

1

2

3

4

5

This document is based on
German draft SML version 1.04
dated 13.07.2010
including some minor corrections

6

Editorial Version by MW, dated 07.01.2011

7

Update during WG-Meeting 17.01.2011

8

Editorial Version by MW, dated 24.03.2011

9

Update during WG-Meeting 05.05.2011 / restructured according to meeting decisions

10

Update during WG-Meeting 02.12.2011

11

12

SML Container Services

13

14

15

The first page has to be adapted later.

16

17			
18			
			List of contents
19	i	List of illustrations	4
20	ii	List of tables	4
21	iii	List of abbreviations	4
22			
23	1	Scope	6
24	2	Normative references	6
25	3	Terms and definitions	6
26	4	Basic structure of SML	6
27	5	Basic Structure of Smart Message Language SML	8
28	6	SML messages	9
29	6.1	SML Messages	12
30	6.1.1	SML_PublicOpen Req	12
31	6.1.2	SML_PublicOpen Res	13
32	6.1.3	SML_PublicClose Req	15
33	6.1.4	SML_PublicClose Res	15
34	6.1.5	SML_Attention Res	15
35	6.1.5.1	Global SML Attention Numbers	16
36	6.1.6	SML_GetCosem Req	17
37	6.1.7	SML_GetCosem Res	18
38	6.1.8	SML_SetCosem Req	20
39	6.1.9	SML_SetCosem Res	21
40	6.1.10	SML_ActionCosem Req	21
41	6.1.11	SML_ActionCosem Res	21
42	7	SML binary encoding, directly packed coding	22
43	7.1	Type-length field	22
44	7.2	Coding of data types	23
45	7.2.1	Data type Octet String	23
46	7.2.2	Data types Integer8, Integer16, Integer32 and Integer64	24
47	7.2.3	Data types Unsigned8, Unsigned16, Unsigned32 and Unsigned64	24
48	7.2.4	Data type Boolean	25
49	7.2.5	Data type List of	25
50	7.3	Coding of special features	26
51	7.3.1	Feature End of an SML message	26
52	7.3.2	Feature SEQUENCE	26
53	7.3.3	Feature CHOICE	26
54	7.3.4	Feature OPTIONAL	26
55	8	XML - coding	26
56	A.1	SML Messages for non COSEM environments	36
57	A.1.1	SML_GetProfilePack Req	36
58	A.1.2	SML_GetProfilePack Res	37
59	A.1.3	SML_GetProfileList Req	40
60	A.1.4	SML_GetProfileList Res	41
61	A.1.5	SML_GetProcParameter Req	42
62	A.1.6	SML_GetProcParameter Res	43
63	A.1.7	SML_SetProcParameter Req	45

64	A.1.8	SML_GetList.Req	45
65	A.1.9	SML_GetList.Res	45
66	B.1	SML Transport Protocol (CHAPTER has to move out ...)	47
67	B.1.1	Version 1	47
68	B.1.2	Version 2	48
69	B.1.3	Initiating of the transmission in version 2	48
70	B.1.4	Labelling of blocks	48
71	B.1.5	Labelling of SML files	49
72	B.1.6	Feature ,VV'	49
73	B.1.7	Negotiation of the timeout to be used	49
74	B.1.8	Negotiation of the maximum allowed block size	49
75	B.1.9	Process of establishing a transmission	49
76	B.1.10	Process of the course of a transmission	50
77	B.1.11	Example of the course of a version 2 transmission process	50
78			

79 List of illustrations

80			
81	FIG. 1:	SML MESSAGES AND COMMUNICATION PATHS	7
82	FIG. 2:	SML COMMUNICATION MODEL.	7
83	FIG. 3:	LOCATION OF SML WITHIN THE FIELD OF EUROPEAN AND INTERNATIONAL STANDARDS	8
84	FIG. 4:	XML SCHEME FILE FOR CODING OF SML PER XML	35

85

86 List of tables

87			
88	TAB. 1:	EXAMPLE FOR USING THE 'GROUP NUMBER' FEATURE	10
89	TAB. 2:	LIST OF GLOBAL ERROR NUMBERS	16
90	TAB. 3:	LIST OF GLOBAL ATTENTION NUMBERS	17
91	TAB. 4:	BIT CODING IN THE TYPE-LENGTH FIELD FOR THE FIRST BYTE OF A TL-FIELD PARTICULAR	22
92	TAB. 5:	BIT CODING IN THE TYPE-LENGTH FIELD FOR THE SECOND AND FOLLOWING TL-FIELD BYTES	23
93	TAB. 6:	BIT CODING IN TYPE-LENGTH FIELD FOR AN OCTET STRING	23
94	TAB. 7:	ESCAPE FEATURES FOR THE SML TRANSPORT PROTOCOL	47

95

96

List of abbreviations

97 Units:

98 In regard to physical measured variables and units, the agreements laid down in SI
99 (see ISO 1000) shall apply.

100 Relevant abbreviations:

101 The abbreviations below can be followed by Arabic numerals enabling distinctions to
102 be made between multiple variants of the same function/signal.

103	ASN.1	⇔ Abstract Syntax Notation One,
104	BER	⇔ Basic Encoding Rules,
105	CCITT	⇔ Comité Consultatif International Télégraphique et Téléphonique,
106	CR	⇔ Carriage Return,
107	DLMS	⇔ Device Language Message Specification,
108	EN	⇔ European Standard,
109	ID	⇔ Identification Number,
110	IEC	⇔ International Electrotechnical Commission,
111	IEEE	⇔ Institute of Electrical and Electronics Engineers,
112	IP	⇔ Internet Protocol,
113	ISO	⇔ International Organisation for Standardisation,
114	LAN	⇔ Local Area Network,
115	LSB	⇔ Least Significant Bit, niederwertigstes Bit,
116	MDE	⇔ Mobile data acquisition device,
117	MSB	⇔ Most Significant Bit, höchstwertigstes Bit,

118	OBIS	⇔ Object Identification System,
119	OSI	⇔ Open Systems Interconnection Reference Model,
120	RS232	⇔ Serial interface,
121	S-I	⇔ Seconds Index,
122	SML	⇔ Smart Message Language,
123	TAG	⇔ Feature / identification / tagging for the coding
124		of data elements,
125	TCP	⇔ Transmission Control Protocol,
126	TL	⇔ Type-Length,
127	UDP	⇔ User Datagram Protocol,
128	WAN	⇔ Wide Area Network,
129	XML	⇔ Extensible Markup Language.
130		

131 SPECIFICATION OF THE SMART MESSAGE LANGUAGE SML

132 1 Scope

133 The goal of the specification was to find a maximally simple structure also suitable for
134 implementation in low-power embedded systems that can be used for data procurement over
135 wide-area routes. This specification defines a communication protocol for applications in the
136 environment of data procurement and equipment parameterisation.

137 2 Normative references

138 The following referenced documents are indispensable for the application of this document.
139 For dated references, only the edition cited applies. For undated references, the latest edition
140 of the referenced document (including any amendments) applies.

141 CCITT-CRC16, *Standard of the CCITT for checksum calculation*

142 IEC 62056-21, *Measuring electrical energy – meter reading transmission, Part 21: Direct local*
143 *data exchange*

144 IEC 62056-46, *Measuring electrical energy - meter reading transmission, tariff and load*
145 *control – Part 46: Use of the HDLC protocol in the connection layer*

146 IEC 62056-61, *Measuring electrical energy – meter reading transmission, Part 61: OBIS*
147 *Object Identification System*

148 IEC 62056-62, *Measuring electrical energy - meter reading transmission, tariff and load*
149 *control – Part 62: Interface Classes*

150 ISO 1000, *SI units and recommendations for the use of their multiples and of certain other*
151 *units*

152 ISO 8859-15, *Information technology – 8-bit-individual-byte-coded character sets – Part 15:*
153 *Latin Alphabet No. 9*

154 3 Terms and definitions

155 Some important terms used in this document are explained or defined below.

156 3.1 SML file

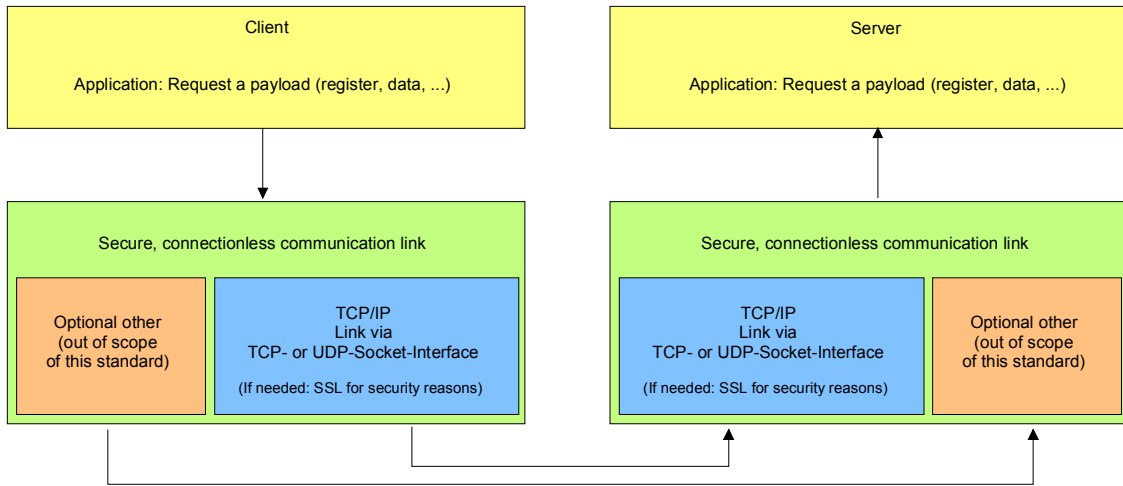
157 denotes an information unit that, completely detached from the transport technology involved
158 (internet, telephone, ...), is self-contained. SML files can in this sense be comprehended as
159 self-contained information units that (just like an email) are embedded and transmitted in a
160 protocol (see Fig. 2). The approach of using SML files means the concept is independent of
161 the task of having to define specific protocols for information interchange. Instead, all that is
162 demanded is to select a certain protocol (e.g. HTTP, FTP, ...) in a particular duty case and
163 parameterise this appropriately. If SML files are used as files on computer systems, these
164 files must be noted without the use of additional frames and utilising the coding defined with
165 Section 7, unless the particular application involved explicitly specifies a divergent stipulation.

166 4 Basic structure of SML

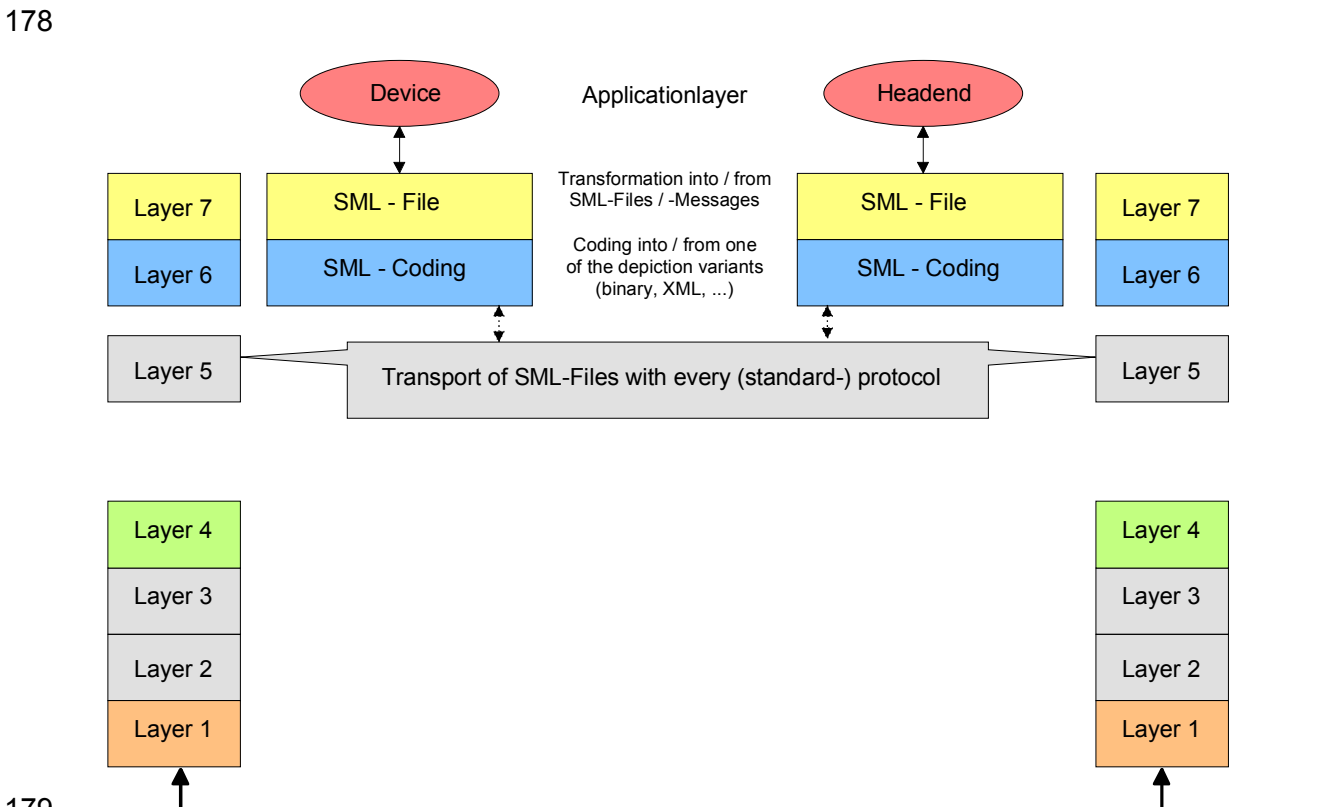
167 The basic structure contains the following elements:

- 168 – Smart Message Language defines a file structure / document structure for recording the
- 169 useful loads between the end points
- 170 – SML Binary Encoding defines a packed binary coding for SML;
- 171 – SML XML Encoding defines the coding of SML in XML;
- 172 – SML Transport Protocol, required for serial point-to-point links.

