SML

Smart Message Language

Version 1.03

Historie

Version	Datum	Kommentar	Verantwortlich
0.xx	26.04.07	• Initiale Version, basierend auf Arbeiten aus 2005 / 2006	Wisy
0.xx	27.04.07	• Redaktionelle / inhaltliche Überarbeitung	Landis+Gyr / Wisy
0.xx	08.05.07	• Berechnung CRC-16: Das zu verwendende Verfahren wird konkretisiert	Wisy
0.xx	15.05.07	• Kodierer vereinfacht: Die Länge wird bei allen Datentypen immer in 4 Bit kodiert (alle Versionen zuvor hatten unterschiedliche Längen in Abhängigkeit vom Datentyp vorgesehen) • Berechnung CRC-16: Die Berechnung erfolgt jetzt über alle Bytes im SML-Transportprotokoll	Landis+Gyr / Wisy
1.00	20.05.07	• Mit ZMP 2007 veröffentlichte, finale Version	Wisy
1.01	25.06.07	• Die mit Tab. 2 und Tab. 3 festgelegten Bezeichner wurden aus dem herstellerspezifischen Bereich des OBIS in den applikationsspezifischen Bereich von OBIS-T verschoben • Das Verhalten von ‚SML_SetProcParameter' bei Angabe fehlerhafter Adressen wurde präzisiert (siehe Tab. 2) • Die Begriffe ‚SML_Message', ‚SML_MessageIdentifier' und ‚SML_MessageBody' wurden redaktionell korrekt zugeordnet • Die Erkennung fehlerhafter SML-Dateien per CRC im SML-Transportprotokoll wurde um die Erkennung fehlerhafter SML-Nachrichten per CRC in der Applikationsschicht erweitert, da nur mit dieser Variante Fehler bei der „on the fly" Bearbeitung (Einsatz per Streaming) bereits mit der Auswertung einer SML-Nachricht erkannt und behandelt werden können (siehe Absatz (A) sowie Erläuterungen zu ‚SML_Message') • Zur Verwendung von SML in Broadcast-Anwendungen wurden Ergänzungen in ‚SML_Open' und ‚SML_SetProcParameter' vorgenommen • In der Datenstruktur ‚SML_TupelEntry' wurden die Elemente ‚unit' und ‚scaler' ergänzt • Der Begriff ‚constraints' in ‚SML_GetProcParameter.Req' wurde zu 'attribute' geändert • In 'SML_Message' wurde das Element 'endOfSmlMsg' ergänzt • Die Formulierung zur 'clientId' bei 'SML_Open' wurde präzisiert	Landis+Gyr / Dr. Neuhaus / Wisy
1.02	19.01.08	• Der Einsatz von ‚SML_GetProfList' und ‚SML_GetProfPack' im Zusammenhang mit ereignisorientierten Aufzeichnungen wurde präzisiert. • Das Verhalten von ‚SML_GetProfList' und ‚SML_GetProfPack' für den Fall der fehlenden	Landis+Gyr / Dr. Neuhaus / Wisy

Version	Datum	Kommentar	Verantwortlich
		Information zu den angefragten OBIS-Kennzahlen wurde präzisiert. • SML wurde auf den Anwendungsfall ‚Broadcast' erweitert. • Die Verwendung von SML in Dateien wurde präzisiert. • Die Liste der Fehlernummern (siehe Tab. 2) wurde erweitert.	
1.03	12.02.08 30.10.08	Spezifikation eines weiteren Transport-Layers mit integrierter Flußsteuerung zur Verwendung bei Halb-Duplex-Medien („Block Transport Layer") Ergänzung von ‚SML_GetList' für Werte-Liste. Ergänzung weiterer Fehlernummern. Freigabe als Version 1.03	Wisy tLZ-Projektgruppe

Inhaltsverzeichnis

Bildverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Einheiten:

(1) Hinsichtlich physikalischer Messgrößen und Einheiten gelten die im SI (siehe DIN 1301, Teil 1) getroffenen Vereinbarungen.

Relevante Abkürzungen:

(2) Den nachfolgenden Abkürzungen können arabische Ziffern nachgestellt werden, um mehrfach auftretende Ausprägungen derselben Funktion / desselben Signals unterscheiden zu können.

+A	⇔ Wirkenergie, Kunde bezieht aus Netz,
-A	⇔ Wirkenergie, Kunde liefert an Netz,
ASN.1	⇔ Abstract Syntax Notation One,
BER	⇔ Basic Encoding Rules,
CCITT	⇔ Comité Consultatif International Télégraphique et Téléphonique,
CR	⇔ Carriage Return,
DIN	⇔ Deutsches Institut für Normung e.V.,
DKE	⇔ Deutsche Elektrotechnische Kommision im DIN,
DLMS	⇔ Device Language Message Specification,
(E) DIN	⇔ Entwurf einer Norm des DIN,
EN	⇔ Europäische Norm,
ID	⇔ Identifikationsnummer,
IEC	⇔ International Electrotechnical Commission,
IEEE	⇔ Institute of Electrical and Electronics Engineers,
IP	⇔ Internet Protocol,
ISO	⇔ Internationale Organisation für Normung,
LAN	⇔ Local Area Network,
LSB	⇔ Least Significant Bit, niederwertigstes Bit,
MDE	⇔ Mobile Datenerfassungseinrichtung,
MSB	⇔ Most Significant Bit, höchstwertigstes Bit,
OBIS	⇔ Objekt-Identifikations-System,
OBIS-T	⇔ OBIS Telemetrie,
OSI	⇔ Open Systems Interconnection Reference Model,
R1	⇔ Blindenergie Quadrant I,
R2	⇔ Blindenergie Quadrant II,
R3	⇔ Blindnergie Quadrant III,
R4	⇔ Blindnergie Quadrant IV,
RS232	⇔ Serielle Schnittstelle,
S-I	⇔ Sekunden-Index,
SML	⇔ Smart Message Language,
TAG	⇔ Merkmal / Kennzeichnung / Auszeichnung bei der Kodierung von Datenelementen,

TCP	⇔ Transmission Control Protocol,
TL	⇔ Type-Length,
UDP	⇔ User Datagram Protocol,
WAN	⇔ Wide Area Network,
XML	⇔ Extensible Markup Language,
ZVEI	⇔ Zentralverband Elektrotechnik- und Elektronikindustrie.

Normen

DIN 1301, Teil 1	10.02	Einheiten, Teil 1: Einheitennamen, Einheitenzeichen
E DIN 43863-4	09.06	Zählerdatenkommunikation – IP-Telemetrie
DIN EN 62056-21	01.03	Elektrizitätszähler, Zählerstandsübertragung, Teil 21: Datenübertragung für festen und mobilen Anschluss (3rd edition of IEC 61107, vormals IEC 1107)
DIN EN 62056-61	01.03	Messung der elektrischen Energie – Zählerstandsübertragung, Teil 61: OBIS Objekt Identification System
IEC 62056-62 DIN EN 62056-62	2002 01/03	Messung der elektrischen Energie - Zählerstandsübertragung, Tarif- und Laststeuerung - Teil 62: Interface-Klassen (IEC 62056-62:2002)
DIN EN 62056-46	01/03	Messung der elektrischen Energie - Zählerstandsübertragung, Tarif- und Laststeuerung - Teil 46: Anwendung des HDLC-Protokolls in der Verbindungsschicht (IEC 62056-46:2002)
CCITT-CRC16	--.--	Standard der CCITT zur Prüfsummenberechnung
ISO 8859-15	03.99	Informationstechnik - 8-Bit-Einzelbyte-codierte Schriftzeichensätze - Teil 15: Lateinisches Alphabet Nr. 9

1 Bezug

(3) Nachfolgende Spezifikation legt ein Kommunikationsprotokoll für Anwendungen im Umfeld der Datenbeschaffung und Parametrierung von Geräten fest.

(4) Zielsetzung bei der Ausarbeitung der Spezifikation war der primäre Wunsch, eine möglichst einfache, auch zur Implementation auf leistungsschwachen embedded Systems geeignete Struktur zu finden, die für die Datenbeschaffung über Weitverkehrsstrecken genutzt werden kann.

(5) Vor diesem Hintergrund wurde die „Smart Message Language“, SML, geschaffen.

2 Grundstruktur

(6) Die Grundstruktur gliedert sich in die Elemente:

- Smart Message Language definiert eine Dateistruktur / Dokumentstruktur zur Aufnahme der zwischen den Endpunkten zu übertragenden Nutzlasten.
- SML Binary Encoding definiert eine gepackte binäre Kodierung der SML.
- SML XML Encoding definiert die Kodierung von SML in XML.
- SML-Transport-Protokoll, benötigt für serielle Punkt-zu-Punkt Verbindungen.

