International Research Journal of Engineering and Management Studies (IRJEMS)
Volume: 03 Issue: 05 | May -2019 ISSN: 2395-0126

Custom Parser for Serialize Data Schema for Energy Meter

Dinesh Singh[1], Ankur Goyal[2]

[1]Student, Department of Computer Science & Engg., YIT, Jaipur, India
[2]Associate Professor, Department of Computer Science & Engg., YIT, Jaipur, India

---***---

Abstract - *In this paper, we study the Profile generic class (Class id = 7) of DLMS/COSEM. Profile generic class provides a generalized concept to store, sort and access data groups or data series. We suggest a custom parser of serialize data schema for energy meters which convert DLMS APDU (Application Protocol Data Unit) in user understandable format that easily shoe in user interface (UI).*

***Key Words*: DSL, DLMS/COSEM, Energy meter, Profile Generic, Parser etc.**

1. INTRODUCTION

Domain specific languages (DSLs) aim at raising the abstraction level for programmers, thereby enhancing both quality and productivity in software development. Device Language Message Specification (DLMS) is domain specific language which used in power domain. This is the standard language for smart devices. It is an application layer specification, independent of the lower layers and thus of the communication channel, designed to support messaging to and from (energy) distribution devices in a computer-integrated environment. The DLMS (Device Language Message Specification) User Association (UA) provides the DLMS/COSEM [2-4] that is the suite of standards. DLMS UA is considered by the IEC TC13 WG14 into the IEC 62056 series of standards. DLMS UA has been established for international standards to exchange data of a meter. Stated by DLMS UA, the meter must be embedded with the information including registration, maintenance and conformance testing services. COSEM (Companion Specification for Energy Metering) integrated set of the protocol layer (Transport Layer and Application Layer) to combine with DLMS protocol. The DLMS/COSEM Specification determines the protocol of an interface model and communication to exchange data of meter equipment.

2. THE COSEM INTERFACE CLASSES

For specification purposes, DLMS uses the technique of object modelling. An object is a collection of attributes and methods. Attributes represent the characteristics of an object. The value of an attribute may affect the behavior of an object. The first attribute of any object is the logical_name. It is one part of the identification of the object. An object may offer a number of methods to either examine or modify the values of the attributes. Objects that share common characteristics are generalized as an interface class, identified with a class_id. Within a specific IC, the common characteristics (attributes and methods) are described once for all objects. Instantiations of ICs are called COSEM interface objects.

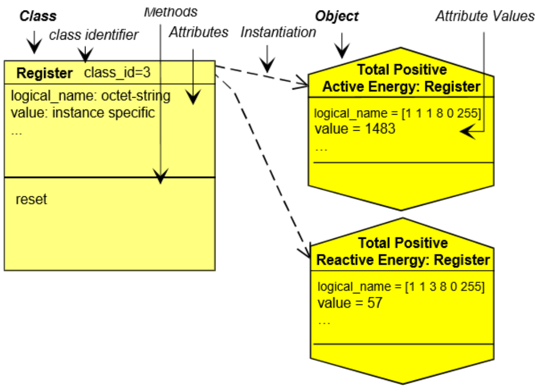

Class
class identifier
Methods
Attributes
Instantiation
Object
Attribute Values
Register class_id=3
logical_name: octet-string
value: instance specific
...
reset
Total Positive Active Energy: Register
logical_name = [1 1 1 8 0 255]
value = 1483
...
Total Positive Reactive Energy: Register
logical_name = [1 1 3 8 0 255]
value = 57
...