173 SML messages, see no. 5, can, like an email in the final analysis, be transported over
 174 stateless, secure communication paths. For the duty scenario being targeted, the following
 175 model can accordingly be adduced for an overview:

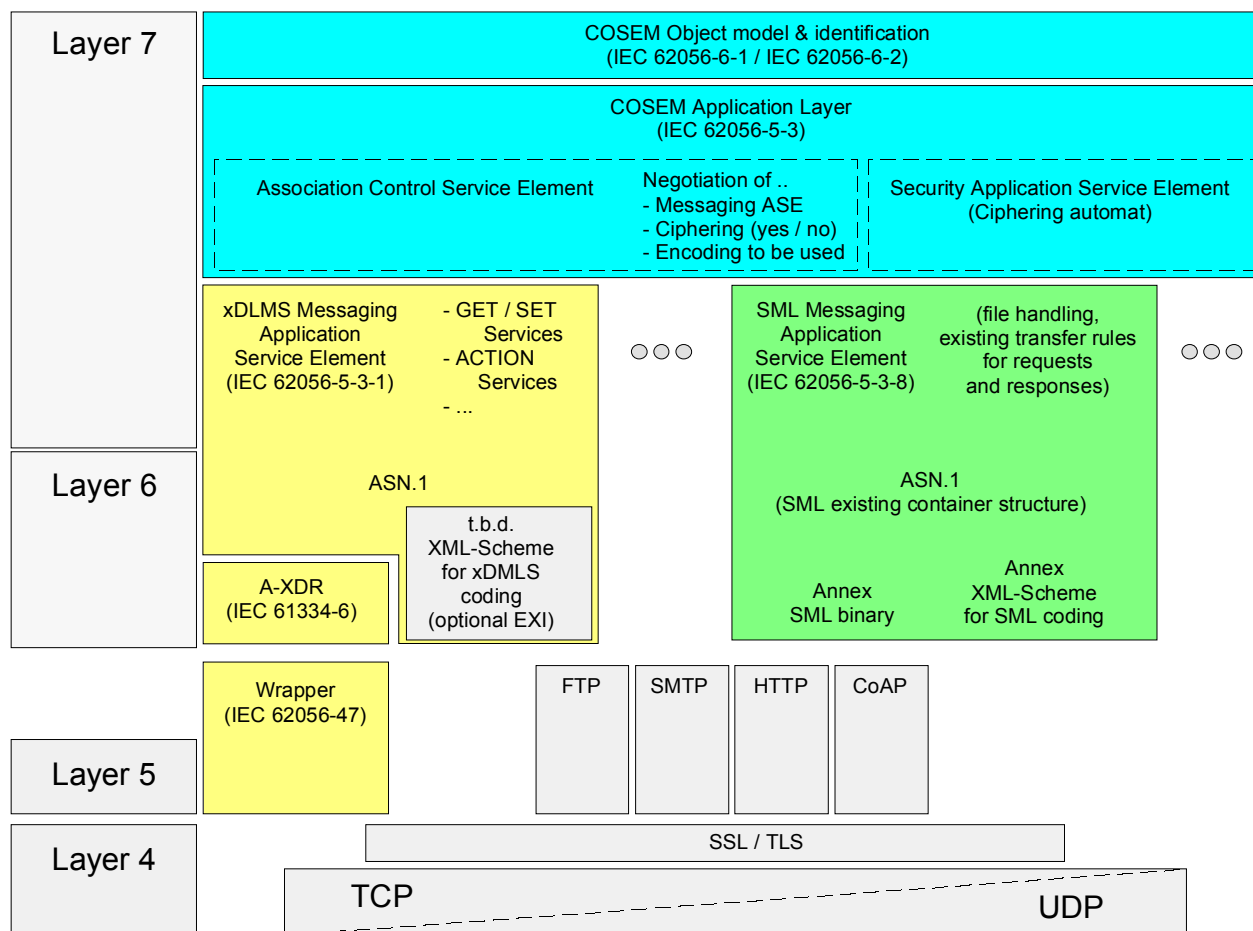


176
 177 **Fig. 1: SML messages and communication paths**



179
 180 **Fig. 2: SML communication model.**

181



182

183

184

Fig. 3: Location of SML within the field of European and international standards

185 5 Basic Structure of Smart Message Language SML

186 An SML file is always structured as a chain of SML messages.

187 SML files can be segmented in order to reduce the file size involved.

188 SML files can occur in the following variants:

189 ... SML order file,

190 ... SML response file or

191 ... SML combi-file

192

193 SML order files contain the orders ("Requests"). SML response files group together the responses to
194 the requests.

195 Each SML order file begins with exactly one and contains exactly one SML_...Open.Reg message. It
196 ends with exactly one and contains exactly one SML_...Close.Reg message. This specification also
197 applies for SML response files to which no SML order file can be assigned, where the 'response'
198 variant must be used instead of the 'request' variant.

199 SML response files to which SML order files can be assigned begin with exactly one
200 SML_...Open.Res- or one SML_Attention.Res message. They end with exactly one
201 SML_...Close.Res- or one SML_Attention.Res message

202 For the use of SML per transport media with lower performance¹ SML messages can also be sent
203 without the SML file frame and exclusively as „Response without Request“. This use case must be
204 defined explicitly by the application.

205 An SML combi-file contains the SML messages of an SML order file plus the SML messages of the
206 associated SML response file(s).

207 **6 SML messages**

208 An SML message is either a ‘Request message’ or a ‘Response message’.

209 An SML message comprises task and assigned attributes.

```
210 (A) SML_Message ::= SEQUENCE
211     {
212     transactionId      Octet String,
213     groupNo           Unsigned8,
214     abortOnError      Unsigned8,
215     messageBody       SML_MessageBody,
216     crc16             Unsigned16,
217     endOfSmlMsg      EndOfSmlMsg
218     }
```

219 The ‘transactionId’ is formed by the client in an unambiguous form² at generation of ‘request
220 messages’. Each ‘response message’ reflects back the unchanged transaction number belonging to
221 its ‘request’, so that a ‘response’ can always be assigned to the associated ‘request’.

222 [Reference Ref. 1, see below]

223 When an SML file is created in the sense of ‘push mode’, and by reason of the principle involved no
224 SML request messages exist for its SML response messages, the creator of these SML response
225 messages automatically generates unambiguous transaction numbers.

226 The ‘groupNo’ attribute permits the formation of SML message groups. This mechanism shall be used
227 for stating which SML messages must be processed in a particular sequence, and which may be
228 executed on the side (in the sense of simultaneous work of both the first and second kind).

229 As a general principle, the groups are processed sequentially, in the same order as they occur within
230 an SML file.

231 Mixing SML messages from different groups is not allowed.

232 Messages inside a particular group can be processed by the recipient either serially or on the side.

233 The checksum (element ‘crc16’) must be calculated as CRC16 in accordance with IEC 62056-46.

234 Calculation begins with the first byte for ‘SML_Message’ and ends with the last byte for

¹ In this specification only the variants ‘short range radio’ and ‘PLC connections’ between sensors and actors are considered as *transport media with lower performance*.

² The unambiguity must in each case be assured within the context of the generator of the Transaction Number; a worldwide unambiguity is not demanded (comparable to the MAC address with Ethernet). Thanks to the other particulars in the SML-Open message, the unambiguity can be assigned to the generator involved.

235 'messageBody'. This means the bytes of elements 'crc16' and 'zero' are excluded from checksum
236 calculation.

237 When an SML request message is received that could be decoded and whose checksum is faulty, this
238 SML request message must be answered with an 'SML_Attention' (81 81 C7 C7 FE 0B, see Tab. 2)
239 and the field 'transactionId' according to reference Ref. 1. If in this case the 'SML_CloseRequest'
240 message is received, the fault must be ignored and in the response a correct 'SML_Close-Request'
241 must be sent whose 'transactionId' field is set according to reference Ref. 1.

242 If the 'SML_OpenRequest' message is received with a faulty checksum or a faulty
243 ,SML_PublicOpen.Reg' structure or a missing clientId or a missing reqFileId, the entire SML file must
244 be ignored³.

245 If a SML-Request-file was received which could be decoded but did not start with ,SML_Open-
246 Request' then the answer has to be a SML-Request file with one SML Attention message and error
247 code 'Unexpected SML message'. A further processing and/or a correct answering to the subsequent
248 SML messages is forbidden.

249 If an SML message is received that cannot be decoded, the following procedure must be adopted:

- 250 - If the SML message concerned is the first message, the entire SML file will be rejected (this
251 applies both for SML request files and also for SML response files).
- 252 - If one of the following SML messages is involved, then as the response for the SML message
253 concerned an SML-Attention (81 81 C7 C7 FE 08, see Tab. 2) is transmitted followed by an SML-
254 Close (applies only for SML order files). All further SML messages of the SML file involved will be
255 ignored (in the case of both SML order files and SML response files).
- 256 - The transactionId' for the SML_Attention' as well as for the following SML_Close must be
257 generated according to reference Ref. 1.
- 258 - If a SML_Close request could be recognised but the CRC inside is wrong the answer has to be a
259 SML_Close response with a transactionId according to reference Ref. 1.

260 SML messages generally have to be aggregated to SML files.

261

Tab. 1: Example for using the 'Group Number' feature

Sequence of SML messages	Transaction Number	Group-No.	SML message	Comment
0	0	1	OPEN	Is executed first.
1	1	4	GET_ProfilPack	First group, is executed as second block. For example, read out three load profiles.
2	2	4	GET_ProfilPack	
3	3	4	GET_ProfilPack	
4	4	7	GET_ProfilPack	Second group, is executed as third action, For example, read out a load profile, the billing list and the logbook.
5	5	7	GET_ProfilList	
6	6	7	GET_ProfilList	
7	7	8	SET_ProcParameter	Is executed as fourth action. For example, correct the reference time

³ Ignoring means: The received content must be rejected. No answer is generated.

Sequence of SML messages	Transaction Number	Group-No.	SML message	Comment
8	8	8	CLOSE.Req	Is processed last.

262 The Transaction Number is thus merely the feature for assigning the responses to the
 263 associated requests. It has no influence whatever on the sequence for executing the SML
 264 messages at the recipient.

265 The 'abortOnError' attribute specifies what procedure is to be followed in the event of errors in
 266 executing the SML message, with the concluding Close always having to be executed and
 267 non-executed requests always generating a response with the 'not executed' error message
 268 as the feedback message:

269	(B)	abortOnError	⇔ 0x00	Continue execution,
270		abortOnError	⇔ 0x01	Continue execution
271				as from next group,
272		abortOnError	⇔ 0x02	Continue execution beginning
273				at the ongoing group, then
274				do not execute any
275				more groups.
276				If a '0x02' can be found
277				within the same group
278				followed by a '0x01',
279				execution must be aborted
280				immediately.
281		abortOnError	⇔ 0xFF	Abort execution immediately.

282	(c)	SML_MessageBody	::=	CHOICE	
283		{			
284		OpenRequest		[0x00000100]	SML_PublicOpen.Req
285		OpenResponse		[0x00000101]	SML_PublicOpen.Res
286					
287		CloseRequest		[0x00000200]	SML_PublicClose.Req
288		CloseResponse		[0x00000201]	SML_PublicClose.Res
289					
290		GetProfilePackRequest		[0x00000300]	SML_GetProfilePack.Req
291		GetProfilePackResponse		[0x00000301]	SML_GetProfilePack.Res
292					
293		GetProfileListRequest		[0x00000400]	SML_GetProfileList.Req
294		GetProfileListResponse		[0x00000401]	SML_GetProfileList.Res
295					
296		GetProcParameterRequest		[0x00000500]	SML_GetProcParameter.Req
297		GetProcParameterResponse		[0x00000501]	SML_GetProcParameter.Res
298					
299		SetProcParameterRequest		[0x00000600]	SML_SetProcParameter.Req
300					
301		GetListRequest		[0x00000700]	SML_GetList.Req
302		GetListResponse		[0x00000701]	SML_GetList.Res
303					
304		GetCosemRequest		[0x00000800]	SML_GetCosem.Req
305		GetCosemResponse		[0x00000801]	SML_GetCosem.Res
306					
307		SetCosemRequest		[0x00000900]	SML_SetCosem.Req
308		SetCosemResponse		[0x00000901]	SML_SetCosem.Res
309					
310		ActionCosemRequest		[0x00000A00]	SML_ActionCosem.Req
311		ActionCosemResponse		[0x00000A01]	SML_ActionCosem.Res
312		AttentionResponse		[0x0000FF01]	SML_Attention.Res
313		}			
314					

315 6.1 SML Messages

316 6.1.1 SML_PublicOpen.Req

317 The SML_PublicOpen.Req must always be present at the beginning of an SML order file. It
 318 serves to identify the client and the authentication per user/password and the assignment of
 319 SML response file(s) to the SML order file.