(7) SML-Nachrichten, siehe Kapitel 4, können, wie letztlich auch eine E-Mail, über zustandslose, gesicherte Kommunikationswege transportiert werden. Für das avisierte Einsatzszenario kann daher folgendes Modell zur Übersicht herangezogen werden:

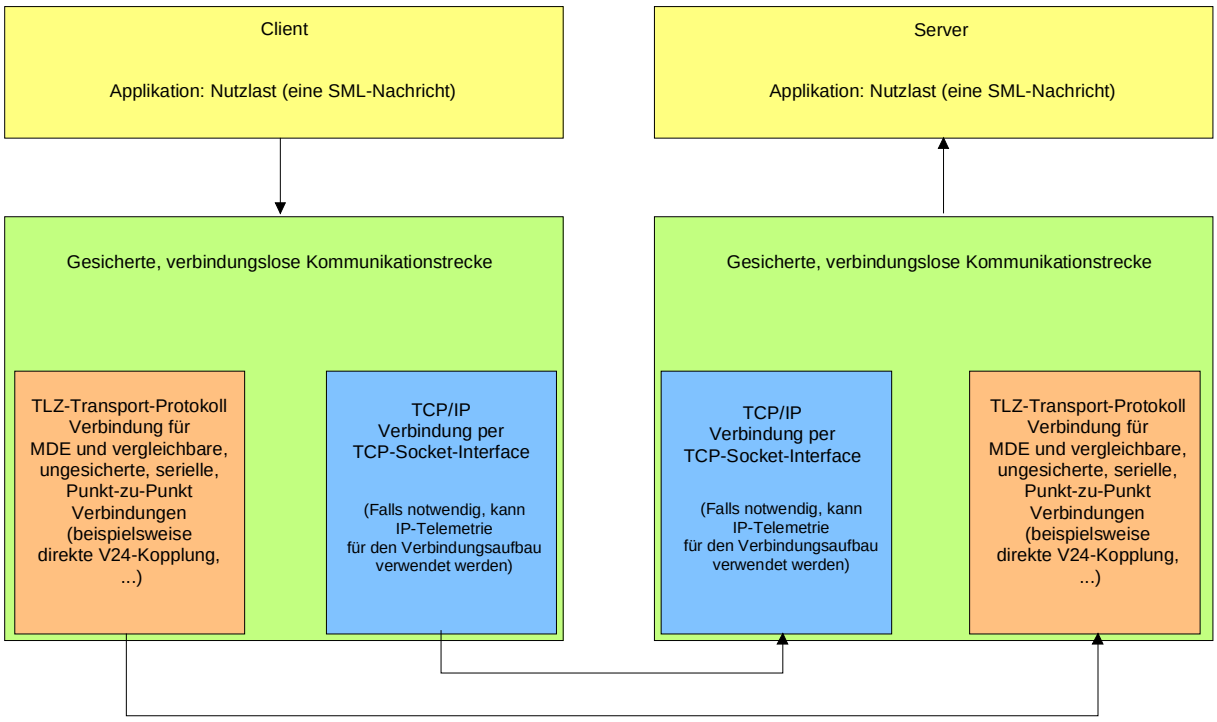

Bild 1: SML-Nachrichten und Kommunikationswege.

3 Begriffe

(8) Nachstehend werden einige / wichtige der in diesem Dokument verwendeten Begriffe erläutert / definiert:

3.1. SML-Datei

(9) Als SML-Datei soll eine Informationseinheit verstanden werden, die, vollkommen losgelöst von der jeweils eingesetzten konkreten Transporttechnik (Internet, Telefon, ...), in sich abgeschlossen ist.

(10) SML-Dateien können in diesem Sinne als abgeschlossene Informationseinheiten aufgefaßt werden, die, genau wie eine E-Mail, in ein Protokoll eingebettet sind und übertragen werden (siehe Bild 2).

(11) Durch den Ansatz der Verwendung von SML-Dateien wird das Konzept unabhängig von der Aufgabe, konkrete Protokolle zum Informationsaustausch definieren zu müssen. Stattdessen wird lediglich verlangt, in einem konkreten Einsatzfall ein bestimmtes Protokoll (beispielsweise HTTP, FTP, ...) auszuwählen und dieses sachdienlich zu parametrieren.

(12) Soweit SML-Dateien als Dateien auf Rechnersystemen verwendet werden, sind diese Dateien ohne Einsatz zusätzlicher Rahmen und unter Verwendung der mit Kapitel 6 definierten Kodierung zu notieren, es sei denn, die konkrete Applikation trifft explizit eine anders lautende Vorgabe.

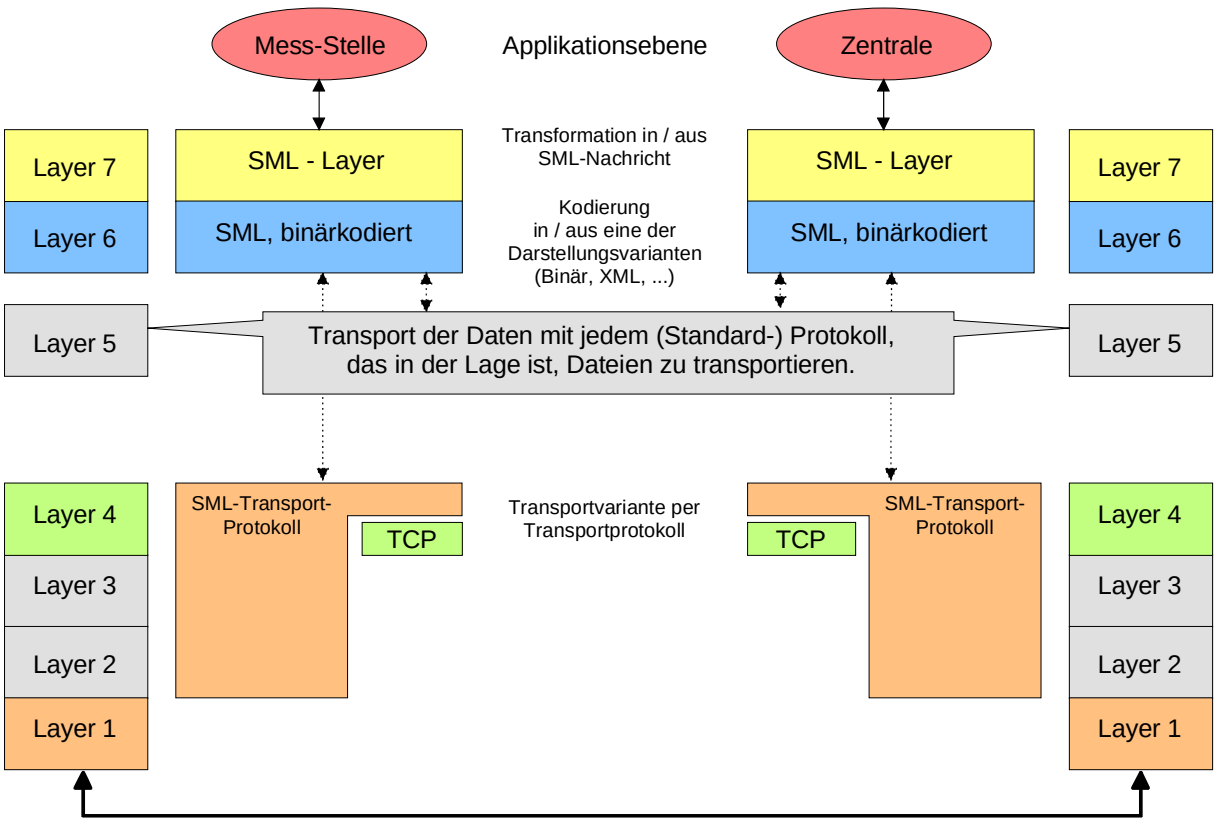

Bild 2: SML-Kommunikationsmodell.

4 SML, Smart Message Language

4.1. Grundaufbau

(13) Eine SML-Datei ist immer als Kette von SML-Nachrichten aufgebaut.

(14) SML-Dateien können zur Reduktion der Dateigröße segmentiert werden.

(15) SML-Dateien können in den Varianten ...
... SML-Auftragsdatei,
... SML-Antwortdatei oder
... SML-Kombidatei
auftreten.

(16) SML-Auftragsdateien enthalten die Aufträge („Requests“). SML-Antwortdateien fassen die Antworten („Responses“) zu den Aufträgen zusammen.

(17) Jede SML-Auftragsdatei beginnt mit genau einer und enthält genau eine SML_...Open.Req-Nachricht. Sie endet mit genau einer und enthält genau eine SML_...Close.Req-Nachricht. Diese Festlegung gilt ebenfalls für SML-Antwortdateien, denen keine SML-Auftragsdatei zugeordnet werden kann, wobei an Stelle der Variante Request die Variante Response zu verwenden ist.

(18) SML-Antwortdateien, denen SML-Auftragsdateien zugeordnet werden können, beginnen mit genau einer SML_...Open.Res- oder einer SML_Attention.Res-Nachricht. Sie enden mit genau einer SML_...Close.Res- oder einer SML_Attention.Res-Nachricht.

(19) Zum Einsatz von SML über Transportmedien mit geringer Performance[1], können SML-Nachrichten ohne den Rahmen einer SML-Datei und ausschließlich als „Response without Request“ versendet werden. Dieser Anwendungsfall ist explizit von der Applikation zu definieren.