Fig - 1: An interface class and its instances

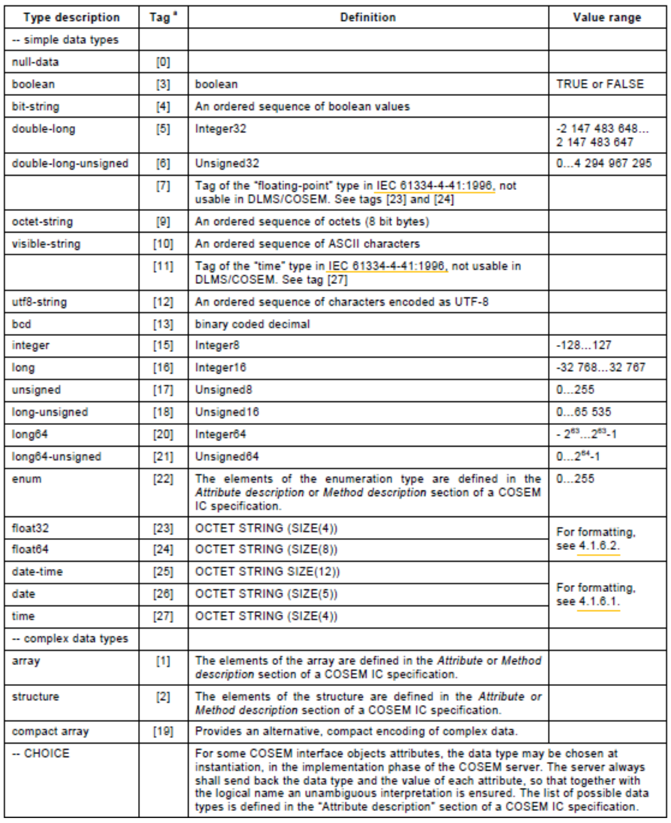
The IC "Register" is formed by combining the features necessary to model the behavior of a generic register (containing measured or static information) as seen from the client (data collection system, hand held terminal). The contents of the register are identified by the attribute logical_name. The logical_name contains an OBIS identifier (see Clause 7). The actual (dynamic) content of the register is carried by its value attribute. Defining a specific meter means defining several specific objects. In the example of Figure 3, the meter contains two registers; i.e. two specific instances of the IC "Register" are instantiated. Through the instantiation, one COSEM object becomes a "total, positive, active energy register" whereas the other becomes a "total, positive, reactive energy register".

3. DATE TIME FORMAT

```
date-time        OCTET STRING (SIZE(12))
                 {
                      year highbyte,
                      year lowbyte,
                      month,
                      day of month,
                      day of week,
                      hour,
                      minute,
                      second,
```

hundredths of second,
deviation highbyte,
deviation lowbyte,
clock status
}

4. COMMON DATA TYPES



Type description	Tag *	Definition	Value range
-- simple data types			
null-data	[0]		
boolean	[3]	boolean	TRUE or FALSE
bit-string	[4]	An ordered sequence of boolean values	
double-long	[5]	Integer32	-2 147 483 648… 2 147 483 647
double-long-unsigned	[6]	Unsigned32	0…4 294 967 295
	[7]	Tag of the "floating-point" type in IEC 61334-4-41:1996, not usable in DLMS/COSEM. See tags [23] and [24]	
octet-string	[9]	An ordered sequence of octets (8 bit bytes)	
visible-string	[10]	An ordered sequence of ASCII characters	
	[11]	Tag of the "time" type in IEC 61334-4-41:1996, not usable in DLMS/COSEM. See tag [27]	
utf8-string	[12]	An ordered sequence of characters encoded as UTF-8	
bcd	[13]	binary coded decimal	
integer	[15]	Integer8	-128…127
long	[16]	Integer16	-32 768…32 767
unsigned	[17]	Unsigned8	0…255
long-unsigned	[18]	Unsigned16	0…65 535
long64	[20]	Integer64	- 2^{63}…2^{63}-1
long64-unsigned	[21]	Unsigned64	0…2^{64}-1
enum	[22]	The elements of the enumeration type are defined in the *Attribute description* or *Method description* section of a COSEM IC specification.	0…255
float32	[23]	OCTET STRING (SIZE(4))	For formatting, see 4.1.6.2.
float64	[24]	OCTET STRING (SIZE(8))	
date-time	[25]	OCTET STRING SIZE(12))	For formatting, see 4.1.6.1.
date	[26]	OCTET STRING (SIZE(5))	
time	[27]	OCTET STRING (SIZE(4))	
-- complex data types			
array	[1]	The elements of the array are defined in the *Attribute* or *Method description* section of a COSEM IC specification.	
structure	[2]	The elements of the structure are defined in the *Attribute or Method description* section of a COSEM IC specification.	
compact array	[19]	Provides an alternative, compact encoding of complex data.	
-- CHOICE		For some COSEM interface objects attributes, the data type may be chosen at instantiation, in the implementation phase of the COSEM server. The server always shall send back the data type and the value of each attribute, so that together with the logical name an unambiguous interpretation is ensured. The list of possible data types is defined in the "Attribute description" section of a COSEM IC specification.	