320 Each client has to respond to the SML_PublicOpen.Req either with an SML_PublicOpen.Res
 321 or an SML_Attention.Res. The empty 'ServerID' must be considered as broadcast address for
 322 the messages which have to be answered. If an application in contrast to this uses special
 323 broadcast or multitask addresses these have to be defined explicitly in the application
 324 together with the behaviour.

```
325 (D) SML_PublicOpen.Req ::= SEQUENCE
326 {
327     codepage          Octet String OPTIONAL,
328     clientId          Octet String
329     reqFileId         Octet String
330     serverId          Octet String OPTIONAL,
331     username          Octet String OPTIONAL,
332     password          Octet String OPTIONAL,
333     smlVersion        Unsigned8 OPTIONAL,
334 }
```

335 Per 'codepage', if the value is stated, a codepage other than the default codepage is agreed.
336 The 'Codepage' specifies which character set is to be used for interpreting strings. If the value
337 is not stated, 'ISO 8859-15' will be used.

338 The content for 'Codepage' itself must always be noted as a string in ISO 8859-15.

339 The 'serverId' parameter, if stated, enables to address an SML data source (a specific
340 measuring device) or a software module selectively per 'PublicOpen.Req' in the sense of an
341 address. In this the case, all subsequent SML messages must likewise state the parameter
342 'serverId' with content. If the parameter 'serverId' is absent in one of the subsequent
343 messages, the response for the SML message concerned must be an 'SML_Attention' (81 81
344 C7 C7 FE 0C, see Tab. 2).

345 If the 'serverId' parameter is absent, the request will be evaluated as a 'broadcast'. In this
346 case, the 'serverId' parameter must likewise be absent in all subsequent SML messages. SML
347 messages that in this situation nonetheless state the 'serverId' parameter must be ignored. In
348 all cases to the final 'SML_Close' must be answered.

349 The 'clientId' parameter is generated by the client at generation of an SML order file and
350 serves for unambiguous addressing of the client's SML response file.

351 The 'reqFileId' parameter designates in a form unambiguous throughout the system⁴ a
352 specific SML order file/SML response file tuple. It enables SML responses to be assigned to
353 their requests.

354 The 'reqFileId' parameter supplies unambiguous identification for the SML file, e.g. formed
355 from the ongoing timestamp.

356 If the 'sml-Version' parameter is absent, Version 1 will be assumed as standard.

357 6.1.2 SML_PublicOpen.Res

358 SML_PublicOpen.Res always stands at the beginning of an SML response file. It serves to
359 identify the SML response file for the associated SML order file.

⁴ The unambiguity must in each case be assured within the context of the generator of 'reqFileId'; a worldwide unambiguity (comparable to the MAC address with Ethernet) is not demanded. Thanks to the other particulars in the SML-Open message, the unambiguity can be assigned to the generator involved.

```
360 (E) SML_PublicOpen.Res ::= SEQUENCE
361     {
362         codepage          Octet String OPTIONAL,
363         clientId          Octet String OPTIONAL,
364         reqFileId        Octet String,
365         serverId         Octet String,
366         refTime          SML_Time OPTIONAL,
367         smlVersion       Unsigned8 OPTIONAL,
368     }
```

369 The 'SML_Time' element is stated either as a seconds index or as a timestamp.

```
370 (F) SML_Time ::= CHOICE
371     {
372         secIndex          [0x01] Unsigned32,
373         timestamp         [0x02] SML_Timestamp
374         localTimestamp    [0x03] SML_TimestampLocal
375     }
```

376 If a timestamp is involved, this is always formed in seconds, starting from 01.01.1970,
377 00:00:00 (UNIX reference time).

```
378 (G) SML_Timestamp ::= Unsigned32
```

379

380 If a local time stamp is involved it has to be given as follows:

```
381 (H) SML_TimestampLocal ::= SEQUENCE
382     {
383         timestamp         SML_Timestamp,
384         localOffset       Integer16,
385         seasonTimeOffset Integer16
386     }
```

387 The element 'localOffset' must be given in minutes.

388 The element 'seasonTimeOffset' must be given in minutes.

389 The local time results as follows:

390 $localTime = Timestamp + localOffset + seasonTimeOffset$

391 If a time value is invalid, it must be coded by '0xFFFFFFFF'. If in that case it is a
392 SML_TimestampLocal, the values for 'localOffset' and 'seasonTimeOffset' must be coded by
393 '0x0000'.

394 If the response is supplied by a device or a software module that does not itself possess a
395 piece of time information, this particular will be absent in 'PublicOpen.Res'.

396 If the 'smlVersion' parameter is absent, Version 1 will be assumed as standard.

397 If the SML file concerned is an SML response file to which an SML request file belongs, the
398 'reqFileId' element of the SML request file will be returned.

399 If the SML file is an SML response file to which no SML request file belongs, the 'reqFileId'
400 element, comparable to generating this attribute when creating an SML request file, must be
401 set to a name unambiguous throughout the system.

402 If the SML file is an SML response file to which no SML request file belongs, the 'clientId'
403 element may be omitted; in all other cases, the value for 'client-ID' supplied with
404 'PublicOpen.Reg' must be entered there.

405 The 'refTime' field supplies the reference time for creating the SML response file.

406 **6.1.3 SML_PublicClose.Reg**

407 The SML_PublicClose.Reg must always be present at the end of an SML order file. It
408 terminates this file.

409 Each client must respond to the SML_PublicClose.Reg with an SML_PublicClose.Res or an
410 SML_Attention.Res. The empty 'ServerID' must be considered as broadcast address for the
411 messages which have to be answered. If an application in contrast to this uses special
412 broadcast or multitask addresses these have to be defined explicitly in the application
413 together with the behaviour.

```
414 (I) SML_PublicClose.Reg ::= SEQUENCE
415     {
416         globalSignature SML_Signature OPTIONAL
417     }
```

418 **6.1.4 SML_PublicClose.Res**

419 SML_PublicClose.Res always stands at the end of an SML response file. It terminates this
420 file.

```
421 (J) SML_PublicClose.Res ::= SEQUENCE
422     {
423         globalSignature SML_Signature OPTIONAL
424     }
```

```
425 (K) SML_Signature ::= Octet String
```

426 **6.1.5 SML_Attention.Res**

427 SML_Attention.Res is used in an SML response file to report potentially positive
428 acknowledgements, error messages, warnings or other notes from the server to the client.

429 To enable the (error) messages to be both automatically evaluated and simply outputted as
430 text to the operator, the 'attentionNo' and 'attentionMsg' elements are used.

431 Assignment of globally valid numbers is defined in Section 6.1.5.1.

```

432 (L)   SML_Attention.Res      ::= SEQUENCE
433     {
434         serverId              Octet String,
435         attentionNo           Octet String,
436         attentionMsg          Octet String OPTIONAL,
437         attentionDetails      SML_Tree OPTIONAL
438     }

```

439 6.1.5.1 Global SML Attention Numbers

440 The list of globally defined Error Numbers is provided below, see Section 6.1.5:

441 **Tab. 2: List of global Error Numbers**

Error Number (displayed as a byte chain in hexadecimal form)	Significance
...	... reserved.
81 81 C7 C7 E0 00	Beginning of application-specific Error Numbers
...	Application-specific Error Numbers
81 81 C7 C7 FC FF	End of application-specific Error Numbers
81 81 C7 C7 FD 00	See T
...	See T
81 81 C7 C7 FD FF	See T
81 81 C7 C7 FE 00	Error message that cannot be assigned to any of the meanings defined below.
81 81 C7 C7 FE 01	Unknown SML designator.
81 81 C7 C7 FE 02	Inadequate authentication, user/password combination impermissible.
81 81 C7 C7 FE 03	Destination address ('serverId')not available.
81 81 C7 C7 FE 04	Order ('reqFileId') not available.
81 81 C7 C7 FE 05	One or more destination attribute(s) cannot be written.
81 81 C7 C7 FE 06	One or more destination attribute(s) cannot be read.
81 81 C7 C7 FE 07	Communication with measuring point disturbed.
81 81 C7 C7 FE 08	Raw data cannot be interpreted.
81 81 C7 C7 FE 09	Value supplied is outside the permissible value range.
81 81 C7 C7 FE 0A	Order not executed (for example, because the 'parameterTreePath' supplies indicates an absent element.
81 81 C7 C7 FE 0B	Checksum faulty
81 81 C7 C7 FE 0C	Broadcast not supported
81 81 C7 C7 FE 0D	Unexpected SML message (e.g. an SML file without an Open Request)
81 81 C7 C7 FE 0E	Unknown object in the profile (the OBIS code in the request for a profile refers to a data source not recorded in the profile)
81 81 C7 C7 FE 0F	Data type not supported (e.g. the data type in a SetProcPar.Reg message does not correspond to the expected data type)
81 81 C7 C7 FE 10	Optional element not supported (an element defined in SML as OPTIONAL, which is not supported by the application, has been received as OPTIONAL.)
81 81 C7 C7 FE 11	No entry in requested profile
81 81 C7 C7 FE 12	In profile requests: end limit before begin limit
81 81 C7 C7 FE 13	In profile requests: No entries in requested area. At least one entry in other areas.
81 81 C7 C7 FE 14	An SML file was terminated without SML-Close.

Error Number (displayed as a byte chain in hexadecimal form)	Significance
81 81 C7 C7 FE 15	In profile requests: Temporarily the profile cannot be sent (e.g. due to reordering at request moment or a signature for the profile entry has to be calculated). In general: The response cannot be sent ("busy").
...	... reserved.

442

443 The list of globally defined Attention Numbers is given below, see Section 6.1.5:

444 **Tab. 3: List of global Attention Numbers**

Attention Number (displayed as a byte chain in hexadecimal form)	Significance
81 81 C7 C7 FD 00	OK, positive acknowledgement.
81 81 C7 C7 FD 01	Order will be executed later and result passed to server address per Response-without-Request.
...	... reserved

445

446 **6.1.6 SML_GetCosem Req**447 **05.05.2011: Instead of 3 COSEM services one generic COSEM service with reference on the**
448 **green book could be a solution?**

449

450 Per SML_GetCosem Req the COSEM service ‚Get’ is transmitted. As an answer either an
451 SML_GetCosem Res oder ein SML_Attention Res must be generated. The empty ‚ServerID’
452 must be considered as broadcast address for the messages which have to be answered. If an
453 application in contrast to this uses special broadcast or multitask addresses these have to be
454 defined explicitly in the application together with the behaviour.

455 (M) SML_GetCosem Req ::= SEQUENCE

456 {

457 clientId Octet String,

458 serverId Octet String OPTIONAL,

459 username Octet String OPTIONAL,

460 password Octet String OPTIONAL,

461

462 **→ 05.05.11: Use a reference to the corresponding COSEM service**

463

464 **logicalName** Octet String,

465 classId Integer16,

466 classVersion Integer16,

467 attributeIndexList SML_CosemAttrIndexList OPTIONAL

468 }

469 (N) SML_CosemAttrIndexList ::= SEQUENCE OF
 470 {
 471 attributeDescription SML_CosemAttributeDesc
 472 }
 473 (O) SML_CosemAttributeDesc ::= SEQUENCE
 474 {
 475 attributeIndex Integer16,
 476 selectiveAccessDescriptor SML_CosemSelAccessDesc OPTIONAL
 477 }
 478 (P) SML_CosemSelAccessDesc ::= SEQUENCE OF
 479 {
 480 accessSelector Unsigned8,
 481 accessParameters SML_CosemValue
 482 }
 483 If the element 'attributeIndexList' is not given then all attributes of the addressed object must
 484 be delivered.

485 6.1.7 SML_GetCosem.Res

486 Per SML_GetCosem.Res the answer to a COSEM service ,Get' is transmitted.

487 (Q) SML_GetCosem.Res ::= SEQUENCE
 488 {
 489 clientId Octet String OPTIONAL,
 490 serverId Octet String,
 491 logicalName Octet String,
 492 classId Integer16,
 493 classVersion Integer16,
 494 attributeList SML_CosemAttrList
 495 }
 496 (R) SML_CosemAttrList ::= SEQUENCE OF
 497 {
 498 cosemAttribute SML_CosemAttribute
 499 }
 500 (S) SML_CosemAttribute ::= SEQUENCE
 501 {
 502 attributeDescription SML_CosemAttributeDesc
 503 attributeContent SML_CosemAttributeContent
 504 }
 505 (T) SML_CosemAttributeContent ::= CHOICE
 506 {
 507 data [0x01] SML_CosemValue
 508 dataAccessResult [0x02] Unsigned8,
 509 }

```

510 (U) SML_CosemValueList ::= SEQUENCE OF
511     {
512         cosemValue SML_CosemValue
513     }
514 (V) SML_CosemValue ::= CHOICE
515     {
516         implicit-data-type [0x01] SML_CosemValueSimple,
517         nullData [0x02] Octet String (SIZE(0)),
518         dont-care [0x03] Octet String (SIZE(0)),
519         bit-string [0x04] Octet String
520             (leading '0'-bits are used, if the bit-string
521              length does not fit inside a complete byte),
522         floating-point [0x05] Octet String (SIZE(4)),
523         visible-string [0x06] Octet String,
524         bcd [0x07] Integer8,
525         enum [0x08] Integer8,
526         float32 [0x09] Octet String (SIZE(4)),
527         float64 [0x0A] Octet String (SIZE(8)),
528         date_time [0x0b] SML_Time,
529         date [0x0C] SML_Time,
530         time [0x0d] SML_Time,
531         array [0x0E] SML_CosemValueList,
532         compact-array [0x0F] SML_CosemCompactArray
533     }
534 (W) SML_CosemValueSimple ::= IMPLICIT CHOICE
535     {
536         boolean-Value boolean,
537         byte-List Octet String,
538         integer Integer8,
539         long Integer16,
540         double-long Integer32,
541         long64 Integer64,
542         unsigned Unsigned8,
543         long-unsigned Unsigned16,
544         double-long-unsigned Unsigned32,
545         long64-unsigned Unsigned64,
546         structure SML_CosemValueList
547     }
548 (X) SML_CosemCompactArray ::= SEQUENCE OF
549     {
550         contents-description SML_CosemTypeDef,
551         array-contents Octet String
552     }