(20) Eine SML-Kombidatei enthält die SML-Nachrichten einer SML-Auftragsdatei zuzüglich der SML-Nachrichten der zugehörigen SML-Antwortdatei(en).

5 SML-Nachrichten

(21) Eine SML-Nachricht ist entweder eine ‚Request-Nachricht‘ oder eine ‚Response-Nachricht‘.

(22) Eine SML-Nachricht umfaßt Aufgabe und zugeordnete Attribute.

[1] Im Sinne dieser Spezifikation gelten als „*Transportmedien mit geringer Performance*“ ausschließlich folgende Varianten: Nahfunk- oder PLC-Strecken zwischen Sensoren und Konzentratoren.

(A)
```
SML_Message                 ::=   SEQUENCE
    {
      transactionId                 Octet String,
      groupNo                       Unsigned8,
      abortOnError                  Unsigned8,
      messageBody                   SML_MessageBody,
      crc16                         Unsigned16,
      endOfSmlMsg                   EndOfSmlMsg
    }
```

(23) Die ‚transactionId' wird bei der Erzeugung von ‚Request-Nachrichten' in ein-eindeutiger[2] Form durch den Auftraggeber gebildet. Jede ‚Response-Nachricht' spiegelt die zu deren ‚Request' gehörende Transaktionsnummer unverändert zurück, so dass eine ‚Response' stets dem zugehörigen ‚Request' zugeordnet werden kann.

(24) Wird im Sinne des ‚Push-Betriebs' eine SML-Datei erzeugt, zu deren SML-Response-Nachrichten prinzipbedingt keine SML-Request-Nachrichten existieren, erzeugt der Ersteller dieser SML-Response-Nachricht selbsttätig ein-eindeutige Transaktionsnummern.

(25) Das Attribut ‚groupNo' erlaubt die Bildung von SML-Nachrichten-Gruppen. Dieser Mechanismus soll dazu verwendet werden, anzugeben, welche SML-Nachrichten in einer bestimmten Reihenfolge abzuarbeiten sind und welche nebenläufig (und zwar sowohl im Sinne von Simultanarbeit der ersten wie auch der zweiten Art) ausgeführt werden dürfen.

(26) Generell erfolgt die Abarbeitung der Gruppen sequentiell und in der Reihenfolge, in der sie innerhalb einer SML-Datei auftreten.

(27) Das Vermischen von SML-Nachrichten verschiedener Gruppen ist unzulässig.

(28) Nachrichten innerhalb einer Gruppe können vom Empfänger wahlweise seriell oder nebenläufig abgearbeitet werden.

(29) Die Prüfsumme (Element ‚crc16') ist als CRC16 nach DIN EN 62056-46 zu berechnen. Die Berechnung beginnt mit dem ersten Byte zu ‚SML_Message' und endet mit dem letzten Byte zu ‚messageBody'. Damit sind die Bytes der Elemente ‚crc16' und ‚null' von der Prüfsummenberechnung ausgeschlossen.

(30) Wird eine SML-Nachricht empfangen, die dekodiert werden konnte und deren Prüfsumme fehlerhaft ist, so ist diese SML-Nachricht mit einem ‚SML_Attention' (81 81 C7 C7 FE 0B, siehe Tab. 2) und dem Feld ‚transactionId'gemäß Absatz (24) zu beantworten. Wird in diesem Fall die Nachricht ‚SML_Close' empfangen, so ist der Fehler zu ignorieren und in der Antwort ein korrektes ‚SML_Close' zu liefern, dessen Feld ‚transactionId'gemäß Absatz (24) gesetzt wird.

(31) Wird die Nachricht ‚SML_Open' mit fehlerhafter Prüfsumme empfangen, so ist die ganze SML-Datei zu ignorieren[3].

[2] Die Eindeutigkeit ist jeweils innerhalb des Umfelds des Erzeugers der Transaktionsnummer zu gewährleisten; nicht gefordert wird eine weltweite Eindeutigkeit (vergleichbar der MAC-Adresse bei Ethernet). Durch die Angaben in der SML-Open-Nachricht kann die Eindeutigkeit auf den jeweiligen Erzeuger zugeordnet werden.

[3] Ignorieren: Der empfangene Inhalt wird verworfen. Es wird keine Antwort generiert.

(32) Wird eine SML-Datei empfangen, die dekodiert werden konnte und die nicht mit ‚SML_Open' beginnt, so ist mindestens[4] für die erste SML-Nachricht dieser SML-Datei ein SML-Attention mit Fehlerkode „unerwartete SML-Nachricht" zu senden.

(33) Wird eine SML-Nachricht empfangen, die nicht dekodiert werden kann, so ist wie folgt zu verfahren:

- Handelt es sich bei der SML-Nachricht um die erste Nachricht, so wird die ganze SML-Datei verworfen (gilt sowohl für SML-Auftragsdateien als auch für SML-Antwortdateien).
- Handelt es sich um eine der nachfolgenden SML-Nachrichten, so wird als Antwort für die betroffene SML-Nachricht ein SML-Attention (81 81 C7 C7 FE 01, siehe Tab. 2) gefolgt von einem SML-Close gesendet (gilt nur für SML-Auftragsdateien). Alle weiteren SML-Nachrichten der betroffenen SML-Datei werden ignoriert (sowohl bei SML-Auftragsdateien als auch bei SML-Antwortdateien).
- Die ‚transactionId' ist für die SML-Attention sowie für das sich anschließende SML-Close gemäß Absatz (24) zu erzeugen.

(34) SML-Nachrichten werden grundsätzlich in SML-Dateien zusammengefasst.

[4] Alterantiv ist es zulässig, für jede SML-Nachricht der SML-Datei ein SML-Attention zu generieren.

Reihenfolge der SML-Nachrichten	Trans-aktions-nummer	Gruppen-Nr.	SML-Nachricht	Kommentar
0	0	1	OPEN	Wird zuerst ausgeführt.
1	1	4	GET_ProfilPack	Erste Gruppe, wird als zweiter Block ausgeführt. Beispielsweise drei Lastgänge ablesen.
2	2	4	GET_ProfilPack	
3	3	4	GET_ProfilPack	
4	4	7	GET_ProfilPack	Zweite Gruppe, wird als dritte Aktion ausgeführt, Beispielsweise einen Lastgang, die Verrechnungsliste und das Logbuch ablesen.
5	5	7	GET_ProfilList	
6	6	7	GET_ProfilList	
7	7	8	SET_ProcParameter	Wird als vierte Aktion ausgeführt. Beispielsweise den Bezugszeitpunkt nachführen
8	8	8	CLOSE.Req	Wird zuletzt bearbeitet.

Tab. 1: Beispiel zur Verwendung des Merkmals ‚Gruppen-Nummer'.

(35) Damit ist die Transaktionsnummer lediglich das Merkmal, um die Responses den zugehörigen Requests zuzuordnen. Sie hat keinerlei Einfluss auf die Reihenfolge der Ausführung der SML-Nachrichten beim Empfänger.

(36) Das Attribut ‚abortOnError' legt fest, wie im Falle von Fehlern bei der Ausführung der SML-Nachricht verfahren werden soll, wobei das abschließende Close immer ausgeführt werden muss und nicht ausgeführte Requests immer eine Response mit der Fehlermeldung ‚nicht ausgeführt' als Rückmeldung erzeugen:

(B)

abortOnError	⇔ 0x00	Ausführung fortsetzen,
abortOnError	⇔ 0x01	Ausführung ab der nächsten Gruppe fortsetzen,
abortOnError	⇔ 0x02	Ausführung der aktuellen Gruppe fortsetzen, danach keine weitere Gruppe mehr ausführen. Falls innerhalb der selben Gruppe ein ‚0x02' zu finden ist, auf das ein ‚0x01' folgt, ist die Ausführung sofort abzubrechen.
abortOnError	⇔ 0xFF	Ausführung sofort abbrechen.

(C)

```
SML_MessageBody              ::= CHOICE
  {
    OpenRequest              [0x00000100]   SML_PublicOpen.Req
    OpenResponse             [0x00000101]   SML_PublicOpen.Res

    CloseRequest             [0x00000200]   SML_PublicClose.Req
    CloseResponse            [0x00000201]   SML_PublicClose.Res

    GetProfilePackRequest    [0x00000300]   SML_GetProfilePack.Req
    GetProfilePackResponse   [0x00000301]   SML_GetProfilePack.Res

    GetProfileListRequest    [0x00000400]   SML_GetProfileList.Req
    GetProfileListResponse   [0x00000401]   SML_GetProfileList.Res

    GetProcParameterRequest  [0x00000500]   SML_GetProcParameter.Req
    GetProcParameterResponse [0x00000501]   SML_GetProcParameter.Res

    SetProcParameterRequest  [0x00000600]   SML_SetProcParameter.Req
    SetProcParameterResponse [0x00000601]   SML_SetProcParameter.Res

    GetListRequest           [0x00000700]   SML_GetList.Req
    GetListResponse          [0x00000701]   SML_GetList.Res

    AttentionResponse        [0x0000FF01]   SML_Attention.Res
  }
```

5.1. SML-Bezeichner

5.1.1. SML_PublicOpen.Req

(37) Der SML_PublicOpen.Req muss immer zu Beginn einer SML-Auftragsdatei vorhanden sein. Er dient der Identifikation des Auftraggebers und der Authentifizierung per Benutzer / Passwort sowie der Zuordnung von SML-Antwortdatei(en) an die SML-Auftragsdatei.

