header files obis.h and config.h to specify the actual configuration of the meter.

Start with the obis.h header file.

1. list_AllOBIS[]

Master-list of all meter data-objects. Each entry in this list fully qualifies a meter data-object by specifying its OBIS-code, SN, IC number, IC version, number of historical stored values (for past billing-periods) for that object.

This table must be filled in ascending order of OBIS-codes. This is because the SCL uses an optimized binary-search to locate an OBIS-code when it receives a request. Care must also be taken in assigning short names to ensure that there is no overlap across objects. The SN of an object is the SN of its first attribute. SNs of successive attributes are obtained by adding 8 to the previous attributes SN. However the SNs of methods are not always sequential. Please refer to the IEC-62056-61 specification for more details.

Table Index

In addition to this each entry has a tableIndex parameter that identifies the index of a record in another IC-specific table which contains static-information for that object.

For example if a Register (IC 3) object has a tableIndex value of 6, this means that the static information (data-type of the value, length of the value, the value’s scaler and unit) of that Register is specified in the 7th row (array index 6) of the list_Register[] array

Access Privileges

A notable parameter in this master-table is an array of access-privileges to access this object for each configured association. For example if you have configured 6 associations, this array will contain 6 unsigned longs. The first ulong will contain the access-privileges for this object in the first association. A value of zero means that object is not included in the object-list of that association.

Access privileges are bitwise encoded inside the unsigned long.

Access to each attribute of the object is encoded in 2 bits

00 – No access

01 – Read access

10 – Write access

11 – Read-Write access

Access to each method of the object is encoded in 1 bit

0 – No access

1 – Execute access

Accesses to all the attributes and methods of an object are encoded in a single unsigned long. The access to the first attribute is encoded in the least-significant 2 bits, the next attribute in the next 2 bits going upwards. When all attributes are accounted for, method accesses are added as 1 bit each. For example a Register object has 3 attributes and 1 method. If you wish to specify the following access-rights

Attr 1: Logical Name – Read access – 01

Attr 2: Value – Read-Write access – 11

Attr 3: Scaler-Unit – Read access – 01

Meth 1: Reset – Execute access – 1

The access-rights unsigned long will contain the binary value 1011101 = 0x5D

The same object can have different access-rights in different associations. It can even be excluded from an association's object-list by specifying a value of zero for the access-rights

2. **list_Data[]**

This is an IC-specific table that configures the static information for all Data (IC 1) objects in the master-list.

OEM must configure the data-type of the Data object's value attribute and optionally its length (if the data-type is non-primitive, for eg. An Octet-String)

Data-types are specified using the DLMS standard ASN1 notations as listed in IEC-62056-53. The data-types are also defined in datastructs.h with named defines for your convenience.

Each entry in this table is linked to the object in the master-list via the tableIndex parameter in the master-list

3. **list_Register[]**

This is an IC-specific table that configures the static information for all Register (IC 3) objects in the master-list.

The configurable static information for a Register object is similar to the one specified in the Data class above. The additional attribute here is the Scaler and Unit of the Value attribute. OEM must configure the scaler and unit as per the standard values in IEC-62056-62. A value of zero for the scaler means No-Scaling and a value of 255 for the unit means No-Unit (pulse-counts)

Each entry in this table is linked to the object in the master-list via the tableIndex parameter in the master-list

4. **list_XRegister[]**

This is an IC-specific table that configures the static information for all ExtendedRegister (IC 4) objects in the master-list.

The static information for this object includes all the information for the above Register class and additionally also configures the data-type of the status attribute and optionally its length (for non-primitive data-types)

Each entry in this table is linked to the object in the master-list via the tableIndex parameter in the master-list

5. **list_DRegister[]**

This is an IC-specific table that configures the static information for all DemandRegister (IC 5) objects in the master-list.

It has the same static-information as for the ExtendedRegister class above. Note that the data-type and length of the Value attribute applies to both the Current-Average-Value as well as the Previous-Average-Value attributes of the object.

Each entry in this table is linked to the object in the master-list via the tableIndex parameter in the master-list

6. **list_Profile[]**

This is an IC-specific table that configures the static information for all ProfileGeneric (IC 7) objects in the master-list.

A Profile object captures several snapshots of a set of data-objects at various intervals of time. For example a LoadProfile object may capture the values of Active and Reactive energies, say, every 15 minutes. The 15 minutes duration is called the Capture-Period and the set of objects that it captures is called the Capture-Objects. Each profile will have a maximum number of snapshots that it can capture.