```

553	(Y)	SML_CosemTypeDef	::=	CHOICE
554		{		
555		null-data	[0x01]	Octet String (SIZE(0)),
556		array	[0x02]	SML_CosemArrayDef,
557		structure	[0x03]	SML_CosemTypeList,
558		boolean	[0x04]	Octet String (SIZE(0)),
559		bit-string	[0x05]	Octet String (SIZE(0)),
560		double-long	[0x06]	Octet String (SIZE(0)),
561		double-long-unsigned	[0x07]	Octet String (SIZE(0)),
562		floating-point	[0x08]	Octet String (SIZE(0)),
563		octet-string	[0x09]	Octet String (SIZE(0)),
564		visible-string	[0x0A]	Octet String (SIZE(0)),
565		bcd	[0x0b]	Octet String (SIZE(0)),
566		integer	[0x0C]	Octet String (SIZE(0)),
567		long	[0x0d]	Octet String (SIZE(0)),
568		unsigned	[0x0E]	Octet String (SIZE(0)),
569		long-unsigned	[0x0F]	Octet String (SIZE(0)),
570		long64	[0x10]	Octet String (SIZE(0)),
571		long64-unsigned	[0x11]	Octet String (SIZE(0)),
572		enum	[0x12]	Octet String (SIZE(0)),
573		float32	[0x13]	Octet String (SIZE(0)),
574		float64	[0x14]	Octet String (SIZE(0)),
575		date_time	[0x15]	Octet String (SIZE(0)),
576		date	[0x16]	Octet String (SIZE(0)),
577		time	[0x17]	Octet String (SIZE(0)),
578		dont-care	[0x18]	Octet String (SIZE(0)),
579		}		
580	(Z)	SML_CosemArrayDef	::=	SEQUENCE
581		{		
582		array-element		SML_CosemArrayElement
583		}		
584	(AA)	SML_CosemArrayElement	::=	SEQUENCE OF
585		{		
586		number-of-elements		Unsigned16,
587		type-description		SML_CosemTypeDef
588		}		
589	(BB)	SML_CosemTypeList	::=	SEQUENCE
590		{		
591		type-description		SML_CosemTypeDef
592		}		

593 6.1.8 SML_SetCosem.Resq

594 Per SML_SetCosem.Resq the COSEM service 'Set' is transmitted. As an answer either a
595 SML_SetCosem.Res or a SML_Attention.Res must be generated. The empty 'ServerID' must
596 be considered as broadcast address for the messages which have to be answered. If an
597 application in contrast to this uses special broadcast or multitask addresses these have to be
598 defined explicitly in the application together with the behaviour.

```

599 (CC) SML_SetCosem.Req ::= SEQUENCE
600     {
601         clientId          Octet String,
602         serverId          Octet String OPTIONAL,
603         username          Octet String OPTIONAL,
604         password          Octet String OPTIONAL,
605         objName           Octet String,
606         classId           Integer16,
607         classVersion      Integer16,
608         attributeList     SML_CosemAttrList
609     }

```

610 **6.1.9 SML_SetCosem.Res**

611 Per SML_SetCosem.Res the answer to a COSEM service ,Set' is transmitted.

```

612 (DD) SML_SetCosem.Res ::= SEQUENCE
613     {
614         clientId          Octet String OPTIONAL,
615         serverId          Octet String,
616         objName           Octet String,
617         classId           Integer16,
618         classVersion      Integer16,
619         attributeList     SML_CosemAttrList OPTIONAL
620     }

```

621 **6.1.10 SML_ActionCosem.Req**

622 Per SML_ActionCosem.Req the COSEM service ,Execute' is transmitted. As an answer either
623 a SML_ActionCosem.Res or a SML_Attention.Res must be generated. The empty 'ServerID'
624 must be considered as broadcast address for the messages which have to be answered. If an
625 application in contrast to this uses special broadcast or multitask addresses these have to be
626 defined explicitly in the application together with the behaviour.

```

627 (EE) SML_ActionCosem.Req ::= SEQUENCE
628     {
629         clientId          Octet String,
630         serverId          Octet String OPTIONAL,
631         username          Octet String OPTIONAL,
632         password          Octet String OPTIONAL,
633         objName           Octet String,
634         classId           Integer16,
635         classVersion      Integer16,
636         serviceIndex      Unsigned8,
637         serviceParameter  SML_CosemValue OPTIONAL
638     }

```

639 **6.1.11 SML_ActionCosem.Res**

640 Per SML_ActionCosem.Res the answer to a COSEM service ,Execute' is transmitted.

```

641 (FF) SML_ActionCosem.Res ::= SEQUENCE
642     {
643         clientId          Octet String OPTIONAL,
644         serverId          Octet String,
645         objName           Octet String,
646         classId           Integer16,
647         classVersion      Integer16,
648         attributeList     SML_CosemAttrList OPTIONAL
649     }
650

```

651 7 SML binary encoding, directly packed coding

652 SML uses a coding optimised for the goal of producing minimised messages in the data flow.

653 The coding for this purpose is based on the classical type-length-value structure. In contrast
654 to BER, however, it groups together type and length in a value known as the type-length field
655 (“TL-Field”) reduced to a single byte for most application cases.

656 The length field quantifies the number of elements that belong to a simple or complex data
657 type.

658 In the case of simple data types, the length particular corresponds to the number of bytes that
659 belong to the data type concerned.

660 Since the length field is a part of the code, and the TL-Field itself can be regarded as an
661 element, the TL-Field is also counted as a further element in the length particular.

662 In the case of complex data types (such as lists), the length particular corresponds to the
663 number of elements that are grouped together with the complex data type (for example, the
664 number of list entries). The TL-Field itself is not counted here.

665 7.1 Type-length field

666 The TL-Field uses the bit combination in the higher-valued bits of the data word to specify
667 whether, and if so with what significance, further bytes shall be put together with the current
668 byte to form a single word.

669 **Tab. 4: Bit coding in the type-length field for the first byte of a TL-Field particular**

Bit index	MSB, D7	6	5	4	3	2	1	LSB, D0
Used for feature ‘Further byte for TL-follows’	1	X	X	X	X	X	X	X
Used for feature ‘No further byte for TL-Field follows’	0	X	X	X	X	X	X	X
Used for feature ‘Data type Octet String’	X	0	0	0	L	L	L	L
Used for feature ‘Boolean’	0	1	0	0	L	L	L	L
Used for feature ‘Data type integer’	X	1	0	1	L	L	L	L
Used for feature ‘Data type	X	1	1	0	L	L	L	L

Bit index	MSB, D7	6	5	4	3	2	1	LSB, D0
unsigned'								
Used for feature 'Data type List of ...'	X	1	1	1	L	L	L	L
Further byte with space for defining additional data types, currently reserved	1	1	0	0	L	L	L	L
Reserved for future use	X	0	0	1	L	L	L	L
Reserved for future use	X	0	1	0	L	L	L	L
Reserved for future use	X	0	1	1	L	L	L	L

670

671 If a second (and perhaps further) byte(s) follow(s) with a TL-Field particular, the bits for the
672 length information of the TL-Field particular preceding in each case will be shifted to the left
673 and the "new" bits of the following TL-Field particular added coming from the right.

674 For the second and any following additional bytes with a TL-Field particular, the bits for the
675 data type must always be set as follows:

676 **Tab. 5: Bit coding in the type-length field for the second and following TL-Field bytes**

Bit index	MSB, D7	6	5	4	3	2	1	LSB, D0
Used for feature 'Further byte for TL-Field follows'	1	X	X	X	X	X	X	X
Used for feature "No further byte for TL-Field follows'	0	X	X	X	X	X	X	X
Feature 'Use following 4 bits for the length'	X	0	0	0	L	L	L	L
Reserved for future purposes	X	0	0	1	X	X	X	X
Reserved for future purposes	X	0	1	0	X	X	X	X
Reserved for future purposes	X	0	1	1	X	X	X	X
Reserved for future purposes	X	1	X	X	X	X	X	X

677

678 7.2 Coding of data types

679 7.2.1 Data type Octet String

680 An Octet String (a byte chain) with a length of 0 to max. 14 bytes is coded as follows:

681 **Tab. 6: Bit coding in type-length field for an Octet String**

Bit index	MSB, D7	6	5	4	3	2	1	LSB, D0
Octet String with a number from 0 to 14 bytes	0	0	0	0	L	L	L	L

682

683 The TL-Field is followed by the Octet String (the byte chain), where the byte of the byte chain
684 with the index '0' has to follow first behind the TL-Field.

685 If the Octet String contains more than 14 bytes, then in accordance with the description in
686 Section 7.1 further bytes of the TL-Field are inserted before the first byte of the byte chain.

687 7.2.2 Data types Integer8, Integer16, Integer32 and Integer64

688 These integer data types are coded as follows; note that in each data transmission entire
689 bytes that contain leading zeros (in the case of positive numbers) or leading ones (in the case
690 of negative numbers), may be omitted in such a way that no falsification is produced at the
691 recipient:

692	(GG)	Integer8	::= SEQUENCE	
693		{		
694		TL-Field	0x52,	
695		Data value	0xYY	
696		}		
697	(HH)	Integer16	::= SEQUENCE	
698		{		
699		TL-Field	0x53,	
700		Data value	0xYY 0xZZ (0xYY ⇔ High-Byte,	
701			0xZZ ⇔ Low-Byte,	
702			Big Endian representation)	
703		}		
704	(II)	Integer32	::= SEQUENCE	
705		{		
706		TL-Field	0x55,	
707		Data value	0xYY 0xZZ 0xUU 0xVV	(0xYY ⇔ High-Byte,
708				0xVV ⇔ Low-Byte,
709				Big Endian representation)
710		}		
711	(JJ)	Integer64	::= SEQUENCE	
712		{		
713		TL-Field	0x59,	
714		Data value	0xYY ... 0xVV	(0xYY ⇔ High-Byte,
715				0xVV ⇔ Low-Byte,
716				Big Endian representation)
717		}		

718 7.2.3 Data types Unsigned8, Unsigned16, Unsigned32 and Unsigned64

719 These unsigned integer data types are coded as follows; note that in each data transmission
720 entire bytes that contains leading zeroes may be omitted:

721	(KK)	Unsigned8	::= SEQUENCE
722		{	
723		TL-Field	0x62,
724		Data value	0xYY
725		}	

726	(LL)	Unsigned16	::=	SEQUENCE	
727		{			
728		TL-Field		0x63,	
729		Data value		0xYY 0xZZ (0xYY ⇔ High-Byte,	
730				0xZZ ⇔ Low-Byte,	
731				depiction Big Endian)	
732		}			
733	(MM)	Unsigned32	::=	SEQUENCE	
734		{			
735		TL-Field		0x65,	
736		Data value		0xYY 0xZZ 0xUU 0xVV (0xYY ⇔ High-Byte,	
737				0xVV ⇔ Low-Byte,	
738				depiction Big Endian)	
739		}			
740	(NN)	Unsigned64	::=	SEQUENCE	
741		{			
742		TL-Field		0x69,	
743		Data value		0xYY ... 0xVV (0xYY ⇔ High-Byte,	
744				0xVV ⇔ Low-Byte,	
745				depiction Big Endian)	
746		}			

747 7.2.4 Data type Boolean

748 This Boolean data type is coded as follows:

749	(OO)	Boolean	::=	SEQUENCE	
750		{			
751		TL-Field		0x42,	
752		Data value		0xYY (0x00 ⇔ 'false',	
753		}		everything else ⇔ 'true')	

754 7.2.5 Data type List of ...

755 If lists (or arrays or structures) are to be coded, the TL field shall contain at the beginning of
 756 the list/array/structure the number of elements. This is followed by the first element of the
 757 list/array/structure, which likewise again begins with a TL-Field.

758 The element particular in the TL-Field of 'List of ...' always refers to the next TL-Field, which
 759 follows behind all elements of 'List of ...'. The value thus 'leaps' over all TL-Fields that for
 760 each element of 'List of ...' stand inside the 'List of ...' data.

761	(PP)	List of ...	::=	SEQUENCE	
762		{			
763		TL-Field		0x7z ('z' ⇔ Number of elements	
764				in the list)	
765		}			

766 7.3 Coding of special features

767 7.3.1 Feature End of an SML message

768 Due to concatenation of the individual attributes of an SML message, the end of an SML
769 message can be defined/found using the TL-Field:

```
770 (QQ) EndOfSmlMsg ::= 0x00 ⇔ "Entry without
771 TL-Field"
772 ⇔ End
```

773 7.3.2 Feature SEQUENCE

774 In the case of sequences, all components of the sequence are always accepted into the
775 coded data flow in the order in which they are listed in the ASN.1 definition.

776 A sequence is always grouped together as a structure, and thus initiated by the data type 'List
777 of ...' (see Section 7.2.5).

778 7.3.3 Feature CHOICE

779 In the case of selection lists, the TAG is coded to the selected element Unsigned. The TAG
780 contains, just like any other unsigned component, its TL-Field as a prefix.

781 Elements of the type SML_MessageBody code the TAG as Unsigned32, all others as
782 Unsigned8.

783 The CHOICE itself is treated as a structure and is thus initiated by the data type 'List of ...'
784 (see Section 7.2.5). Thus it always leads to a structure with two elements: the first element is
785 the TAG and the second the thereby defined CHOICE element.