(38) Jeder Auftragnehmer muss auf den SML_PublicOpen.Req entweder mit einer SML_PublicOpen.Res oder einer SML_Attention.Res antworten.

(D)

```
SML_PublicOpen.Req   ::=  SEQUENCE
    {
      codepage              Octet String OPTIONAL,
      clientId              Octet String
      reqFileId             Octet String
      serverId              Octet String OPTIONAL,
      username              Octet String OPTIONAL,
      password              Octet String OPTIONAL,
      sml-Version           Unsigned8 OPTIONAL,
    }
```

(39) Per ‚codepage' wird, ist der Wert angegeben, eine andere als die Default-Codepage vereinbart. Die ‚Codepage' legt fest, welcher Zeichensatz zur Interpretation von Zeichenketten zu verwenden ist. Fehlt der Wert, wird ‚ISO 8859-15' verwendet.

Der Inhalt zur ‚Codepage' selber ist immer als Zeichenkette in ISO 8859-15 zu notieren.

(40) Der Parameter ‚serverId' erlaubt es, ist er angegeben, per ‚PublicOpen.Req' im Sinne einer Adresse gezielt eine SML-Datenquelle (ein konkretes Messgerät) oder ein Softwaremodul anzusprechen. In diesem Fall müssen alle nachfolgenden SML-Nachrichten ebenfalls den Parameter ‚serverId' mit Inhalt angeben. Fehlt der Parameter ‚serverId' in einer der nachfolgenden Nachrichten, so ist für die entsprechende SML-Nachricht mit einem ‚SML_Attention' (81 81 C7 C7 FE 0C, siehe Tab. 2) zu antworten.

(41) Fehlt der Parameter ‚serverId', wird die Anfrage als ‚Broadcast' gewertet. In diesem Fall muß bei allen nachfolgenden SML-Nachrichten der Parameter ‚serverId' ebenfalls fehlen. SML-Nachrichten, die in dieser Situation dennoch den Parameter ‚serverId' angeben, sind zu ignorieren.

(42) Der Parameter ‚clientId' wird bei der Erzeugung einer SML-Auftragsdatei vom Auftraggeber erzeugt und dient der eindeutigen Adressierung der SML-Antwortdatei des Clients.

(43) Der Parameter ‚reqFileId' bezeichnet in systemweit[5] eindeutiger Form ein konkretes SML-Auftragsdatei- / SML-Antwortdatei-Tupel. Er erlaubt es, SML-Antworten zu deren Aufträgen zuzuordnen.

(44) Die ‚reqFileId' liefert die eindeutige Kennzeichnung der SML-Datei, beispielsweise gebildet aus dem aktuellen Zeitstempel.

(45) Fehlt der Parameter ‚sml-Version', wird die Version 1 als Standard angenommen.

5.1.2. SML_PublicOpen.Res

(46) Der SML_PublicOpen.Res steht immer zu Beginn einer SML-Antwortdatei. Er dient der Identifikation der SML-Antwortdatei zu der zugehörigen SML-Auftragsdatei.

(E)
```
SML_PublicOpen.Res      ::=  SEQUENCE
    {
      codepage                Octet String OPTIONAL,
      clientId                Octet String OPTIONAL,
      reqFileId               Octet String,
      serverId                Octet String,
      refTime                 SML_Time OPTIONAL,
      sml-Version             Unsigned8 OPTIONAL,
    }
```

[5] Die Eindeutigkeit ist jeweils innerhalb des Umfelds des Erzeugers der ‚reqFileId' zu gewährleisten; nicht gefordert wird eine weltweite Eindeutigkeit (vergleichbar der MAC-Adresse bei Ethernet). Durch die anderen Angaben in der SML-Open-Nachricht kann die Eindeutigkeit auf den jeweiligen Erzeuger zugeordnet werden.

(47) Das Element ‚SML_Time' wird entweder als Sekunden-Index oder als Zeitstempel angegeben.

(F)
```
SML_Time                    ::=  CHOICE
    {
      secIndex                   [0x01]     Unsigned32,
      timestamp                  [0x02]     SML_Timestamp
    }
```

(48) Handelt es sich um einen Zeitstempel, wird dieser immer in Sekunden ausgehend vom 01.01.1970, 00:00:00 (UNIX-Bezugszeitpunkt, bezogen auf UTC), gebildet.

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

(49) Falls die Antwort von einem Gerät oder einem Softwaremodul geliefert wird, das selbst nicht über eine Zeitinformation verfügt, fehlt diese Angabe im ‚Public-Open.Res'.

(50) Fehlt der Parameter ‚sml-Version', wird die Version 1 als Standard angenommen.

(51) Ist die SML-Datei eine SML-Response-Datei, zu der eine SML-Request-Datei gehört, wird das Element ‚reqFileId' der SML-Request-Datei zurückgegeben.

(52) Ist die SML-Datei eine SML-Response-Datei, zu der keine SML-Request-Datei gehört, wird das Element ‚reqFileId', vergleichbar der Erzeugung dieses Attributs beim Erstellen einer SML-Request-Datei, auf einen systemweit eindeutigen Namen gesetzt.

(53) Ist die SML-Datei eine SML-Response-Datei, zu der keine SML-Request-Datei gehört, darf das Element ‚clientId' weggelassen werden; in allen anderen Fällen ist der mit dem ‚PublicOpen.Req' angelieferte Wert dort einzutragen.

(54) Das Feld ‚refTime' liefert den Referenzzeitpunkt zur Erstellung der SML-Antwortdatei.

5.1.3. SML_PublicClose.Req

(55) Der SML_PublicClose.Req muss immer am Ende einer SML-Auftragsdatei vorhanden sein. Er beendet diese Datei.

(56) Jeder Auftragnehmer muss auf den SML_PublicClose.Req mit einer SML_PublicClose.Res oder einer SML_Attention.Res antworten.

(H)
```
SML_PublicClose.Req         ::=  SEQUENCE
    {
      globalSignature            SML_Signature OPTIONAL
    }
```

5.1.4. SML_PublicClose.Res

(57) Der SML_PublicClose.Res steht immer am Ende einer SML-Antwortdatei. Er beendet diese Datei.

(I) SML_PublicClose.Res ::= SEQUENCE
{
globalSignature SML_Signature OPTIONAL
}

(J) SML_Signature ::= Octet String

5.1.5. SML_GetProfilePack.Req

(58) In einer SML-Auftragsdatei kann eine oder können mehrere SML_GetProfilePack-Nachrichten vorhanden sein.

(59) Jede SML_GetProfilePack.Req dient der Anfrage von einzelnen Messwerten oder Messwerte-Listen, die in gepackter Form übertragen werden.

(60) Der Auftragnehmer muss auf jeden SML_GetProfilePack.Req mit genau einer SML_GetProfilePack.Res oder einer SML_Attention.Res antworten.

(K) SML_GetProfilePack.Req ::= SEQUENCE
{
serverId Octet String OPTIONAL,
username Octet String OPTIONAL,
password Octet String OPTIONAL,
withRawdata Boolean OPTIONAL,
beginTime SML_Time OPTIONAL,
endTime SML_Time OPTIONAL,
parameterTreePath SML_TreePath,
object_List List_of_SML_ObjReqEntry OPTIONAL,
dasDetails SML_Tree OPTIONAL
}

(L) List_of_SML_ObjReqEntry ::= SEQUENCE OF
{
object_List_Entry SML_ObjReqEntry
}

(M) SML_ObjReqEntry ::= Octet String

(61) Das Element ‚parameterTreePath' bezeichnet die Messwerte-Liste und dient damit der Einordnung in Varianten wie Logbuch, Lastgang oder vergleichbar. Es muß mindestens mit einer Kennzahl besetzt sein {zur Kodierung siehe Absätze (BB) und (92)}.

(62) Die ‚object_List' wird benötigt, um die gewünschten Kanäle zur Ablesung von Lastgängen mit dem Auftrag angeben zu können. Ist dieses Element nicht angegeben oder leer, so sind alle in dem Profil vorhandenen Kanäle in der Antwort aufzuführen.

(63) Der gewünschte Zielbereich wird über Zeitgrenzen angegeben.

(64) Es gilt folgende Vereinbarung, falls Zeitgrenzen per Sekunden-Index definiert werden[6]:

(65) Der Sekunden-Index mit dem Wert ‚00000000' beschreibt stets den zeitlich am weitesten in der Vergangenheit liegenden Wert.

(66) Der Sekunden-Index mit dem Wert ‚FFFFFFFF' beschreibt stets den ‚aktuellen Eintrag', also den zeitlich direkt vor dem aktuellen Zeitpunkt liegenden Wert.