Static Information

This table is used to configure the Capture-Objects, capture-period, sort-method, sort-object (if not default sort-method) and max-entries of the Profiles. This list also must contain the encoded-size of each snapshot. The encoded size of each snapshot (called "entry") is the sum of the encoded-sizes of each capture-value (see below) in the capture-objects list

Capture Objects

Capture objects must be configured in a separate table as shown in the default configuration and the address of the cap-objects table must be provided in the profile's static information

OEM must create a separate Capture-Objects table for each Profile object. The default configuration only creates one Capture-Objects table and assigns the same table to all Profiles.

A Capture-Objects table configures the OBIS-code, IC number, attribute and data-indices of each captured object. In addition it also specifies the encoded-size of each value and the ASN1 code for that value.

For example if one of the Capture-Objects is an Energy Register and its value is of data-type long-unsigned. The DLMS type-code for this is 18 and its length is 2 bytes. The value will be sent as 18xxyy which means that the encoded size is 3 bytes and the ASN1 code is just "18". The ASN1 code does not require a length since the data-type is a primitive value.

Consider another example. If the data-type of a capture-object's value is of type Octet-String of length 5 characters, its value will be sent as 0905aabbccdde.

This means the encoded-size of the value is 7 bytes and its ASN1 code is "09,05"

NOTE: The SCL does not actually capture the values or store the profile data in any way. The OEM code inside the meter must capture the data and store the different snapshots of data. If the OEM configures the Profile's sort method, it must perform the sorting inside the meter and return the sorted values when requested through the Data Interface

7. list_Clock[]

This is an IC-specific table that configures the static information for all Clock (IC 8) objects in the master-list.

The only static attribute of a Clock object is its Clock-base attribute. OEM can configure this value from the standard defines in datastructs.h

8. list_ScriptTable[]

This is an IC-specific table that configures the information for all ScriptTable (IC 9) objects in the master-list.

A ScriptTable object is used to model Actions that can be triggered from the client. A ScriptTable contains a number of Scripts. Each script is identified by a script-identifier and contains an array of actions. The actions specify which attribute of which object must be written to and with what data, or which method of which object must be executed and with what data.

The information of a ScriptTable object contains the number of scripts in the object as well as the number of actions in each script. In addition, if the script actions require a parameter (Write data or Execute argument), this table must list the ASN1 code and length of the parameter.

9. list_SpecialDaysTable[]

This is an IC-specific table that configures the information for all SpecialDaysTable (IC 11) objects in the master-list.

A SpecialDaysTable object is used to list all special days that can over-ride the default Tariffication schedules as defined in ActivityCalendar objects. Each entry in this object defines a special day (date) linked to a Tariffication day-profile ID in the ActivityCalendar.

The only information in this object is the number of Special days.

10. list_ActivityCalendar[]

This is an IC-specific table that configures the information for all ActivityCalendar (IC 20) objects in the master-list

An ActivityCalendar object specifies the meter's Tariffication schedule in the forms of Seasons, Weeks and Day profiles.

Each Day Profile defines a number of time-switches which indicate the start-times at which specific scripts are activated. The scripts associated with each time-switch refer to ScriptTable objects and specific script-selectors in those objects.

A Day Profile is identified by a DayID and is linked to one or more Week-profiles. Each Week-profile is linked to one or more Seasons.

The static information of an ActivityCalendar object includes the lengths of the Calendar, Season and week names and the number of Seasons, Weeks, Days and Time-switches in each Day.

11. list_SingleActionSchedule[]

This is an IC-specific table that configures the information for all SingleActionSchedule (IC 22) objects in the master-list
A SingleActionSchedule object is used to trigger specific Scripts (from a ScriptTable object with a script-selector). These scripts typically are not related to Tariffication.
The information of this object contains only the number of execution-times for all the scripts.

12. list_Associations[]

This is an IC-specific table that configures the static information for all Associations (IC 12 as well as IC 15) objects in the master-list.
This table thus includes both LN and SN associations.
The associations are entirely static in nature and include the referencing used (LN or SN), authentication mechanism, the associated Client-ID and Server-ID, the DLMS conformance block (specifies the functions supported in that association) and the DLMS version (always 6, currently)
The number of entries in this table must match the array-length of the access-rights array in the master-list.

Protocol related information

Protocol related interface objects are defined in init function(hdlc_Init() or wrapper_Init ()) of corresponding source files (hdlc.c or wrapper.c).