786 7.3.4 Feature OPTIONAL

787 If in the ASN.1 definition a component has been designated as 'OPTIONAL', it must be set in
788 the data flow with the TL-Field '0x01'. Note that for this TL-Field the data type 'Octet String' is
789 always assumed, and the element particular positioned directly behind the TL-Field.

790 8 XML - coding

791 Alternatively SML can also be transported per XML coding. Compared with the method
792 described in Section 7, XML coding basically entails a significantly increased data volume, but
793 supplies SML files in standardised readable form.

794 In the following the XML scheme for coding of SML per XML is concerted.

```
795
796 <?xml version="1.0" encoding="utf-8"?>
797 <xs:schema elementFormDefault = "qualified"
798           xmlns:xs = "http://www.w3.org/2001/XMLSchema">
799
800
801 <!-- Simple data types -->
802 <!-- ===== -->
803
804 <xs:simpleType name = "SML_Stamp">
805   <xs:restriction base = "xs:dateTime"/>
```

```

</xs:simpleType>

<xs:simpleType          name = "SML_Signature">
  <xs:restriction       base = "xs:hexBinary"/>
</xs:simpleType>

<xs:simpleType          name = "SML_ObjReqEntry">
  <xs:restriction       base = "xs:hexBinary"/>
</xs:simpleType>

<xs:simpleType          name = "SML_Unit">
  <xs:restriction       base = "xs:unsignedByte"/>
</xs:simpleType>

<!-- Implicite selection data types -->
<!-- ===== -->

<xs:group               name = "SML_Value">
  <xs:choice>
    <xs:element          name = "valBoolean"      type = "xs:boolean"/>
    <xs:element          name = "valByteList"     type = "xs:hexBinary"/>
    <xs:element          name = "valInteger8"     type = "xs:byte"/>
    <xs:element          name = "valInteger16"    type = "xs:short"/>
    <xs:element          name = "valInteger32"    type = "xs:int"/>
    <xs:element          name = "valInteger64"    type = "xs:long"/>
    <xs:element          name = "valUnsigned8"    type = "xs:unsignedByte"/>
    <xs:element          name = "valUnsigned16"   type = "xs:unsignedShort"/>
    <xs:element          name = "valUnsigned32"   type = "xs:unsignedInt"/>
    <xs:element          name = "valUnsigned64"   type = "xs:unsignedLong"/>
  </xs:choice>
</xs:group>

<xs:group               name = "SML_Status">
  <xs:choice>
    <xs:element          name = "status8"         type = "xs:unsignedByte"/>
    <xs:element          name = "status16"        type = "xs:unsignedShort"/>
    <xs:element          name = "status32"        type = "xs:unsignedInt"/>
    <xs:element          name = "status64"        type = "xs:unsignedLong"/>
  </xs:choice>
</xs:group>

<!--Complex data types -->
<!-- ===== -->

<xs:complexType         name = "SML_TimestampLocal">
  <xs:sequence>
    <xs:element          name = "timestamp"       type = "SML_Timestamp"/>
    <xs:element          name = "localOffset"     type = "xs:short"/>
    <xs:element          name = "summerTimeOffset" type = "xs:short"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType         name = "SML_Time">
  <xs:choice>
    <xs:element          name = "secIndex"        type = "xs:unsignedInt"/>
    <xs:element          name = "timestamp"       type = "SML_Timestamp"/>
    <xs:element          name = "localTimestamp"  type = "SML_TimestampLocal"/>
  </xs:choice>
</xs:complexType>

<xs:complexType         name = "SML_ProfObjHeaderEntry">
  <xs:sequence>
    <xs:element          name = "objName"         type = "xs:hexBinary"/>
    <xs:element          name = "unit"           type = "SML_Unit"/>
    <xs:element          name = "scaler"         type = "xs:byte"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType         name = "SML_ProfObjPeriodEntry">
  <xs:sequence>
    <xs:element          name = "valTime"         type = "SML_Time"/>
    <xs:element          name = "status"         type = "xs:unsignedLong"/>
    <xs:element          name = "value_List"     type = "List_of_SML_ValueEntry"/>
    <xs:element          name = "periodSignature" type = "SML_Signature"
      minOccurs = "0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType         name = "SML_ValueEntry">
  <xs:sequence>
    <xs:group            ref = "SML_Value" />
    <xs:element          name = "valueSignature" type = "SML_Signature"
      minOccurs = "0"/>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name = "SML_PeriodEntry">
  <xs:sequence>
    <xs:element name = "objName" type = "xs:hexBinary"/>
    <xs:element name = "unit" type = "SML_Unit"/>
    <xs:element name = "scaler" type = "xs:byte"/>
    <xs:group ref = "SML_Value" />
    <xs:element name = "valueSignature" type = "SML_Signature"
      minOccurs = "0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "SML_ProcParValue">
  <xs:choice>
    <xs:element name = "smlValue">
      <xs:complexType>
        <xs:group ref = "SML_Value"/>
      </xs:complexType>
    </xs:element>
    <xs:element name = "smlPeriodEntry" type = "SML_PeriodEntry"/>
    <xs:element name = "smlTupelEntry" type = "SML_TupelEntry"/>
    <xs:element name = "smlTime" type = "SML_Time"/>
    <xs:element name = "smlListEntry" type = "SML_ListEntry"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name = "SML_TupelEntry">
  <xs:sequence>
    <xs:element name = "serverId" type = "xs:hexBinary"/>
    <xs:element name = "secIndex" type = "SML_Time"/>
    <xs:element name = "status" type = "xs:unsignedLong"/>
    <xs:element name = "unit_pA" type = "SML_Unit"/>
    <xs:element name = "scaler_pA" type = "xs:byte"/>
    <xs:element name = "value_pA" type = "xs:long"/>
    <xs:element name = "unit_R1" type = "SML_Unit"/>
    <xs:element name = "scaler_R1" type = "xs:byte"/>
    <xs:element name = "value_R1" type = "xs:long"/>
    <xs:element name = "unit_R4" type = "SML_Unit"/>
    <xs:element name = "scaler_R4" type = "xs:byte"/>
    <xs:element name = "value_R4" type = "xs:long"/>
    <xs:element name = "signature_pA_R1_R4" type = "xs:hexBinary"/>
    <xs:element name = "unit_mA" type = "SML_Unit"/>
    <xs:element name = "scaler_mA" type = "xs:byte"/>
    <xs:element name = "value_mA" type = "xs:long"/>
    <xs:element name = "unit_R2" type = "SML_Unit"/>
    <xs:element name = "scaler_R2" type = "xs:byte"/>
    <xs:element name = "value_R2" type = "xs:long"/>
    <xs:element name = "unit_R3" type = "SML_Unit"/>
    <xs:element name = "scaler_R3" type = "xs:byte"/>
    <xs:element name = "value_R3" type = "xs:long"/>
    <xs:element name = "signature_mA_R2_R3" type = "xs:hexBinary"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "SML_TreePath">
  <xs:sequence>
    <xs:element name = "path_Entry" type = "xs:hexBinary"
      minOccurs = "1"
      maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "SML_Tree">
  <xs:sequence>
    <xs:element name = "parameterName" type = "xs:hexBinary"/>
    <xs:element name = "parameterValue" type = "SML_ProcParValue"
      minOccurs = "0"
      maxOccurs = "1"/>
    <xs:element name = "child_List" type = "List_of_SML_Tree"
      minOccurs = "0"
      maxOccurs = "1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "SML_ListEntry">
  <xs:sequence>
    <xs:element name = "objName" type = "xs:hexBinary"/>
    <xs:group ref = "SML_Status" minOccurs = "0"/>
    <xs:element name = "valTime" type = "SML_Time"
      minOccurs = "0"/>
    <xs:element name = "unit" type = "SML_Unit"
      minOccurs = "0"/>
    <xs:element name = "scaler" type = "xs:byte"
      minOccurs = "0"/>
    <xs:group ref = "SML_Value"/>
    <xs:element name = "valueSignature" type = "SML_Signature"
      minOccurs = "0"/>
  </xs:sequence>
</xs:complexType>

```

```

<!--List data types -->
<!-- ===== -->

<xs:complexType name = "List_of_SML_ValueEntry">
  <xs:sequence>
    <xs:element name = "value_List_Entry" type = "SML_ValueEntry"
               minOccurs = "0"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "List_of_SML_ProfObjHeaderEntry">
  <xs:sequence>
    <xs:element name = "header_List_Entry" type = "SML_ProfObjHeaderEntry"
               minOccurs = "0"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "List_of_SML_ProfObjPeriodEntry">
  <xs:sequence>
    <xs:element name = "period_List_Entry" type = "SML_ProfObjPeriodEntry"
               minOccurs = "0"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "List_of_SML_PeriodEntry">
  <xs:sequence>
    <xs:element name = "period_List_Entry" type = "SML_PeriodEntry"
               minOccurs = "0"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "List_of_SML_Tree">
  <xs:sequence>
    <xs:element name = "tree_Entry" type = "SML_Tree"
               minOccurs = "1"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "SML_List">
  <xs:sequence>
    <xs:element name = "valListEntry" type = "SML_ListEntry"
               minOccurs = "0"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "List_of_SML_ObjReqEntry">
  <xs:sequence>
    <xs:element name = "object_List_Entry" type = "SML_ObjReqEntry"
               minOccurs = "1"
               maxOccurs = "unbounded"/>
  </xs:sequence>
</xs:complexType>

<!-- SML messages -->
<!-- ===== -->

<!-- SML-Open -->
<!-- ..... -->

<xs:complexType name = "SML_PublicOpen.Res">
  <xs:sequence>
    <xs:element name = "codepage" type = "xs:hexBinary" minOccurs = "0"/>
    <xs:element name = "clientId" type = "xs:hexBinary"/>
    <xs:element name = "reqFileId" type = "xs:hexBinary"/>
    <xs:element name = "serverId" type = "xs:hexBinary" minOccurs = "0"/>
    <xs:element name = "username" type = "xs:hexBinary" minOccurs = "0"/>
    <xs:element name = "password" type = "xs:hexBinary" minOccurs = "0"/>
    <xs:element name = "smlVersion" type = "xs:unsignedByte" minOccurs = "0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name = "SML_PublicOpen.Res">
  <xs:sequence>
    <xs:element name = "codepage" type = "xs:hexBinary" minOccurs = "0"/>
    <xs:element name = "clientId" type = "xs:hexBinary" minOccurs = "0"/>
    <xs:element name = "reqFileId" type = "xs:hexBinary"/>
    <xs:element name = "serverId" type = "xs:hexBinary"/>
    <xs:element name = "refTime" type = "SML_Time" minOccurs = "0"/>
    <xs:element name = "smlVersion" type = "xs:unsignedByte" minOccurs = "0"/>
  </xs:sequence>
</xs:complexType>

```



```

66 <xs:complexType name = "SML_GetProfileList.Res">
67 <xs:sequence>
68 <xs:element name = "serverId" type = "xs:hexBinary"/>
69 <xs:element name = "actTime" type = "SML_Time"/>
70 <xs:element name = "regPeriod" type = "xs:unsignedInt"/>
71 <xs:element name = "parameterTreePath" type = "SML_TreePath"/>
72 <xs:element name = "valTime" type = "SML_Time"/>
73 <xs:element name = "status" type = "xs:unsignedLong"/>
74 <xs:element name = "period_List" type = "List_of_SML_PeriodEntry"/>
75 <xs:element name = "rawdata" type = "xs:hexBinary"
76 minOccurs = "0"/>
77 <xs:element name = "periodSignature" type = "SML_Signature"
78 minOccurs = "0"/>
79 </xs:sequence>
80 </xs:complexType>
81
82 <!-- SML-GetProcParameter -->
83 <!-- ..... -->
84
85 <xs:complexType name = "SML_GetProcParameter Req">
86 <xs:sequence>
87 <xs:element name = "serverId" type = "xs:hexBinary"
88 minOccurs = "0"/>
89 <xs:element name = "username" type = "xs:hexBinary"
90 minOccurs = "0"/>
91 <xs:element name = "password" type = "xs:hexBinary"
92 minOccurs = "0"/>
93 <xs:element name = "parameterTreePath" type = "SML_TreePath"/>
94 <xs:element name = "attribute" type = "xs:hexBinary"
95 minOccurs = "0"/>
96 </xs:sequence>
97 </xs:complexType>
98
99 <xs:complexType name = "SML_GetProcParameter.Res">
100 <xs:sequence>
101 <xs:element name = "serverId" type = "xs:hexBinary"/>
102 <xs:element name = "parameterTreePath" type = "SML_TreePath"/>
103 <xs:element name = "parameterTree" type = "SML_Tree"/>
104 </xs:sequence>
105 </xs:complexType>
106
107 <!-- SML-SetProcParameter -->
108 <!-- (Here only one request is possible) -->
109 <!-- ..... -->
110
111 <xs:complexType name = "SML_SetProcParameter Req">
112 <xs:sequence>
113 <xs:element name = "serverId" type = "xs:hexBinary"
114 minOccurs = "0"/>
115 <xs:element name = "username" type = "xs:hexBinary"
116 minOccurs = "0"/>
117 <xs:element name = "password" type = "xs:hexBinary"
118 minOccurs = "0"/>
119 <xs:element name = "parameterTreePath" type = "SML_TreePath"/>
120 <xs:element name = "parameterTree" type = "SML_Tree"/>
121 </xs:sequence>
122 </xs:complexType>
123
124 <!-- SML-GetList -->
125 <!-- ..... -->
126
127 <xs:complexType name = "SML_GetList Req">
128 <xs:sequence>
129 <xs:element name = "clientId" type = "xs:hexBinary"/>
130 <xs:element name = "serverId" type = "xs:hexBinary"
131 minOccurs = "0" />
132 <xs:element name = "username" type = "xs:hexBinary"
133 minOccurs = "0"/>
134 <xs:element name = "password" type = "xs:hexBinary"
135 minOccurs = "0"/>
136 <xs:element name = "listName" type = "xs:hexBinary"
137 minOccurs = "0"/>
138 </xs:sequence>
139 </xs:complexType>
140
141 <xs:complexType name = "SML_GetList.Res">
142 <xs:sequence>
143 <xs:element name = "clientId" type = "xs:hexBinary"
144 minOccurs = "0"/>
145 <xs:element name = "serverId" type = "xs:hexBinary"/>
146 <xs:element name = "listName" type = "xs:hexBinary"