(67) Falls bei Bereichsanfragen der gewünschte Sekunden-Index nicht verfügbar ist, ist in der Antwort ...

... falls für den Beginn einer Bereichsgrenze vorhanden,
immer der direkt zeitlich danach[7] liegende Wert oder
... falls dieser nicht vorhanden ist,
der ‚aktuelle Eintrag'

... falls für das Ende einer Bereichsgrenze vorhanden,
immer der direkt zeitlich davor[8] liegende Wert oder
... falls dieser nicht vorhanden ist,
der ‚aktuelle Eintrag'

zu verwenden.

(68) Falls die ‚beginTime' nicht angegeben ist, ist der Wert ‚00000000' anzunehmen.

(69) Falls die ‚endTime' nicht angegeben ist, ist der Wert ‚FFFFFFFF' anzunehmen.

(70) Per ‚dasDetails' können bedarfsweise ergänzend für die Datenbeschaffung benötigte Parameter übertragen werden. Diese werden, siehe Kapitel 5.1.10, als Baumstruktur übertragen.

5.1.6. SML_GetProfilePack.Res

(71) In einer SML-Antwortdatei kann eine oder können mehrere SML_GetProfilePack.Res-Nachrichten vorhanden sein.

(72) Jede SML-Datenquelle liefert genau eine SML_GetProfilePack.Res-Antwort auf einen SML_GetProfilePack.Req-Auftrag.

[6] Das aktuell vorliegende Einsatzumfeld sieht ausschließlich die Verwendung des Sekunden-Index vor, siehe Erläuterung zu ‚SML_Time'. Mögliche, künftige Erweiterungen können hierzu weitere Alternativen bezeichnen.

[7] In Richtung größerer Indices.

[8] In Richtung kleiner Indices.

```
(N)  SML_GetProfilePack.Res                ::=  SEQUENCE
       {
         serverId                               Octet String,
         actTime                                SML_Time,
         regPeriod                              Unsigned32,
         parameterTreePath                      SML_TreePath,
         header_List                            List_of_SML_ProfObjHeaderEntry,
         period_List                            List_of_SML_ProfObjPeriodEntry,
         rawdata                                Octet String OPTIONAL,
         profileSignature                       SML_Signature OPTIONAL
       }

(O)  List_of_SML_ProfObjHeaderEntry        ::=  SEQUENCE OF
       {
         header_List_Entry                      SML_ProfObjHeaderEntry
       }

(P)  SML_ProfObjHeaderEntry                ::=  SEQUENCE
       {
         objName                                Octet String,
         unit                                   SML_Unit,
         scaler                                 Integer8
       }

(Q)  List_of_SML_ProfObjPeriodEntry        ::=  SEQUENCE OF
       {
         period_List_Entry                      SML_ProfObjPeriodEntry
       }

(R)  SML_ProfObjPeriodEntry                ::=  SEQUENCE
       {
         valTime                                SML_Time,
         status                                 Unsigned64,
         value_List                             List_of_SML_ValueEntry,
         periodSignature                        SML_Signature OPTIONAL
       }

(S)  List_of_SML_ValueEntry                ::=  SEQUENCE OF
       {
         value_List_Entry                       SML_ValueEntry
       }

(T)  SML_ValueEntry                        ::= SEQUENCE
       {
         value                                  SML_Value,
         valueSignature                         SML_Signature OPTIONAL
       }
```

(U) SML_Value ::= IMPLICIT CHOICE

```
{
  boolean-Value          boolean,
  byte-List              Octet String[9],
  8-Bit-Integer          Integer8,
  16-Bit-Integer         Integer16,
  32-Bit-Integer         Integer32,
  64-Bit-Integer         Integer64,
  8-Bit-Unsigned         Unsigned8,
  16-Bit-Unsigned        Unsigned16,
  32-Bit-Unsigned        Unsigned32,
  64-Bit-Unsigned        Unsigned64
}
```

(V) SML_Unit ::= Unsigned8

Zahlenwerte siehe DLMS-Unit-List, zu finden beispielsweise in IEC 62056-62.

(73) Die Anzahl der Elemente in der ‚Überschriften-Liste' (‚header_List') muss immer identisch mit der Anzahl der Elemente in der Liste der Registrierperioden (‚value_-List') sein.

(74) Die Anordnung der Registrierperioden in der Liste erfolgt nach dem Muster „der Wert mit dem kleinsten Sekunden-Index zuerst".

(75) Das Element ‚actTime' liefert die Zeitinformation, die zum Zeitpunkt des Beginns der Ausführung des Auftrags bei der Datenquelle vorgelegen hat.

(76) Das Element ‚valTime' liefert die Zeitinformation, die zum Zeitpunkt der Messwertbildung bei der Datenquelle vorgelegen hat.

(77) Das Element ‚regPeriod' legt die Dauer der verwendeten Registrierperiode fest. Der Wert wird in Sekunden angegeben. Handelt es sich bei der Aufzeichnung um ein ereignisorientiertes Profil (liegt also keine konkrete Registrierperiode vor), ist als Wert ‚0' zu verwenden.

(78) Die Verwendung der beiden voneinander getrennten Listen (‚header_List' und ‚period_List') wird gewählt, um auf die mehrfache Erfassung redundanter Informationen, wie beispielsweise Sekunden-Index und Status zum Zeitpunkt der Messwertbildung je Registrierperiode bei mehreren Lastgängen („+A", „-A", ...), verzichten zu können.

(79) Durch die wahlweise Verwendung von ‚profileSignature', ‚periodSignature' oder ‚valueSignature' können sowohl ganze Lastgänge, EinzelMesswerte wie auch Tupel ganzer Registrierperioden gemeinsam geschützt werden. Welcher Ansatz verwendet wird, hängt dabei von der Applikation ab.

(80) Mit dem Feld ‚scaler' wird der Bezug zwischen der Einheit und dem Zahlenwert wie folgt hergestellt:

$$\text{Zahlenwert} = \text{SML\_Value} \times 10^{\text{scaler}}$$

[9] Hinweis: Die Verwendung eines Octet String mit der länge ‚0' ist zulässig.

5.1.7. SML_GetProfileList.Req

(81) In einer SML-Auftragsdatei kann eine oder können mehrere SML_GetProfileList-Nachrichten vorhanden sein.

(82) Jede SML_GetProfileList.Req dient der Anfrage von einzelnen Messwerten oder Messwerte-Listen, die in simpler Listenform übertragen werden. Im Gegensatz zu SML_GetProfilePack, bei der die Antworten möglichst ohne Redundanz und optimal gepackt übertragen werden, erwartet SML_GetProfile simple Listen. Diese bieten den Nachteil, bei der Übertragung von Tageslastgängen erheblich mehr Datenvolumen zu erzeugen, liefern im Gegenzug aber den Vorteil einer einfachen Struktur, die bei Systemen mit viertelstündlicher Datenbeschaffung erheblich effizienter in der Anwendung wird.

(83) Der Auftragnehmer kann auf jeden SML_GetProfileList.Req mit einer oder mehreren SML_GetProfileList.Res oder einer SML_Attention.Res antworten.

(W)
```
SML_GetProfileList.Req      ::=  SEQUENCE
    {
      serverId                   Octet String OPTIONAL,
      username                   Octet String OPTIONAL,
      password                   Octet String OPTIONAL,
      withRawdata                Boolean OPTIONAL,
      beginTime                  SML_Time OPTIONAL,
      endTime                    SML_Time OPTIONAL,
      parameterTreePath          SML_TreePath,
      object_List                List_of_SML_ObjReqEntry OPTIONAL,
      dasDetails                 SML_Tree OPTIONAL
    }
```

(84) Für die Elemente zu SML_GetProfileList.Req gelten die unter SML_GetProfilePack.Req definierten Vorgaben.

5.1.8. SML_GetProfileList.Res

(85) In einer SML-Antwortdatei kann eine oder können mehrere SML_GetProfileList.Res-Nachrichten vorhanden sein.

(86) Jede SML-Datenquelle liefert eine oder mehrere SML_GetProfileList.Res-Antwort auf einen SML_GetProfileList.Req-Auftrag. Jede SML_GetProfileList.Res-Antwort enthält die Messwerte einer Registrierperiode (damit üblicherweise einer Viertelstunde) im Sinne eines Atoms.

(X)
```
SML_GetProfileList.Res          ::=   SEQUENCE
      {
        serverId                      Octet String,
        actTime                       SML_Time,
        regPeriod                     Unsigned32,
        parameterTreePath             SML_TreePath,
        valTime                       SML_Time,
        status                        Unsigned64,
        period-List                   List_of_SML_PeriodEntry,
        rawdata                       Octet String OPTIONAL,
        periodSignature               SML_Signature OPTIONAL
      }
```

(Y)
```
List_of_SML_PeriodEntry         ::=   SEQUENCE OF
      {
        period_List_Entry             SML_PeriodEntry
      }
```

(Z)
```
SML_PeriodEntry                 ::= SEQUENCE
      {
        objName                       Octet String,
        unit                          SML_Unit,
        scaler                        Integer8,
        value                         SML_Value,
        valueSignature                SML_Signature OPTIONAL
      }
```

(87) Durch die wahlweise Verwendung von ‚periodSignature' oder ‚valueSignature' können sowohl EinzelMesswerte als auch Tupel ganzer Registrierperioden gemeinsam geschützt werden. Welcher Ansatz verwendet wird, hängt dabei von der Applikation ab.