1. list_IECLocalPortSetup[]

This is an IC specific table that configures information for all IECLocalPortSetup (IC 19) objects in the master list. The information is used during the opening sequence for Mode E(IEC 62056-21) before switching to HDLC.
This table contains all attribute values except logical name, for Interface Class 19, as specified in IEC 62056-62.
This table is initialized in hdlc_Init() function, contained in hdlc.c

2. listIECLocalPortSetup_XtrInfo

This is an IC specific table that holds extra information for all IECLocalPortSetup (IC 19) objects in the master list. The information is used during the opening sequence for Mode E(IEC 62056-21) before switching to HDLC.
This information includes
XXX – manufacturer identification comprising three upper case letters. If the third letter is lower case, the minimum reaction time for the device is 20 milli seconds, instead of normal 200 milli seconds.
Ident – Manufacturer specific, 16 printable characters except '/' and '!'
This table is initialized in hdlc_Init() function, contained in hdlc.c

3. list_IECHDLCSetup[]

This is an IC specific table that configures information for all IECHDLCSetup (IC 23) objects in the master list. In HDLC based communication profile, these values are used for effective data communication between client and server HDLC layers.
This table contains all attribute values except logical name, for Interface Class 23, as specified in IEC 62056-62.
This table is initialized in hdlc_Init() function, contained in hdlc.c

- 4. list_PSTNModemConfiguration[]**

This is an IC-specific table that configures the information for all PSTNModemConfiguration (IC 27) objects in the master-list
A PSTNModemConfiguration object describes the configuration to handle an attached PSTN modem.
The information includes the number of modem initialization strings, the length of the request and response init-strings and the maximum length of the modem-profile as described in IEC-62056-62
- 5. list_PSTNAutoAnswer[]**

This is an IC-specific table that configures the information for all PSTNAutoAnswer (IC 28) objects in the master-list
A PSTNAutoAnswer object defines the configuration of the Auto-answer functionality of an attached PSTN modem.
The information includes the number of configured listening-windows
- 6. list_TcpUdpSetup_Info[]**

This is an IC specific table that configures information for all TcpUdpSetup(IC 41) objects in the master list. In IP based communication profile, this information is used for initialization of IP port, and managing data communication through the same.
This table contains all attribute values except logical name, for Interface Class 41, as specified in IEC 62056-62.
This table is initialized in wrapper_Init() function, contained in wrapper.c
- 7. list_IPv4Setup_Info[]**

This is an IC specific table that configures information for all IPv4Setup(IC 42) objects in the master list. In IP based communication profile, this information is used for initialization of IP port, and managing data communication through the same.
This table contains all attribute values except logical name, for Interface Class 42, as specified in IEC 62056-62.
This table is initialized in wrapper_Init() function, contained in wrapper.c
- 8. list_EthernetSetup_Info[]**

This is an IC specific table that configures information for all EthernetSetup(IC 43) objects in the master list. The information is used in IP based communication profile.
This table contains all attribute values except logical name, for Interface Class 43, as specified in IEC 62056-62.
- 9. list_GprsModemSetup_Info[]**

This is an IC specific table that configures information for all GprsModemSetup(IC 45) objects in the master list. In IP based communication profile, this information is used to initialize a GPRS modem.
This table contains all attribute values except logical name, for Interface Class 45, as specified in IEC 62056-62.
This table is initialized in wrapper_Init() function, contained in wrapper.c

10. list_SmtpSetup_Info[]

This is an IC specific table that configures information for all SntpSetup(IC 46) objects in the master list. The information is used in IP based communication profile.

This table contains all attribute values except logical name, for Interface Class 46, as specified in IEC 62056-62.

This table is initialized in wrapper_Init() function, contained in wrapper.c

Once the configuration of obis.h is completed, the OEM must set the parameters in config.h. This configuration file contains various pre-compiler directives, the purpose of which are described in code comments. User is advised to understand each pre-compiler directive and edit the values as required.

The config.h header file also contains **#SUPPORT_IC_XX** macros to conditionally compile-in support for required Interface Classes only. This can be used to exclude unnecessary classes from the code during build and thus can help reduce code-space

6. Test Configuration

At this stage OEM will have a SCL with their specific configuration and platform interface fully defined.

OEM can test the implementation so far by connecting to the server with a client and testing the associations, object-lists and reads of static-information.

OEM can also perform reads of dynamic values but will receive junk-bytes (containing whatever is available in cosemExtDataBuf)

7. Implement Data Interface

Data Interface connects the SCL with Meter's data. For each DLMS service that needs access to Meter's data, separate Data Interface function is made. The functions are declared in data.h and defined with empty bodies in data.c. OEM must fill these functions, to suitably pass data between Meter and SCL.