```

```

256         <xs:element name = "actSensorTime" type = "SML_Time" minOccurs = "0"/>
257         <xs:element name = "valList" type = "SML_List"/>
258         <xs:element name = "listSignature" type = "SML_Signature" minOccurs = "0"/>
259         <xs:element name = "actGatewayTime" type = "SML_Time" minOccurs = "0"/>
260     </xs:sequence>
261 </xs:complexType>
262
263
264
265
266
267
268
269
270
271 <!-- SML-Attention -->
272 <!-- (Here only one response is possible) -->
273 <!-- ..... -->
274
275 <xs:complexType name = "SML_Attention.Res">
276 <xs:sequence>
277 <xs:element name = "serverId" type = "xs:hexBinary"/>
278 <xs:element name = "attentionNo" type = "xs:hexBinary"/>
279 <xs:element name = "attentionMsg" type = "xs:string" minOccurs = "0"/>
280 <xs:element name = "attentionDetails" type = "SML_Tree" minOccurs = "0"/>
281 </xs:sequence>
282 </xs:complexType>
283
284
285
286
287
288
289
290
291 <!-- SML-Message-Body -->
292 <!-- ..... -->
293
294 <xs:complexType name = "SML_Message">
295 <xs:sequence>
296 <xs:element name = "transactionId" type = "xs:hexBinary"/>
297 <xs:element name = "groupNo" type = "xs:unsignedByte"/>
298 <xs:element name = "abortOnError" type = "xs:unsignedByte"/>
299 <xs:choice>
300 <xs:element name = "OpenRequest" type = "SML_PublicOpen.Req"/>
301 <xs:element name = "OpenResponse" type = "SML_PublicOpen.Res"/>
302 <xs:element name = "CloseRequest" type = "SML_PublicClose.Req"/>
303 <xs:element name = "CloseResponse" type = "SML_PublicClose.Res"/>
304 <xs:element name = "GetProfilePackRequest" type = "SML_GetProfilePack.Req"/>
305 <xs:element name = "GetProfilePackResponse" type = "SML_GetProfilePack.Res"/>
306 <xs:element name = "GetProfileListRequest" type = "SML_GetProfileList.Req"/>
307 <xs:element name = "GetProfileListResponse" type = "SML_GetProfileList.Res"/>
308 <xs:element name = "GetProcParameterRequest" type = "SML_GetProcParameter.Req"/>
309 <xs:element name = "GetProcParameterResponse" type = "SML_GetProcParameter.Res"/>
310 <xs:element name = "SetProcParameterRequest" type = "SML_SetProcParameter.Req"/>
311 <xs:element name = "GetListRequest" type = "SML_GetList.Req"/>
312 <xs:element name = "GetListResponse" type = "SML_GetList.Res"/>
313 <xs:element name = "GetCossemRequest" type = "SML_GetCossem.Req"/>
314 <xs:element name = "GetCossemResponse" type = "SML_GetCossem.Res"/>
315 <xs:element name = "SetCossemRequest" type = "SML_SetCossem.Req"/>
316 <xs:element name = "SetCossemResponse" type = "SML_SetCossem.Res"/>
317 <xs:element name = "RunCossemRequest" type = "SML_RunCossem.Req"/>
318 <xs:element name = "RunCossemResponse" type = "SML_RunCossem.Res"/>
319 <xs:element name = "AttentionResponse" type = "SML_Attention.Res"/>
320 </xs:choice>
321 </xs:sequence>
322 </xs:complexType>
323
324
325
326
327 <!-- SML file -->
328 <!-- ===== -->
329
330 <xs:complexType name = "SML_File">
331 <xs:sequence>
332 <xs:element name = "SmlMessage" type = "SML_Message" minOccurs = "1" maxOccurs = "unbounded"/>
333 </xs:sequence>
334 </xs:complexType>
335
336
337
338
339
340
341 <!-- XML root elements -->
342 <!-- ===== -->
343
344 <xs:complexType name = "SML_XML_Body_V_1_04">
345 <xs:choice>
346 <xs:element name = "smlFile" type = "SML_File"/>
347 <xs:element name = "smlMessage" type = "SML_Message"/>
348 </xs:choice>
349 </xs:complexType>

```

```

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

433 <xs:element name = "password" type = "xs:hexBinary"
434 minOccurs = "0"/>
435 <xs:element name = "objName" type = "xs:hexBinary"/>
436 <xs:element name = "classId" type = "xs:short"/>
437 <xs:element name = "classVersion" type = "xs:short"/>
438 <xs:element name = "serviceIndex" type = "xs:byte"/>
439 <xs:element name = "attributeList" type = "SML_CosemAttrList"
440 minOccurs = "0"/>
441 </xs:sequence>
442 </xs:complexType>
443
444 <xs:complexType name = "SML_RunCosem.Res">
445 <xs:sequence>
446 <xs:element name = "clientId" type = "xs:hexBinary"
447 minOccurs = "0"/>
448 <xs:element name = "serverId" type = "xs:hexBinary"/>
449 <xs:element name = "objName" type = "xs:hexBinary"/>
450 <xs:element name = "classId" type = "xs:short"/>
451 <xs:element name = "classVersion" type = "xs:short"/>
452 <xs:element name = "attributeList" type = "SML_CosemAttrList"
453 minOccurs = "0"/>
454 </xs:sequence>
455 </xs:complexType>
456
457 <!--COSEM attributes -->
458 <!-- ..... -->
459
460 <xs:simpleType name = "SML_CosemAttrIndexList">
461 <xs:list itemType = "xs:short"/>
462 </xs:simpleType>
463
464 <xs:complexType name = "SML_CosemAttrList">
465 <xs:sequence>
466 <xs:element name = "SML_CosemAttr" minOccurs = "0"
467 maxOccurs = "unbounded"/>
468 </xs:sequence>
469 </xs:complexType>
470
471 <xs:complexType name = "SML_CosemAttr">
472 <xs:sequence>
473 <xs:element name = "attributeIndex" type = "xs:short"/>
474 <xs:element name = "attributeContent" type = "SML_CosemValue"/>
475 </xs:sequence>
476 </xs:complexType>
477
478 <xs:complexType name = "SML_CosemValue">
479 <xs:choice>
480 <xs:element name = "valBoolean" type = "xs:boolean"/>
481 <xs:element name = "valByteList" type = "xs:hexBinary"/>
482 <xs:element name = "valInteger8" type = "xs:byte"/>
483 <xs:element name = "valInteger16" type = "xs:short"/>
484 <xs:element name = "valInteger32" type = "xs:int"/>
485 <xs:element name = "valInteger64" type = "xs:long"/>
486 <xs:element name = "valUnsigned8" type = "xs:unsignedByte"/>
487 <xs:element name = "valUnsigned16" type = "xs:unsignedShort"/>
488 <xs:element name = "valUnsigned32" type = "xs:unsignedInt"/>
489 <xs:element name = "valUnsigned64" type = "xs:unsignedLong"/>
490 <xs:element name = "valList" type = "SML_CosemList"/>
491 <xs:element name = "valStruct" type = "SML_CosemList"/>
492 <xs:element name = "valArray" type = "SML_CosemList"/>
493 </xs:choice>
494 </xs:complexType>
495
496 <xs:complexType name = "SML_CosemList">
497 <xs:sequence>
498 <xs:element name = "SML_CosemListEntry" minOccurs = "0"
499 maxOccurs = "unbounded"/>
500 </xs:sequence>
501 </xs:complexType>
502
503 <xs:complexType name = "SML_CosemListEntry">
504 <xs:choice>
505 <xs:element name = "valBoolean" type = "xs:boolean"/>
506 <xs:element name = "valByteList" type = "xs:hexBinary"/>
507 <xs:element name = "valInteger8" type = "xs:byte"/>
508 <xs:element name = "valInteger16" type = "xs:short"/>
509 <xs:element name = "valInteger32" type = "xs:int"/>
510 <xs:element name = "valInteger64" type = "xs:long"/>
511 <xs:element name = "valUnsigned8" type = "xs:unsignedByte"/>
512 <xs:element name = "valUnsigned16" type = "xs:unsignedShort"/>
513 <xs:element name = "valUnsigned32" type = "xs:unsignedInt"/>
514 <xs:element name = "valUnsigned64" type = "xs:unsignedLong"/>
515 </xs:choice>
516 </xs:complexType>
517
518 </xs:schema>

```

1523 Fig. 4: XML scheme file for coding of SML per XML
1524

1525 **Annex A (normative)**1526 **A.1 SML Messages for non COSEM environments**1527 **A.1.1 SML_GetProfilePack.Req**1528 **Deprecated: not to be used in a COSEM environment**

1529 In an SML order file one or more SML_GetProfilePack messages can be present.

1530 Each SML_GetProfilePack.Req serves to request individual measured values or measured-
1531 value lists, which are transmitted in packed form.

1532 The client must respond to each SML_GetProfilePack.Req with precisely one
1533 SML_GetProfilePack.Res or one SML_Attention.Res. The empty 'ServerID' must be
1534 considered as broadcast address for the messages which have to be answered. If an
1535 application in contrast to this uses special broadcast or multitask addresses these have to be
1536 defined explicitly in the application together with the behaviour.

```

1537 (RR)  SML_GetProfilePack.Req ::= SEQUENCE
1538      {
1539          serverId          Octet String OPTIONAL,
1540          username          Octet String OPTIONAL,
1541          password          Octet String OPTIONAL,
1542          withRawdata       Boolean OPTIONAL,
1543          beginTime         SML_Time OPTIONAL,
1544          endTime           SML_Time OPTIONAL,
1545          parameterTreePath SML_TreePath,
1546          object_List       List_of_SML_ObjReqEntry OPTIONAL,
1547          dasDetails        SML_Tree OPTIONAL
1548      }
1549 (SS)  List_of_SML_ObjReqEntry ::= SEQUENCE OF
1550      {
1551          object_List_Entry SML_ObjReqEntry
1552      }

```

```

1553 (TT)  SML_ObjReqEntry ::= Octet String

```

1554 The 'parameterTreePath' element designates the measured-value list and thus serves for
1555 classification into variants like logbook, load profile or comparable. It must be occupied with at
1556 least one Code Number {for coding, see Paragraphs (LLL) and 0}.

1557 The 'object_List' is required to enable the desired channel for reading off load profiles to be
1558 stated with the order. If this element is not stated or empty, then all channels included in the
1559 profile must be stated in the response.

1560 The desired target range is stated using time limits.

- 1561 The following agreement shall apply if time limits are defined per seconds index⁵:
- 1562 The seconds index with the value '00000000' always describes the value lying temporally
1563 furthest in the past.
- 1564 The seconds index with the value 'FFFFFFFF' always describes the 'current entry', meaning
1565 the value lying temporally directly before the ongoing time.
- 1566 All values between '00000000' and 'FFFFFFFF' are offsets to be counted from the beginning
1567 ('00000000').
- 1568 If in the case of range queries the desired seconds index is not available, then it must be
1569 used in the response:
1570
1571 ... if present for the beginning of a range,
1572 always the value lying temporally directly afterwards⁶ or
1573 ... if this is not present,
1574 the 'current entry'
1575
1576 ... if present for the end of a range,
1577 always the value lying temporally before it⁷ or
1578 ... if this is not present,
1579 the 'current entry'
1580
1581 .
- 1582 If 'beginTime' is not stated, the value '00000000' must be assumed.
- 1583 If 'endTime' is not stated, the value 'FFFFFFFF' must be assumed.
- 1584 Per 'dasDetails' parameters required as supplement for data procurement can be transmitted
1585 at need. These are transmitted as a tree structure - see Section A.1.6.
- 1586 **A.1.2 SML_GetProfilePack.Res**
- 1587 **Deprecated: not to be used in a COSEM environment**
- 1588 In an SML response file, one or more SML_GetProfilePack.Res messages can be present.
- 1589 Each SML data source supplies precisely one SML_GetProfilePack.Res response to one
1590 SML_GetProfilePack.Req request.

⁵ The duty environment at present provides solely for use of the seconds index; see the explanation on 'SML_Time'. Possible future expansion may designate further alternatives for this.

⁶ Towards larger indices.

⁷ Towards smaller indices.