Fig - 2: Common Data Types

5. PROFILE GENERIC (Class_id=7, version=1)

This IC provides a generalized concept allowing to store, sort and access data groups or data series, called *capture objects* [3]. Capture objects are appropriate attributes or elements of (an) attribute(s) of COSEM objects. The capture objects are collected periodically or occasionally.

A profile has a *buffer* to store the captured data. To retrieve only a part of the buffer, either a value range or an entry range may be specified, asking to retrieve all entries that fall within the range specified.

The list of *capture objects* defines the values to be stored in the *buffer* (using auto capture or the method *capture*). The list is defined statically to ensure homogenous buffer entries (all entries have the same size and structure). If the list of capture objects is modified, the *buffer* is cleared. If the buffer is captured by other "Profile generic" objects, their *buffer* is cleared as well, to guarantee the homogeneity of their *buffer* entries.

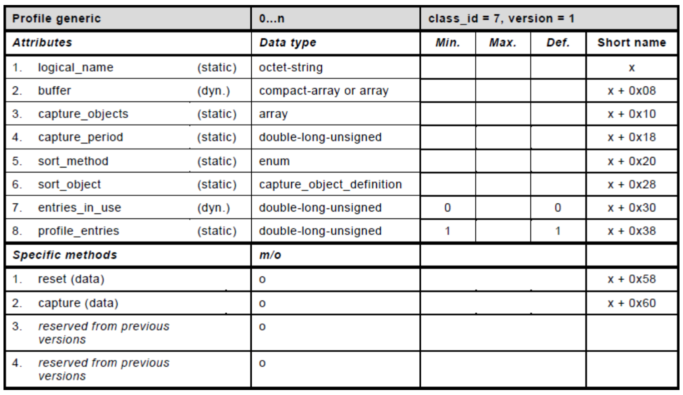
The *buffer* may be defined as sorted by one of the *capture objects*, e.g. the clock, or the entries are stacked in a "last in first out" order. For example, it is very easy to build a "maximum demand register" with a one entry deep sorted profile capturing and sorted by a "Demand register" *last_average_value* attribute. It is just as simple to define a profile retaining the three largest values of some period.

The size of profile data is determined by three parameters:

a) The number of entries filled (*entries_in_use*). This will be zero after clearing the profile;

b) The maximum number of entries to retain (profile_entries). If all entries are filled and a capture () request occurs, the least important entry (according to the requested sorting method) will get lost. This maximum number of entries may be specified. Upon changing it, the buffer will be adjusted;

c) The physical limit for the buffer. This limit typically depends on the objects to capture. The object will reject an attempt of setting the maximum number of entries that is larger than physically possible



Profile generic		0…n	class_id = 7, version = 1			
Attributes		***Data type***	***Min.***	***Max.***	***Def.***	**Short name**
1. logical_name	(static)	octet-string				x
2. buffer	(dyn.)	compact-array or array				x + 0x08
3. capture_objects	(static)	array				x + 0x10
4. capture_period	(static)	double-long-unsigned				x + 0x18
5. sort_method	(static)	enum				x + 0x20
6. sort_object	(static)	capture_object_definition				x + 0x28
7. entries_in_use	(dyn.)	double-long-unsigned	0		0	x + 0x30
8. profile_entries	(static)	double-long-unsigned	1		1	x + 0x38
Specific methods		***m/o***				
1. reset (data)		o				x + 0x58
2. capture (data)		o				x + 0x60
3. *reserved from previous versions*		o				
4. *reserved from previous versions*		o				