5.1.9. SML_GetProcParameter.Req

(88) Per SML_GetProcParameter.Req können in einer SML-Auftragsdatei Betriebsparameter (Modem-Parameter, Protokoll-Parameter, Auslastung von Software-Modulen, ...) abgefragt werden.

(89) Der Auftragnehmer antwortet auf diese Nachricht per SML_GetProcParameter.Res oder SML_Attention.Res in der Antwortdatei.

(90) Ein einzelner Parameter ist immer durch dessen OBIS-Kennzahl definiert. Aus der OBIS-Kennzahl leitet sich implizit der konkrete Datentyp und der Wertebereich für das zu liefernde Ergebnis ab. Die konkrete Zuordnung sowie zulässige OBIS-Kennzahlen und deren Bedeutung sind dem Anhang zu entnehmen. Innerhalb der Kodierung einer SML-Nachricht werden die Datentypen außerdem mit kodiert, so dass die generische Implementation zur Konvertierung der Parameter in interne Programmelemente einfach möglich ist.

(91) Da die Struktur und Anzahl der für einen konkreten Anwendungsfall benötigten Parameter im Vorfeld unbekannt ist, das ständige Anpassen der SML-Spezifikation

an neue Anforderungen aber vermieden werden sollte, verwenden die ‚SML_..._ProcParameter'-Nachrichten eine Baumstruktur, so dass Parameter-Bäume beliebiger Anordnung transportiert werden können.

(AA) SML_GetProcParameter.Req ::= SEQUENCE
{
  serverId Octet String OPTIONAL,
  username Octet String OPTIONAL,
  password Octet String OPTIONAL,
  parameterTreePath SML_TreePath
  attribute Octet String OPTIONAL,
}

(BB) SML_TreePath ::= SEQUENCE OF
{
  path_Entry Octet String
}

(92) Per SML_TreePath können gezielt Parameter innerhalb eines Parameterbaums adressiert werden. Dabei beginnt ein SML_TreePath grundsätzlich ab dem Root-Element des Parameterbaums. Die Bedeutung (entsprechend der Adresse) des Root-Elements ist mit dessen Parameter-Namen gegeben.

(93) Durch die Listen-Struktur wird der Parameter-Name des Root-Elements grundsätzlich in jenem SML_TreePath-Element zu finden sein, das als erstes und damit direkt unter ‚SML_GetProcParameter.Req' zu finden ist.

(94) Über die Eigenschaft ‚attribute' kann im Bedarfsfall eine Einschränkung / Detailinformation zur Präzisierung des gewünschten Anfrageergebnisses mitgegeben werden.

5.1.10. SML_GetProcParameter.Res

(95) Per SML_GetProcParameter.Res wird in einer SML-Antwortdatei der Auftrag SML_GetProcParameter.Req quittiert.

(CC) SML_GetProcParameter.Res ::= SEQUENCE
{
  serverId Octet String,
  parameterTreePath SML_TreePath,
  parameterTree SML_Tree
}

(DD) SML_Tree ::= SEQUENCE
{
  parameterName Octet String,
  parameterValue SML_ProcParValue OPTIONAL,
  child_List List_of_SML_Tree OPTIONAL
}

(EE)
```
SML_ProcParValue                ::= CHOICE
    {
      smlValue                      [0x01]      SML_Value,
      smlPeriodEntry                [0x02]      SML_PeriodEntry,
      smlTupelEntry                 [0x03]      SML_TupelEntry,
      smlTime                       [0x04]      SML_Time
    }
```

(FF)
```
SML_TupelEntry                  ::= SEQUENCE
    {
      serverId                      Octet String,
      secIndex                      SML_Time,
      status                        Unsigned64,
      unit_pA                       SML_Unit,
      scaler_pA                     Integer8,
      value_pA                      Integer64,
      unit_R1                       SML_Unit,
      scaler_R1                     Integer8,
      value_R1                      Integer64,
      unit_R4                       SML_Unit,
      scaler_R4                     Integer8,
      value_R4                      Integer64,
      signature_pA_R1_R4            Octet String,
      unit_mA                       SML_Unit,
      scaler_mA                     Integer8,
      value_mA                      Integer64,
      unit_R2                       SML_Unit,
      scaler_R2                     Integer8,
      value_R2                      Integer64,
      unit_R3                       SML_Unit,
      scaler_R3                     Integer8,
      value_R3                      Integer64,
      signature_mA_R2_R3            Octet String
    }
```

(GG)
```
List_of_SML_Tree                ::=   SEQUENCE OF
    {
      tree_Entry                        SML_Tree
    }
```

(96) Per ‚SML_Tree' können einzelne Parameter (Blätter oder Knoten) mit deren (bei Knoten) darunter folgenden Kindern (siehe ‚child_List') aufgebaut werden. Konkret kann durch einen ‚SML_Tree' damit ...
... ein einzelner Parameter,
... ein Knoten mit einer darunter hängenden Liste von weiteren Parametern oder
... ein Knoten mit einer darunter hängenden Liste von weiteren Teilbäumen abgebildet werden.

(97) Die Elemente ‚parameterValue' und ‚child_List' sind beide optional, so dass folgende Varianten abgebildet werden können:

- Ein Parameter besteht aus Namen (angegeben per OBIS) und Wert.
- Ein Parameter besteht aus Namen und Teilbaum oder Liste weiterer Parameter.
- Ein Parameter besteht aus Namen, Wert und Teilbaum oder Liste weiterer Parameter.
- Namen werden per OBIS-Kennzahl kodiert.

(98) Ob ein ‚SML_Tree' als Blatt oder Knoten zu werten ist, wird durch das Vorhandensein des Elements ‚child_List' festgelegt. Hat ein ‚SML_Tree' das Element ‚child_List', ist es ein Knoten, fehlt dieses Element, ist es ein Blatt.

5.1.11. SML_SetProcParameter.Req

(99) Per SML_SetProcParameter.Req können in einer SML-Auftragsdatei Betriebsparameter (beispielsweise die Baudrate zum Zugriff auf ein Endgerät) übertragen werden.

(100) Zur Strukturierung sei auf die Hinweise in Kapitel 5.1.9 verwiesen.

(HH)
```
SML_SetProcParameter.Req        ::=   SEQUENCE
    {
      serverId                        Octet String OPTIONAL,
      username                        Octet String OPTIONAL,
      password                        Octet String OPTIONAL,
      parameterTreePath               SML_TreePath,
      parameterTree                   SML_Tree
    }
```

(101) Der Auftragnehmer antwortet auf diese Nachricht per SML_Attention.Res.

5.1.12. SML_Attention.Res

(102) Per SML_Attention.Res werden in einer SML-Antwortdatei potentielle positive Quittungen, Fehlermeldungen, Warnungen oder andere Hinweise des Auftragnehmers an den Auftraggeber gemeldet.

(103) Damit die (Fehler-) Meldungen sowohl automatisch ausgewertet als auch als Text einfach an den Bediener ausgegeben werden können, werden die Elemente ‚attentionNo' und ‚attentionMsg' verwendet.

(104) Die Vergabe global gültiger Nummern wird mit Kapitel 5.1.13 definiert.

(II)
```
SML_Attention.Res               ::=   SEQUENCE
    {
      serverId                        Octet String,
      attentionNo                     Octet String,
      attentionMsg                    Octet String OPTIONAL,
      attentionDetails                SML_Tree OPTIONAL
    }
```

5.1.13. Globale SML-Attentionnumber

(105) Nachstehend folgt die Liste der global definierten Fehlernummern, siehe Kapitel 5.1.12:

Fehlernummer (Darstellung als Bytekette in hexadezimaler Form)	Bedeutung
...	... reserviert.
81 81 C7 C7 E0 00	Beginn applikationsspezifischer Fehlernummern
...	Applikationsspezifische Fehlernummern
81 81 C7 C7 FC FF	Ende applikationsspezifischer Fehlernummern
81 81 C7 C7 FD 00	Siehe Tab. 3
...	Siehe Tab. 3
81 81 C7 C7 FD FF	Siehe Tab. 3
81 81 C7 C7 FE 00	Fehlermeldung, die keiner der nachstehend definierten Bedeutungen zugeordnet werden können.
81 81 C7 C7 FE 01	Unbekannter SML-Bezeichner.
81 81 C7 C7 FE 02	Unzureichende Authentifizierung, Benutzer- / Passwort-Kombination unzulässig.
81 81 C7 C7 FE 03	Zieladresse (‚serverId') nicht verfügbar.
81 81 C7 C7 FE 04	Auftrag (‚reqFileId') nicht verfügbar.
81 81 C7 C7 FE 05	Ein oder mehrere Zielattribut(e) nicht zu beschreiben.
81 81 C7 C7 FE 06	Ein oder mehrere Zielattribut(e) nicht zu lesen.
81 81 C7 C7 FE 07	Kommunikation mit Messstelle gestört.
81 81 C7 C7 FE 08	Rohdaten nicht zu interpretieren.
81 81 C7 C7 FE 09	Gelieferter Wert außerhalb des zulässigen Wertebereichs.
81 81 C7 C7 FE 0A	Auftrag nicht ausgeführt (beispielsweise, weil der angelieferte ‚parameterTreePath' auf ein nicht vorhandenes Element zeigt).
81 81 C7 C7 FE 0B	Prüfsumme fehlerhaft
81 81 C7 C7 FE 0C	Broadcast nicht unterstützt
81 81 C7 C7 FE 0D	Unerwartete SML-Nachricht (z.B. eine SML-Datei ohne ein Open Request)
81 81 C7 C7 FE 0E	Unbekanntes Objekt im Lastgang (der OBIS-Kode in der Anfrage eines Lastgangs verweist auf eine Datenquelle, die nicht im Lastgang aufgezeichnet worden ist)
81 81 C7 C7 FE 0F	Nicht unterstützter Datentyp innerhalb eines Setzbefehles (z.B. entspricht der Datentyp in einer SetProcPar.Req Nachricht nicht dem erwarteten Datentyp)
81 81 C7 C7 FE 10	Optionales Element nicht unterstützt (Ein in SML als OPTIONAL definiertes Element wurde entgegen der von der Applikation getroffenen Annahme empfangen.)
81 81 C7 C7 FE 11	Angefragtes Profil hat keinen einzigen Eintrag

Fehlernummer (Darstellung als Bytekette in hexadezimaler Form)	Bedeutung
81 81 C7 C7 FE 12	Bei Profilanfragen: Endegrenze liegt vor Beginngrenze
81 81 C7 C7 FE 13	Bei Profilanfragen: Im angefragten Bereich liegen keine Einträge vor. In anderen Bereichen ist mindestens ein Eintrag vorhanden
81 81 C7 C7 FE 14	Eine SML-Datei wurde ohne SML-Close beendet.
...	... reserviert.

Tab. 2: Liste globaler Fehlernummern.

(106) Nachstehend folgt die Liste der global definierten Hinweisnummern, siehe Kapitel 5.1.12:

Hinweisnummer (Darstellung als Bytekette in hexadezimaler Form)	Bedeutung
81 81 C7 C7 FD 00	Ok, positive Quittung.
81 81 C7 C7 FD 01	Auftrag wird später ausgeführt und Ergebnis wird per Response-without-Request an Serveradresse übermittelt.
...	... reserviert

Tab. 3: Liste globaler Hinweisnummern.

5.1.14. SML_GetList.Req

(107) Per SML_GetList.Req kann eine im Server vorparametrierte Liste von Datenwerten angefragt werden. Als Antwort ist entweder ein SML_GetList.Res oder ein SML_Attention.Res zu erzeugen.

(JJ)
```
SML_GetList.Req          ::=  SEQUENCE
    {
      clientId                  Octet String,
      serverId                  Octet String    OPTIONAL,
      username                  Octet String    OPTIONAL,
      password                  Octet String    OPTIONAL,
      listName                  Octet String    OPTIONAL
    }
```

(108) Per Attribut ‚listName' kann die gewünschte Liste benannt werden. Es legt per OBIS die gewünschte Größe / Liste fest. Dadurch, dass dieses Element als „optional" gekennzeichnet ist, können einfache Geräte auf dessen Interpretation verzichten und senden lediglich den einzig von ihnen zu liefernden Zählerstand, wobei sie in der Antwort dessen Bedeutung per OBIS angeben müssen.

5.1.15. SML_GetList.Res

(109) Per SML_GetList.Res kann eine Liste vorparametrierter Datenwerte übertragen werden.

(KK)
```
SML_GetList.Res                 ::=  SEQUENCE
        {
          clientId                   Octet String      OPTIONAL,
          serverId                   Octet String,
          listName                   Octet String      OPTIONAL,
          actSensorTime              SML_Time          OPTIONAL,
          valList                    SML_List,
          listSignature              SML_Signature     OPTIONAL,
          actGatewayTime             SML_Time          OPTIONAL
        }
```

(110) Mit dem optionalen Attribut ‚actSensorTime' kann ein Sensor, der diese SML-Nachricht erzeugt, seine eigene, aktuelle Zeitinformation beifügen.

(111) Mit dem optionalen Attribut ‚actGatewayTime' kann ein Gateway, das diese SML-Nachricht transportiert / einer Zwischenverarbeitung unterzieht, seine eigene, aktuelle Zeitinformation anhängen.

(LL)
```
SML_List                        ::=  SEQUENCE OF
        {
        valListEntry                 SML_ListEntry
        }
```

(MM)
```
SML_ListEntry                   ::=  SEQUENCE
        {
          objName                    Octet String,
          status                     SML_Status        OPTIONAL,
          valTime                    SML_Time          OPTIONAL,
          unit                       SML_Unit          OPTIONAL,
          scaler                     Integer8          OPTIONAL,
          value                      SML_Value,
          valueSignature             SML_Signature     OPTIONAL
        }
```

(NN)
```
SML_Status                      ::=  IMPLICIT CHOICE
        {
          status8                    Unsigned8
          status16                   Unsigned16
          status32                   Unsigned32
          status64                   Unsigned64
        }
```

6 SML binary encoding, direkt gepackte Kodierung

(112) SML verwendet eine auf die Zielsetzung, im Datenvolumen möglichst kleine Nachrichten zu produzieren, optimierte Kodierung.

(113) Die Kodierung basiert dazu auf der klassischen Type-Length-Value Struktur. Im Gegensatz zu BER faßt sie aber Type und Length in einen für die meisten Anwendungsfälle auf ein einziges Byte reduzierten, als Type-Length-Field („TL-Field“) bezeichneten, Wert zusammen.

(114) Das Längenfeld beziffert die Anzahl der Elemente, die zu einem einfachen oder komplexen Datentyp gehören.

(115) Bei einfachen Datentypen entspricht die Längenangabe der Anzahl von Bytes, die zu dem Datentyp gehören.

(116) Da das Längenfeld ein Teil des Kodes ist und das TL-Field selbst als Element angesehen werden kann, wird das TL-Field wie ein weiteres Element in der Längenangabe mitgezählt.

(117) Bei komplexen Datentypen (beispielsweise Listen) entspricht die Längenangabe der Anzahl Elemente, die mit dem komplexen Datentyp zusammengefaßt werden (beispielsweise die Anzahl der Listeneinträge). Hier wird das TL-Field selber nicht mitgezählt.

6.1. Type-Length-Field

(118) Das TL-Field legt über die Bitkombination in den höherwertigen Bits des Datenworts fest, ob und wenn ja mit welcher Bedeutung, weitere Bytes mit dem aktuellen Byte zu einem Wort zusammengesetzt werden sollen.

Bitindex	MSB, D7	6	5	4	3	2	1	LSB, D0
Verwendet für Merkmal ‚weiteres Byte zum TL-Field folgt'	1	X	X	X	X	X	X	X
Verwendet für Merkmal ‚kein weiteres Byte zum TL-Field folgt'	0	X	X	X	X	X	X	X
Verwendet für Merkmal ‚Datentyp Octet String' verwenden	X	0	0	0	L	L	L	L
Verwendet für Merkmal ‚Boolean' verwenden	0	1	0	0	L	L	L	L
Verwendet für Merkmal ‚Datentyp Integer' verwenden	X	1	0	1	L	L	L	L
Verwendet für Merkmal ‚Datentyp Unsigned' verwenden	X	1	1	0	L	L	L	L
Merkmal ‚Datentyp List of ...' verwendet.	X	1	1	1	L	L	L	L
Weiteres Byte mit Platz zur Definition zusätzlicher Datentypen folgt; derzeit reserviert	1	1	0	0	L	L	L	L
Reserviert für künftige Verwendung	X	0	0	1	L	L	L	L
Reserviert für künftige Verwendung	X	0	1	0	L	L	L	L
Reserviert für künftige Verwendung	X	0	1	1	L	L	L	L

Tab. 4: Bitkodierung im Type-Length-Field für das erste Byte einer TL-Field-Angabe.

(119) Falls ein zweites (und evtl. weitere) Byte mit einer TL-Field-Angabe folgt, werden die Bits zur Längeninformationen der jeweilig vorangegangenen TL-Field-Angabe nach links geschoben und die „neuen" Bits der folgenden TL-Field-Angabe von rechts her kommend angefügt.