In all the data interface functions the buffer *cosemExtDataBuf[]* is used as an intermediate data buffer for exchange of data between the SCL and meter code. When the SCL receives a GET or READ request, the meter code must fill data into the cosemExtDataBuf and when the SCL receives a SET or WRITE or ACTION request, the meter code must extract the data from the cosemExtDataBuf.

Guidelines to implement Data Interface functions:

1. KDEFS_USHORT data_GetAttributeValue(void)

The function is used when server receives a remote request for Getting Meter data (Note that the original request may be a LN GET request or a SN Read request).

Cosem Attribute Descriptor – global variables in data.c (curOBIS, curIC, curAttrIndex) helps OEM to identify the Attribute of Object Instance to be accessed.

OEM must write requested Meter data into cosemExtDataBuf[] and return the status of operation(success/error code).

Please note that only the raw data bytes should be written into cosemExtDataBuf.

The SCL will automatically encode the type and length information into the response

from the configuration interface data. For long data Get (data to be encoded as array), OEM must return the exact number of entries as requested by SCL using global variables “ext_fromEntry” and “ext_toEntry”.

NOTE: The mechanism of long data Get using Globals “ext_fromEntry” and “ext_toEntry” is applicable for all Interface Class attributes except for Passive and Active Day Profile attributes of an Activity Calendar object (Attributes 5 and 9 of an IC: 20 object). For these attributes OEM must always return day ID as first byte following information of timeswitch (day action). Global variables “ext_from_DayAction” and “ext_to_DayAction” informs OEM about the from and to index of day action information that it must pass to SCL.

2. KDEFS_UCHAR data_SetAttributeValue(void)

The function is used when server receives a remote request to Set Meter data. Cossem Attribute Descriptor in data.c (curOBIS, curIC, curAttrIndex) helps OEM to identify the Attribute of Object Instance to be written. OEM must Set Meter with data stored in cosemExtDataBuf[]. If the Attribute to be written stores array data, SCL will indicate the “From” and “To” array index of data in cosemExtDataBuf[] using global variables “ext_fromEntry” and “ext_toEntry” respectively. The function should return a SUCCESS or FAILURE.

Once again the cosemExtDataBuf will contain only the raw data. The encoding information will already have been verified and stripped away by the SCL.

NOTE: All Interface Class attributes will be “written” using this interface function except for the Passive and Active Day Profile attributes of an Activity Calendar object (Attributes 5 and 9 of an IC:20 object). For these attributes a separate interface function is defined below, since the set-data contains 2 levels of nesting. For this special function, user does not need to use the ext_fromEntry and ext_toEntry global variables.

3. KDEFS_UCHAR data_SetDayProfile(void)

The function is used when server receives a remote request to Set Day Profile Table of Activity Calendar. Cossem Attribute Descriptor in data.c (curOBIS, curIC, curAttrIndex) helps OEM to identify the Attribute of Object Instance to be accessed. The Day ID of data to set is indicated by global variable “ext_dayID”. “From” and “To” index of Day profile Action contained in the specific Day ID is indicated by variables “ext_from_DayAction” and “ext_to_DayAction” respectively. The function should return a SUCCESS or FAILURE.

Note that if a SET or WRITE request packet contains data for multiple DayProfiles, the SCL will call this function repeatedly, passing in the values for an individual DayProfile each time

4. KDEFS_UCHAR data_ActionAttributeValue(void)

The function is used when server receives a remote request to Execute Meter method. CossemMethod Descriptor in data.c (curOBIS, curIC, curMethIndex) helps OEM to identify the Method of Object Instance to be executed. Argument Data (if any) required to execute Method will be contained in cosemExtDataBuf[]. OEM must execute specified Method and return a SUCCESS or FAILURE.

5. KDEFS_UCHAR data_CheckPassword(KDEFS_UCHAR* passwd, KDEFS_UCHAR len)

The function is used to verify Authentication password during Association establishment phase. The password received from remote client is contained in “passwd []”, the number of character in the password received is contained in “len” variable. OEM must check the password and return a SUCCESS or FAILURE.

The Association for which the password-check is being performed is indicated by the value of the global variable curAssoc. This value indicates the index of the current association in the list_Associations[] array.

6. KDEFS_BOOL data_CheckDeviceAddress(KDEFS_USHORT device_address)

The function is used to verify Meter Physical Device Address during Data Link Connection establishment phase. Device address received from remote client is contained in global variable “device_address”. OEM must check device address and return TRUE (Device address correct) or FALSE (Device address wrong).

8. Test Complete Server Implementation

Now the Server Implementation is complete. OEM can test the Server Implementation at Data Link Level, Application Level and Data Interface (Accessing Meter Data) Level.