Fig - 3: Profile Generic Class

Table -1: Attribute Description

Logical name	It is always the first attribute of an IC. It identifies the instantiation (COSEM object) of this IC. The value of the logical_name conforms to OBIS.
buffer	Contains a sequence of entries. Each entry contains values of the captured objects. compact-array or array entry entry ::= structure

	{
	CHOICE
	{
	-- simple data types
	null-data [0],
	boolean [3],
	bit-string [4],
	double-long [5],
	double-long-unsigned [6],
	octet-string [9],
	visible-string [10],
	utf8-string [12],
	bcd [13],
	integer [15],
	long [16],
	unsigned [17],
	long-unsigned [18],
	long64 [20],
	long64-unsigned [21],
	enum [22],
	float32 [23],
	float64 [24],
	date-time [25],
	date [26],
	time [27],
	-- complex data types
	array [1],
	structure [2],
	compact-array [19]
	}
	}
	The number and the order of the elements of the structure holding the entries is the same as in the definition of the capture_objects. The buffer is filled by auto captures or by subsequent calls of the method capture. The sequence of the entries within the array is ordered according to the sort_method specified.
	Default: The buffer is empty after reset.

capture _ objects	Specifies the list of capture objects that are assigned to the object instance.
	Upon a call of the *capture* (data) method or automatically in defined intervals, the selected attributes are copied into the *buffer* of the profile.
	array capture_object_definition
	capture_object_definition ::= structure
	{
	class_id: long-unsigned,
	logical_name: octet-string,
	attribute_index: integer,
	data_index: long-unsigned
	}
	- Where attribute_index is a pointer to the attribute within the object identified by its class_id and *logical_name*. Attribute_index 1 refers to the 1st attribute (i.e. the *logical_name*), attribute_index 2 to the 2nd, etc.); attribute_index 0 refers to all public attributes;
	- Where data_index is a pointer selecting a specific element of the attribute. The first element in the attribute structure is identified by data_index 1. If the attribute is not a structure, then the data_index has no meaning.

capture _ period	>=1: Automatic capturing assumed. Specifies the capturing period in seconds. 0: No automatic capturing; capturing is triggered externally or capture events occur asynchronously.
sort_me thod	If the profile is unsorted, it works as a "first in first out" buffer. If the *buffer* is full, the next call to *capture* () will push out the first (oldest) entry of the *buffer* to make space for the new entry. If the profile is sorted, a call to *capture*() will store the new entry at the appropriate position in the buffer, moving all following entries and probably losing the least interesting entry. If the new entry would enter the *buffer* after the last entry and if the *buffer* is already full, the new entry will not be retained at all. enum: (1) fifo (first in first out), (2) lifo (last in first out), (3) largest, (4) smallest, (5) nearest_to_zero, (6) farest_from_zero ***Def.*** fifo
sort_obj ect	If the profile is sorted, this attribute specifies the register or clock that the ordering is based upon. capture_object _definition See above. ***Def.*** no object to sort by (only possible with sort_method fifo or lifo)
entries_ in_use	Counts the number of entries stored in the buffer. After a call of the *reset* () method, the buffer does not contain any entries, and this value is zero. Upon each subsequent call of the *capture* () method, this value will be incremented up to the maximum number of entries that will get stored (see *profile_entries*). double-long-unsigned 0...profile_entries *Def.* 0
profile_ entries	Specifies how many entries shall be retained in the buffer. double-long-unsigned 1...(limited by physical size) *Def.* 1

Method description

reset (data) Clears the *buffer*. It has no valid entries afterwards; *entries_in_use* is zero after

	this call. This call does not trigger any additional operations on the capture objects. Specifically, it does not reset any attributes captured. data ::= integer (0)
capture (data)	Copies the values of the objects to capture into the *buffer* by reading each capture object. Depending on the *sort_method* and the actual state of the *buffer* this produces a new entry or a replacement for the less significant entry. As long as not all entries are already used, the *entries_in_use* attribute will be incremented. This call does not trigger any additional operations within the capture objects such as *capture* () or *reset* (). Note, that if more than one attribute of an object need to be captured, they have to be defined one by one on the list of *capture_objects*. If the attribute_index = 0, all attributes are captured. data ::= integer (0)

6. ANALYSIS OF PROFILE GENERIC CLASS

Attribute 3 is capture object for profile generic class. It contains list of parameters which is captured by energy meter at specified interval or ad-hoc. Specified interval contains daily for daily energy snapshots, monthly for billing data. Here ad-hoc means when data captured instantly in case of any events like power off or power on.