1591 (UU) SML_GetProfilePack.Res ::= SEQUENCE
1592 {
1593 serverId Octet String,
1594 actTime SML_Time,
1595 regPeriod Unsigned32,
1596 parameterTreePath SML_TreePath,
1597 header_List List_of_SML_ProfObjHeaderEntry,
1598 period_List List_of_SML_ProfObjPeriodEntry,
1599 rawdata Octet String OPTIONAL,
1600 profileSignature SML_Signature OPTIONAL
1601 }

1602 (VV) List_of_SML_ProfObjHeaderEntry ::= SEQUENCE OF
1603 {
1604 header_List_Entry SML_ProfObjHeaderEntry
1605 }

1606 (WW) SML_ProfObjHeaderEntry ::= SEQUENCE
1607 {
1608 objName Octet String,
1609 unit SML_Unit,
1610 scaler Integer8
1611 }

1612 (XX) List_of_SML_ProfObjPeriodEntry ::= SEQUENCE OF
1613 {
1614 period_List_Entry SML_ProfObjPeriodEntry
1615 }

1616 (YY) SML_ProfObjPeriodEntry ::= SEQUENCE
1617 {
1618 valTime SML_Time,
1619 status Unsigned64,
1620 value_List List_of_SML_ValueEntry,
1621 periodSignature SML_Signature OPTIONAL
1622 }

1623 (ZZ) List_of_SML_ValueEntry ::= SEQUENCE OF
1624 {
1625 value_List_Entry SML_ValueEntry
1626 }

1627 (AAA) SML_ValueEntry ::= SEQUENCE
1628 {
1629 value SML_Value,
1630 valueSignature SML_Signature OPTIONAL
1631 }

```

1632 (BBB) SML_Value ::= IMPLICIT CHOICE
1633     {
1634         boolean-Value          boolean,
1635         byte-List              Octet String8,
1636         8-Bit-Integer          Integer8,
1637         16-Bit-Integer         Integer16,
1638         32-Bit-Integer         Integer32,
1639         64-Bit-Integer         Integer64,
1640         8-Bit-Unsigned        Unsigned8,
1641         16-Bit-Unsigned        Unsigned16,
1642         32-Bit-Unsigned        Unsigned32,
1643         64-Bit-Unsigned        Unsigned64
1644         smlList                SML_ListType
1645     }

```

```

1646 (CCC) SML_Unit ::= Unsigned8

```

1647 For numerical values, see DLMS-Unit list, to be found, for example, in IEC 62056-62.
1648

1649 The number of elements in the 'header_List' must always be identical to the number of
1650 elements in the list of registration periods ('value_List').

1651 The arrangement of registration periods in the list is based on the pattern "the value with the
1652 smallest second index comes first".

1653 The 'actTime' element supplies the time information that was present at the data source at the
1654 moment when execution of the order began.

1655 The 'valTime' element supplies the time information that was present at the data source sat
1656 the moment when the measured value was formed.

1657 The 'regPeriod' element specifies the duration of the registration period being used. The value
1658 is stated in seconds. If the recording involved is an event-oriented profile, (i.e. no specific
1659 registration period is on file), '0' must be used as the value.

1660 Use of the two mutually independent lists ('header_List' and 'period_List') has been chosen to
1661 be able to omit multiple acquisition of redundant information, such as seconds index and
1662 status at the moment of measured-value formation for each registration period in case of
1663 several load profiles ("A", "-A", ...).

1664 Use of either 'profileSignature', 'periodSignature' or 'valueSignature' enables entire load
1665 profiles, individual measured values and also tuples of entire registration periods to be jointly
1666 protected. Which approach is adopted will depend on the application involved.

1667 The 'scaler' field is used to establish the relationship between the unit and the numerical
1668 value as follows:

```

1669     numerical value = SML_Value x 10scaler
1670

```

⁸ Note: It is allowed to use an Octet String of length '0'.

1671 Per 'smlList' in an SML_Value the following data structures can be transported:

```

1672 (DDD) SML_ListType ::= CHOICE
1673 {
1674     smlTime [0x01] SML_Time
1675     smlTimestampedValue [0x02] SML_TimestampedValue
1676 }
1677

```

```

1678 (EEE) SML_TimestampedValue ::= SEQUENCE
1679 {
1680     smlTime SML_Time
1681     status SML_Status
1682     simpleValue SML_SimpleValue
1683 }

```

```

1684 (FFF) SML_SimpleValue ::= IMPLICIT CHOICE
1685 {
1686     boolean-Value boolean,
1687     byte-List Octet String8,
1688     8-Bit-Integer Integer8,
1689     16-Bit-Integer Integer16,
1690     32-Bit-Integer Integer32,
1691     64-Bit-Integer Integer64,
1692     8-Bit-Unsigned Unsigned8,
1693     16-Bit-Unsigned Unsigned16,
1694     32-Bit-Unsigned Unsigned32,
1695     64-Bit-Unsigned Unsigned64
1696 }
1697

```

1698 A.1.3 SML_GetProfileList.Req

1699 **Deprecated: not to be used in a COSEM environment**

1700 In an SML order file, one or more SML_GetProfileList messages can be present.

1701 Each SML_GetProfileList.Req serves to request individual measured values or lists of
1702 measured values, which are transmitted in simple list form. In contrast to
1703 SML_GetProfilePack, where the responses are as far as possible transmitted without
1704 redundancy and optimally packed, SML_GetProfile expects simple lists. These offer the
1705 disadvantage of generating substantially more volume when transmitting daily load profiles,
1706 but in return provide the advantage of a simple structure, which becomes significantly more
1707 efficient in use with systems featuring quarter-hourly data procurement.

1708 The client can respond to each SML_GetProfileList.Req with one or more
1709 SML_GetProfileList.Res or one SML_Attention.Res. The empty 'ServerID' must be considered
1710 as broadcast address for the messages which have to be answered. If an application in
1711 contrast to this uses special broadcast or multitask addresses these have to be defined
1712 explicitly in the application together with the behaviour.

1713 (GGG) SML_GetProfileList.Req ::= SEQUENCE
 1714 {
 1715 serverId Octet String OPTIONAL,
 1716 username Octet String OPTIONAL,
 1717 password Octet String OPTIONAL,
 1718 withRawdata Boolean OPTIONAL,
 1719 beginTime SML_Time OPTIONAL,
 1720 endTime SML_Time OPTIONAL,
 1721 parameterTreePath SML_TreePath,
 1722 object_List List_of_SML_ObjReqEntry OPTIONAL,
 1723 dasDetails SML_Tree OPTIONAL
 1724 }
 1725 For the elements relating to SML_GetProfileList.Req, the stipulations defined under
 1726 SML_GetProfilePack.Req apply.

1727 **A.1.4 SML_GetProfileList.Res**

1728 **Deprecated: not to be used in a COSEM environment**

1729 In an SML response file, one or more SML_GetProfileList.Res messages can be present.

1730 Each SML data source supplies one or more SML_GetProfileList.Res responses to an
 1731 SML_GetProfileList.Req order. Each SML_GetProfileList.Res response contains the
 1732 measured value of a registration period (thus usually of a quarter of an hour) in the sense of
 1733 an atom.

1734 (HHH) SML_GetProfileList.Res ::= SEQUENCE
 1735 {
 1736 serverId Octet String,
 1737 actTime SML_Time,
 1738 regPeriod Unsigned32,
 1739 parameterTreePath SML_TreePath,
 1740 valTime SML_Time,
 1741 status Unsigned64,
 1742 period_List List_of_SML_PeriodEntry,
 1743 rawdata Octet String OPTIONAL,
 1744 periodSignature SML_Signature OPTIONAL
 1745 }
 1746 (III) List_of_SML_PeriodEntry ::= SEQUENCE OF
 1747 {
 1748 period_List_Entry SML_PeriodEntry
 1749 }

```

1750 (JJJ) SML_PeriodEntry ::= SEQUENCE
1751     {
1752         objName          Octet String,
1753         unit              SML_Unit,
1754         scaler            Integer8,
1755         value             SML_Value,
1756         valueSignature   SML_Signature OPTIONAL
1757     }

```

1758 Use of either 'periodSignature' or 'valueSignature' enables both individual measured values
1759 and also tuples of entire registration periods to be jointly protected. Which approach is
1760 adopted will depend on the application involved.

1761 **A.1.5 SML_GetProcParameter Req**

1762 **Deprecated: not to be used in a COSEM environment**

1763 SML_GetProcParameter Req enable operating parameters (modem parameters, protocol
1764 parameters, capacity utilisation of software modules, ...) to be retrieved in an SML order file.

1765 The client responds to this message per SML_GetProcParameter.Res or SML_Attention.Res
1766 in the response file. The empty 'ServerID' must be considered as broadcast address for the
1767 messages which have to be answered. If an application in contrast to this uses special
1768 broadcast or multitask addresses these have to be defined explicitly in the application
1769 together with the behaviour.

1770 An individual parameter is always defined by its OBIS Code Number. It is from the OBIS Code
1771 Number that the specific data type is implicitly derived, and the value range for the result to
1772 be supplied. The specific assignment and permissible OBIS Code Numbers and their
1773 significance can be taken for the appendix. Within the coding of an SML message, the data
1774 types are also coded, so that the generic implementation for converting the parameters into
1775 internal programs is easily possible.

1776 Since the structure and number of the parameters required for a specific application case is
1777 unknown beforehand, but continued modification of the SML specification to new
1778 requirements is to be avoided if possible, the 'SML_..._ProcParameter' messages use a tree
1779 structure, so that parameter trees of any desired configuration can be transported.

```

1780 (KKK) SML_GetProcParameter Req ::= SEQUENCE
1781     {
1782         serverId          Octet String OPTIONAL,
1783         username          Octet String OPTIONAL,
1784         password          Octet String OPTIONAL,
1785         parameterTreePath SML_TreePath
1786         attribute         Octet String OPTIONAL,
1787     }

```

```

1788 (LLL) SML_TreePath ::= SEQUENCE OF
1789     {
1790         path_Entry       Octet String
1791     }

```

1792

1793 SML_TreePath enables parameters to be selectively addressed within a parameter tree. Note
 1794 that an SML_TreePath always begins from the root element of the parameter tree. The
 1795 meaning (corresponding to the address) of the root element is given with its parameter name.

1796 Thanks to the list structure, the parameter name of the root element can always be found in
 1797 that SML_TreePath element that can be found as the first and thus directly under
 1798 'SML_GetProcParameter.Req'.

1799 The 'attribute' characteristic can at need be also provided with a restriction/detailed
 1800 information for narrowing the desired request result.

1801 **A.1.6 SML_GetProcParameter.Res**

1802 **Deprecated: not to be used in a COSEM environment**

1803 SML_GetProcParameter.Res is used in an SML response file for acknowledging the order
 1804 SML_GetProcParameter.Req.

```

1805 (MMM) SML_GetProcParameter.Res ::= SEQUENCE
1806     {
1807         serverId          Octet String,
1808         parameterTreePath SML_TreePath,
1809         parameterTree     SML_Tree
1810     }
1811 (NNN) SML_Tree ::= SEQUENCE
1812     {
1813         parameterName     Octet String,
1814         parameterValue    SML_ProcParValue OPTIONAL,
1815         child_List        List_of_SML_Tree OPTIONAL
1816     }
1817 (OOO) SML_ProcParValue ::= CHOICE
1818     {
1819         smIValue           [0x01] SML_Value9,
1820         smIPeriodEntry     [0x02] SML_PeriodEntry,
1821         smITupleEntry      [0x03] SML_TupleEntry,
1822         smITime            [0x04] SML_Time
1823         smIListEntry       [0x05] SML_ListEntry
1824     }
  
```

⁹ Note: Applications shall define if contents of the type Typ SML_Time shall be transmitted directly per choice '0x04' or indirectly per content of SML_Value.

```

1825 (PPP) SML_TupleEntry ::= SEQUENCE
1826     {
1827         serverId      Octet String,
1828         secIndex      SML_Time,
1829         status        Unsigned64,
1830         unit_pA       SML_Unit,
1831         scaler_pA     Integer8,
1832         value_pA      Integer64,
1833         unit_R1       SML_Unit,
1834         scaler_R1    Integer8,
1835         value_R1     Integer64,
1836         unit_R4      SML_Unit,
1837         scaler_R4    Integer8,
1838         value_R4     Integer64,
1839         signature_pA_R1_R4 Octet String,
1840         unit_mA      SML_Unit,
1841         scaler_mA    Integer8,
1842         value_mA     Integer64,
1843         unit_R2      SML_Unit,
1844         scaler_R2    Integer8,
1845         value_R2     Integer64,
1846         unit_R3      SML_Unit,
1847         scaler_R3    Integer8,
1848         value_R3     Integer64,
1849         signature_mA_R2_R3 Octet String
1850     }

1851 (QQQ) List_of_SML_Tree ::= SEQUENCE OF
1852     {
1853         tree_Entry      SML_Tree
1854     }

```

1855 'SML_Tree' enables individual parameters (leaves or nodes) to be built up with their (in the
1856 case of nodes) following children (see 'child_List'). Specifically, this means an 'SML_Tree'
1857 can thus be used to image ...
1858 ... an individual parameter,
1859 ... a node with a dependent list of further parameters or
1860 ... a node with a dependent list of further subtrees.

1861 The 'parameterValue' and 'child_List' elements are both optional, so that the following
1862 variants can be imaged:

- 1863 – a parameter consisting of name (coded per OBIS Code Number) and value;
- 1864 – a parameter consisting of name (coded per OBIS Code Number) and subtree or list of
1865 further parameters
- 1866 – a parameter consisting of name (coded per OBIS Code Number), value and subtree or list
1867 of further parameters.

1868 Whether an 'SML_Tree' is to be evaluated as a leaf or node is specified by the presence of
1869 the 'child_List' element. If an 'SML_Tree' has the 'child_List' element, it is a node; if not, it is
1870 a leaf.

1871 **A.1.7 SML_SetProcParameter Req**1872 **Deprecated: not to be used in a COSEM environment**

1873 SML_SetProcParameter Req enables operating parameters in an SML order file (for example,
1874 the baudrate for accessing a terminal) to be transmitted.