Attribute 2 is buffer which contains actual data. It is an array which has meter data values as defined in capture objects. The sequence of data is same as defined in attribute 3.

The maximum size of buffer is defined in attribute 8. When buffer entry reached at maximum size the buffer entry must be roll over in attribute 5(sort method) manner by default it is first in first Out (FIFO).

Total entry present in buffer is defined by attribute 7(entry in use). Once buffer entry reaches maximum size then entry in use is equal to profile entries.

Attribute 6 defined the sorting parameter of buffer.

7. CUSTOM PARSER FOR PROFILE GENERIC CLASS

We purposed custom parsing of profile data in form of table. A table is made up of rows and columns. If you think of a table as a grid, the column go from left to right across the grid and each entry of data is listed down as a row.

7.1 Columns

Columns [5] are defined to hold a specific type of data, such as dates, numeric, or textual data. In the simplest of definitions a column is defined by its name and data type. Columns names can't be duplicated in a table.

7.2 Rows

A table can contain zero or more rows [5]. When there is zero, it said to be empty. There is not practical limit on the number of rows a table can hold. There is no guarantee that the rows in a table are stored in a particular order.

In Profile generic class we mainly focused on attribute 2 and 3 for parsing data by custom parser. An attribute 3 capture object represents columns in table. Attribute 2 buffers consists of an array which act as a rows in table

Example: Daily Load Profile data of meter [1]

DLMS APDU for Attribute 2:
01030203090C07E3031DFF000000FF80000005000
0011F050000014A0203090C07E3031CFF000000FF
800000050000011D05000001470203090C07E303
1BFF000000FF800000050000011805000000141

DLMS APDU for Attribute 3:
010302041200080906000001000FF0F021200000
20412000309060100010800FF0F0212000002041 2
00030906010009080 0FF0F02120000

In capture object attribute 3 of daily Load Profile have 3 columns (Parameters)

1. Class id: 8 Logical Name: 0.0.1.0.0.255 Attribute: 2 (Date Time)
2. Class id: 3 Logical Name: 1.0.1.8.0.255 Attribute: 2 (Cumulative Active Energy)
3. Class id: 3 Logical Name: 1.0.9.8.0.255 Attribute: 2 (Cumulative Apparent Energy)

In buffer attribute 3 of Daily Load Profile have 3 rows

1. 0203090C07E3031DFF000000FF8000000500 00011F050000014A
2. 0203090C07E3031CFF000000FF8000000500 00011D0500000147
3. 0203090C07E3031BFF000000FF8000000500 0001180500000141

So when we convert it in to table format

Table -2: Daily Load Profile

8: 0.0.1.0.0.255:2 Date Time	3: 1.0.1.8.0.255:2 Cum. Active Energy	3: 1.0.1.9.0.255:2 Cum. Apparent Energy
29/03/2019 00:00:00	287	330
28/03/2019 00:00:00	285	327
27/03/2019 00:00:00	280	321

8. CONCLUSIONS

In Custom Parser for Serialize data schema we have to convert profile generic data into database table format. So we convert DLMS APDU complex data into intermediate language (tabular format) which is captured by any UI (user interface) and show the meter data to user.

The paper proposes only profile generic class data custom parsing.

REFERENCES

[1] Data Exchange for Electricity Meter Reading, Tariff and Load Control – Companion Specification, Version 1.2, edition 2,2009
[2] DLMS User Association, COSEM Architecture and Protocols, Eighth Edition.
[3] DLMS User Association, Identification System and Interface Classes, Twelfth Edition, pp.64–68.
[4] http://www.dlms.com/information/whatisdlmscosem/index.html.
[5] https://www.essentialsql.com/what-is-a-database-table/