1875 For structuring, please consult the notes in Section A.1.5.

```

1876 (RRR) SML_SetProcParameter Req ::= SEQUENCE
1877     {
1878         serverId          Octet String OPTIONAL,
1879         username          Octet String OPTIONAL,
1880         password         Octet String OPTIONAL,
1881         parameterTreePath SML_TreePath,
1882         parameterTree     SML_Tree
1883     }

```

1884 The client responds to this message per SML_Attention.Res. The empty 'ServerID' must be
1885 considered as broadcast address for the messages which have to be answered. If an
1886 application in contrast to this uses special broadcast or multitask addresses these have to be
1887 defined explicitly in the application together with the behaviour.

1888 **A.1.8 SML_GetList Req**1889 **Deprecated: not to be used in a COSEM environment**

1890 Per SML_GetList Req a parameterized data list laid down in the server can be retrieved. As
1891 an answer an SML_GetList.Res or an SML_Attention.Res must be generated. The empty
1892 'ServerID' must be considered as broadcast address for the messages which have to be
1893 answered. If an application in contrast to this uses special broadcast or multitask addresses
1894 these have to be defined explicitly in the application together with the behaviour.

```

1895 (SSS) SML_GetList Req ::= SEQUENCE
1896     {
1897         clientId          Octet String,
1898         serverId         Octet String OPTIONAL,
1899         username         Octet String OPTIONAL,
1900         password         Octet String OPTIONAL,
1901         listName         Octet String OPTIONAL
1902     }

```

1903 Per attribute 'listName' the desired list can be named. It defines per OBIS the desired quantity
1904 / list. Since this element is denoted as „optional“ simple devices can omit its interpretation and
1905 send only the meter reading whose meaning they have to give per OBIS code in the answer.

1906 **A.1.9 SML_GetList Res**1907 **Deprecated: not to be used in a COSEM environment**

1908 Per SML_GetList.Res a list of predefined data can be transmitted.

```

1909 (TTT) SML_GetList.Res ::= SEQUENCE
1910     {
1911         clientId          Octet String OPTIONAL,
1912         serverId          Octet String,
1913         listName          Octet String OPTIONAL,
1914         actSensorTime     SML_Time OPTIONAL,
1915         valList           SML_List,
1916         listSignature     SML_Signature OPTIONAL,
1917         actGatewayTime    SML_Time OPTIONAL
1918     }

```

1919 With the optional attribute ,actSensorTime' a sensor which generates this SML message can
 1920 attach its own current time information.

1921 With the optional attribute ,actGatewayTime' a gateway which transports this SML message
 1922 can attach its own current time information.

```

1923 (UUU) SML_List ::= SEQUENCE OF
1924     {
1925         valListEntry      SML_ListEntry
1926     }
1927 (VVV) SML_ListEntry ::= SEQUENCE
1928     {
1929         objName           Octet String,
1930         status            SML_Status OPTIONAL,
1931         valTime           SML_Time OPTIONAL,
1932         unit              SML_Unit OPTIONAL,
1933         scaler            Integer8 OPTIONAL,
1934         value             SML_SimpleValue,
1935         valueSignature    SML_Signature OPTIONAL
1936     }

```

```

1937 (WWW) SML_Status ::= IMPLICIT CHOICE
1938     {
1939         status8           Unsigned8
1940         status16         Unsigned16
1941         status32         Unsigned32
1942         status64         Unsigned64
1943     }

```

1944

Annex B (informative)

1945

B.1 SML Transport Protocol (CHAPTER has to move out ...)

1946

B.1.1 Version 1

1947

1948

1949

For transmitting SML messages over unsecured links, such as direct optical readout per MED on the spot, readout via PSTN modem, GSM modem or comparable routes, the SML Transport Protocol is defined.

1950

It can also be used with secure links (such as TCP).

1951

1952

As a streaming protocol, it follows the approach of coding the data traffic "on the fly" in the terminals, thus enabling it to dispense with the use of extensive buffers.

1953

The rules are is defined as follows:

1954

– Beginning, end and other features of a message are identified using escape sequences.

1955

1956

– An escape sequence is initiated with an escape string. This escape string is followed by the feature added to the message (beginning, end, ...).

1957

1958

1959

– If the escape string occurs at any position in the payload stream itself, this will itself be transmitted as an escape sequence. In this case, the feature added to the message is the escape string (it is thus transmitted twice in succession).

1960

1961

1962

– The number of bytes in the payload stream is always extended at the end to a quantity divisible by the number of bytes in the escape string without leaving a remainder¹⁰. For this extension, bytes with the content '00' (hex) are used.

1963

As an escape string, the byte string '1b 1b 1b 1b' (stated in hex.) is specified.

1964

The following escape features have been defined:

1965

Tab. 7: Escape features for the SML Transport Protocol

Pos.	Escape feature (stated in hex)	Significance / Note
1	1b 1b 1b 1b	Designates the case that the escape sequence is itself contained in the payload stream.
2	01 01 01 01	Initiates a transmission of version 1 as data flow. Designates the feature 'Beginning of a message'.
3	02 TT UU VV	Initiates a transmission of version 2 as block transfer, see chapter B.1.2.
4	03 00 RR RR	Used only in connection with version 2, defines the timeout to be used - see chapter B.1.2.
5	04 00 SS SS	Used only in connection with version 2, defines the block size to be used - see chapter B.1.2.
6	1a XX YY ZZ	Designates the feature "End of a message". 'XX' ⇔ Can assume values in the range '00', '01', '02' or '03' and supplies the number of bytes that have been added at the end of the useful load to enable the number of bytes in the useful load to be divided by the number of bytes in an escape string without

¹⁰ Since the escape string consists of 4 bytes, the number of bytes in the useful load must always likewise be increased in such a way that a division modulo 4 supplies precisely 0. This means the sender must add either no byte, or one, two or three bytes.

Pos.	Escape feature (stated in hex)	Significance / Note
		any remainder. 'YY ZZ' ⇔ Can assume values in the range '00..FF' and contains the checksum for the entire message. YY is here the Most Significant Byte and ZZ the Least Significant Byte of the checksum.
7	All other combinations	Reserved for future expansions.

1966

1967 The checksum must be calculated in accordance with CCITT-CRC16. It is calculated using all
1968 bytes of the data flow in the SML Transport Protocol with the exception of the last two bytes
1969 (and thus without the bytes of the checksum itself).

1970 Calculation is performed in accordance with IEC 62056-46.

1971 If data procurement is performed over TCP or UDP links, the SML Transport Protocol is used
1972 for identifying self-contained SML files.

1973 B.1.2 Version 2

1974 For the SML file transmission per unsecured half-duplex connections the SML transport
1975 protocol of version 2 can be used. This yields a simple mechanism for flow control so allowing
1976 the sender the adaptation of his outputs to the requirements of the clients having possibly
1977 only small resources.

1978 The SML transport protocol version 2 therefore also must be used if SML files are transferred
1979 per optical ways comparable to IEC 62056-21.

1980 The concept which was realised to fulfil these requirements divides the SML file to be
1981 transferred into blocks and uses timeouts for the restart after errors. The block sizes as well
1982 the timeouts are negotiated between sender and receiver at the beginning of the
1983 transmission. As timeout value and as minimum block size are used:

1984 Initially to use timeout: 5 s;

1985 Initially to use block size: 32 Bytes.

1986 Compared with version 1 the version 2 uses three additionally defined ESC sequences.

1987 B.1.3 Initiating of the transmission in version 2

1988 To distinguish version 2 of the SML transport protocol from version 1 (see chapter B.1.1) here
1989 the sequence

1990 1b 1b 1b 1b 02 TT UU VV

1991 is used as start sequence (all data in hex, see Tab. 7).

1992 The elements „TT UU VV“ serve as labelling of blocks and SML files.

1993 B.1.4 Labelling of blocks

1994 Blocks are labelled per element „TT“, see Tab. 7. This labelling uses the following features:

1995 – Bit 7, MSB: 0 ⇔ sent block,

1996 1 ⇔ received as ACK;

1997 – Bit 6: 0 ⇔ further blocks following,

1998 1 ⇔ last block of the SML file.

1999 – Bit 5 ... Bit 0, LSB: Block number, starting with 0x00, cyclically stepping to
2000 0x01 when 0x3F is reached.

2001 **B.1.5 Labelling of SML files**

2002 SML files are labelled per element „UU“, see Tab. 7. This labelling uses the following
2003 features:

- 2004 – The SML file which must be sent first over the transmission line gets the labelling ‚0x00‘;
- 2005 – With each of the following transmitted SML files the feature is increased by one;
- 2006 – When the value ‚0xFF‘ is reached the next one starts with ‚0x00‘ again.

2007 **B.1.6 Feature ‚VV‘**

2008 The element „VV“, see Tab. 7, is reserved for future extensions and must be set always equal
2009 to ‚0x01‘.

2010 **B.1.7 Negotiation of the timeout to be used**

2011 At the beginning of a transmission the sender proposes the timeout to be used. For this an
2012 ESC sequence of the form
2013 1b 1b 1b 1b 03 00 RR RR
2014 is used where in „RR RR“ the proposed timeout in ‘ms’ is given.

2015 The timeout must be noted in the ESC sequence as
2016 1b 1b 1b 1b 03 00 high byte low byte.

2017 The receiver answers to this ESC sequence with the timeout which has been chosen by it
2018 where only the same or a higher value is allowed. The sender must use the timeout defined
2019 by the receiver.

2020 **B.1.8 Negotiation of the maximum allowed block size**

2021 At the beginning of a transmission the sender proposes the maximum block size to be used.
2022 For this an ESC sequence of the form
2023 1b 1b 1b 1b 04 00 SS SS
2024 is used where in „SS SS“ the proposed timeout in ‘bytes’ is given.
2025

2026 The block size must be noted in the ESC sequence as
2027 1b 1b 1b 1b 04 00 high byte low byte.

2028 The receiver answers to this ESC sequence with the block size which has been chosen by it
2029 where only the same or a lower value is allowed. The sender must use the block size defined
2030 by the receiver.

2031 **B.1.9 Process of establishing a transmission**

2032 A version 2 transmission is initiated by the following process:

2033 The sender sends the first data block which has to be exactly of the following form:
2034 1b 1b 1b 1b 02 00 UU VV
2035 1b 1b 1b 1b 03 00 RR RR (with RR RR proposed timeout in ms)
2036 1b 1b 1b 1b 04 00 SS SS (with SS SS proposed block size in Byte)
2037 1b 1b 1b 1b 1a xx yy zz

2038 The receiver acknowledges this first data block with:
2039 1b 1b 1b 1b 02 80 UU VV
2040 1b 1b 1b 1b 03 00 rr rr (with rr rr confirmed timeout in ms)

2041 1b 1b 1b 1b 04 00 ss ss (with ss ss confirmed block size in Byte)
 2042 1b 1b 1b 1b 1a xx yy zz

2043 **B.1.10 Process of the course of a transmission**

2044 A version 2 transmission is initiated as described above and then runs as a „ping-pong“ of
 2045 sent and acknowledging data blocks.

2046 The finite state machine after a single timeout falls back to the beginning of the current
 2047 transmission step if it gets no acknowledge for the block last sent.

2048 In case of error after a double timeout the complete transmission must be repeated.

2049 **B.1.11 Example of the course of a version 2 transmission process**

2050 As an example for the transmission process the following course is given:

2051 – Start of transmission with:

2052 1b 1b 1b 1b 02 00 UU VV

2053 1b 1b 1b 1b 03 00 RR RR (with RR RR proposed timeout in ms)

2054 1b 1b 1b 1b 04 00 SS SS (with SS SS proposed block size in Byte)

2055 1b 1b 1b 1b 1a xx yy zz

2056 – If the start of transmission is not correctly decoded /received, no reaction is shown by the
 2057 receiver. The sender must initiate the transmission once more.

2058 – Transmission confirmed by:

2059 1b 1b 1b 1b 02 80 UU VV

2060 1b 1b 1b 1b 03 00 rr rr (with rr rr confirmed timeout in ms)

2061 1b 1b 1b 1b 04 00 ss ss (with ss ss confirmed block size in Byte)

2062 1b 1b 1b 1b 1a xx yy zz

2063 – First payload is sent with:

2064 1b 1b 1b 1b 02 01 UU VV

2065 payload

2066 1b 1b 1b 1b 1a xx yy zz

2067 – First payload acknowledged with ACK:

2068 1b 1b 1b 1b 02 81 UU VV

2069 1b 1b 1b 1b 1a xx yy zz

2070 – First payload is rejected with NAK:

2071 1b 1b 1b 1b 02 80 UU VV (with „old“ block number in 0x80)

2072 1b 1b 1b 1b 1a xx yy zz

2073 – Second payload is sent with:

2074 1b 1b 1b 1b 02 02 UU VV

2075 payload

2076 1b 1b 1b 1b 1a xx yy zz

2077 – Second payload acknowledged with ACK:

2078 1b 1b 1b 1b 02 82 UU VV

2079 1b 1b 1b 1b 1a xx yy zz

2080 – Second payload is rejected with NAK:

2081 1b 1b 1b 1b 02 81 UU VV (with „old“ block number in 0x81)

2082 1b 1b 1b 1b 1a xx yy zz

2083 – Third and last payload is sent with:

2084 1b 1b 1b 1b 02 43 UU VV

2085 payload

2086 1b 1b 1b 1b 1a xx yy zz

2087 – Third and last payload acknowledged with ACK:

2088 1b 1b 1b 1b 02 C3 UU VV

2089 1b 1b 1b 1b 1a xx yy zz

2090 – Third and last payload rejected with NAK;
2091 1b 1b 1b 1b 02 82 UU VV (with „old“ block number in 0x81)
2092 1b 1b 1b 1b 1a xx yy zz

2093