(120) Für das zweite und evtl. folgende weitere Bytes mit einer TL-Field-Angabe sind die Bits zum Datentyp stets wie folgt zu setzen:

Bitindex	MSB, D7	6	5	4	3	2	1	LSB, D0
Verwendet für Merkmal ‚weiteres Byte zum TL-Field folgt'	1	X	X	X	X	X	X	X
Verwendet für Merkmal ‚kein weiteres Byte zum TL-Field folgt'	0	X	X	X	X	X	X	X
Merkmal ‚nachfolgende 4 Bit für die Länge verwenden'	X	0	0	0	L	L	L	L
Reserviert für künftige Zwecke	X	0	0	1	X	X	X	X
Reserviert für künftige Zwecke	X	0	1	0	X	X	X	X
Reserviert für künftige Zwecke	X	0	1	1	X	X	X	X
Reserviert für künftige Zwecke	X	1	X	X	X	X	X	X

Tab. 5: Bitkodierung im Type-Length-Field für das zweite und folgende TL-Field-Bytes.

6.2. Kodierung der Datentypen

6.2.1. Datentyp Octet String

(121) Ein Octet String (eine Bytekette) mit einer Länge von 0 bis max. 14 Bytes wird wie folgt kodiert:

Bitindex	MSB, D7	6	5	4	3	2	1	LSB, D0
Octet String mit einer Anzahl von 0 bis 14 Bytes	0	0	0	0	L	L	L	L

Tab. 6: Bitkodierung im Type-Length-Field für einen Octet String.

(122) Im Anschluss an das TL-Field folgt der Octet String (die Bytekette), wobei das Byte der Bytekette mit dem Index ‚0' zuerst hinter dem TL-Field folgen muss.

(123) Falls der Octet String mehr als 14 Bytes enthält, werden entsprechend der Beschreibung in Kapitel 6.1 vor dem ersten Byte der Bytekette weitere Bytes des TL-Field eingefügt.

6.2.2. Datentypen Integer8, Integer16, Integer32 und Integer64

(124) Diese Integer-Datentypen werden wie folgt kodiert, wobei in der Datenübertragung jeweils ganze Bytes, die führende Nullen (bei positiven Zahlen) oder führende Einsen (bei negativen Zahlen) enthalten, derart weggelassen werden dürfen, dass beim Empfänger keine Verfälschung entsteht:

(OO) Integer8 ::= SEQUENCE
{
TL-Field 0x52,
Datenwert 0xYY
}

(PP) Integer16 ::= SEQUENCE
{
TL-Field 0x53,
Datenwert 0xYY 0xZZ (0xYY ⇔ High-Byte,
0xZZ ⇔ Low-Byte,
Darstellung Big Endian)
}

(QQ) Integer32 ::= SEQUENCE
{
TL-Field 0x55,
Datenwert 0xYY 0xZZ 0xUU 0xVV (0xYY ⇔ High-Byte,
0xVV ⇔ Low-Byte,
Darstellung Big Endian)
}

(RR) Integer64 ::= SEQUENCE
{
TL-Field 0x59,
Datenwert 0xYY ... 0xVV (0xYY ⇔ High-Byte,
0xVV ⇔ Low-Byte,
Darstellung Big Endian)
}

6.2.3. Datentypen Unsigned8, Unsigned16, Unsigned32 und Unsigned64

(125) Diese vorzeichenlosen Integer-Datentypen werden wie folgt kodiert, wobei in der Datenübertragung jeweils ganze Bytes, die führende Nullen enthalten, weggelassen werden dürfen:

(SS) Unsigned8 ::= SEQUENCE
{
TL-Field 0x62,
Datenwert 0xYY
}

(TT) Unsigned16 ::= SEQUENCE
{
TL-Field 0x63,
Datenwert 0xYY 0xZZ (0xYY ⇔ High-Byte,
0xZZ ⇔ Low-Byte,
Darstellung Big Endian)
}

(UU) Unsigned32 ::= SEQUENCE
{
TL-Field 0x65,
Datenwert 0xYY 0xZZ 0xUU 0xVV (0xYY ⇔ High-Byte,
0xVV ⇔ Low-Byte,
Darstellung Big Endian)
}

(VV) Unsigned64 ::= SEQUENCE
{
TL-Field 0x69,
Datenwert 0xYY ... 0xVV (0xYY ⇔ High-Byte,
0xVV ⇔ Low-Byte,
Darstellung Big Endian)
}

6.2.4. Datentyp Boolean

(126) Dieser Boolean-Datentyp wird wie folgt kodiert:

(WW) Boolean ::= SEQUENCE
{
TL-Field 0x42,
Datenwert 0xYY (0x00 ⇔ ‚false‘,
alles andere ⇔ ‚true‘)
}

6.2.5. Datentyp List of ...

(127) Falls Listen (oder Arrays oder Structures) kodiert werden sollen, enthält das TL-Field zu Beginn der Liste / des Arrays / der Struktur die Anzahl der Elemente. Daran schließt sich das erste Element der Liste / des Arrays / der Struktur an, das ebenfalls wieder mit einem TL-Field beginnt.

(128) Die Elementangabe im TL-Field von ‚List of ...‘ weist immer auf das nächste TL-Field, das hinter allen Elementen von ‚List of ...‘ folgt. Der Wert ‚überspringt‘ damit alle TL-Fields, die je Element von ‚List of ...‘ innerhalb der ‚List of ...‘-Daten stehen.

(XX) List of ... ::= SEQUENCE
{
TL-Field 0x7z (‚z‘ ⇔ Anzahl der Elemente der Liste)
}

6.3. Kodierung besonderer Merkmale

6.3.1. Merkmal Ende einer SML-Nachricht

(129) Wegen der Verkettung der einzelnen Attribute einer SML-Nachricht kann über das TL-Field das Ende einer SML-Nachricht definiert / gefunden werden:

(YY) EndOfSmlMessage ::= 0x00 ⇔ „Eintrag ohne TL-Field“ ⇔ Ende

6.3.2. Merkmal SEQUENCE

(130) Bei Sequenzen werden grundsätzlich alle Komponenten der Sequenz in der Reihenfolge ihrer Auflistung in der ASN.1-Definition in den kodierten Datenstrom übernommen.

(131) Eine Sequence wird generell als Struktur zusammengefaßt und damit durch den Datentyp ‚List of ...‘ eingeleitet (siehe Kapitel 6.2.5).

6.3.3. Merkmal CHOICE

(132) Bei Auswahllisten wird das TAG zum ausgewählten Element Unsigned kodiert. Das TAG erhält, genau wie jede andere Unsigned-Komponente auch, sein TL-Field vorangestellt.

(133) Elemente vom Typ SML_MessageBody kodieren das TAG als Unsigned32, alle anderen als Unsigned8.

(134) Die CHOICE selbst wird als Struktur behandelt und damit durch den Datentyp ‚List of ...‘ eingeleitet (siehe Kapitel 6.2.5). Sie führt also immer zu einer Struktur mit zwei Elementen: Das erste Element ist das TAG und das zweite das damit definierte CHOICE-Element.

6.3.4. Merkmal OPTIONAL

(135) Falls in der ASN.1-Definition eine Komponente als ‚OPTIONAL‘ gekennzeichnet wurde, ist diese in den Datenstrom mit dem TL-Field ‚0x01‘ zu setzen. Dabei wird für dieses TL-Field immer der Datentyp ‚Octet String‘ angenommen und die Element-Angabe direkt hinter dem TL-Field positioniert.

7 XML - Kodierung

(136) Alternativ kann SML auch per XML-Kodierung transportiert werden. Gegenüber der Darstellung nach Kapitel 6 bewirkt die XML-Kodierung prinzipiell ein deutlich erhöhtes Datenvolumen, liefert aber standardisiert lesbare SML-Dateien.

(137) Das Kapitel soll bei Bedarf im weiteren Verlauf der Arbeiten mit Inhalt gefüllt werden; die XML-SML-Transformation wird aktuell nicht benötigt.

8 SML-Transport-Protokoll

8.1. Version 1

(138) Zur Übertragung von SML-Nachrichten über ungesicherte Verbindungen, wie beispielsweise direkte optische Ablesung per MDE vor Ort, Ablesung über PSTN-Mo-

dem, GSM-Modem oder vergleichbare Strecken, wird das SML-Transport-Protokoll definiert.

(139) Es kann ebenfalls bei gesicherten Verbindungen (beispielsweise TCP) angewendet werden.

(140) Es folgt als Streaming-Protocol dem Ansatz, in den Endgeräten den Datenverkehr ‚on the fly' zu kodieren und damit auf den Einsatz umfangreicher Puffer verzichten zu können.

(141) Das Regelwerk wird wie folgt definiert:

- Beginn, Ende sowie weitere Merkmale einer Nachricht werden über Escape-Sequenzen gekennzeichnet.
- Eine Escape-Sequenz wird mit einer Escape-Zeichenfolge eingeleitet. An diese Escape-Zeichenfolge schließt sich das der Nachricht hinzugefügte Merkmal (Beginn, Ende, ...) an.
- Tritt im Nutzlastdatenstrom selbst die Escape-Zeichenfolge auf, wird diese selbst als Escape-Sequenz übertragen. In diesem Fall ist das der Nachricht hinzugefügte Merkmal die Escape-Zeichenfolge (sie wird damit zweimal nacheinander übertragen).
- Die Anzahl der Bytes im Nutzlastdatenstrom wird am Ende immer auf eine restfrei durch die Anzahl der Bytes der Escape-Zeichenfolge teilbare Menge erweitert[10]. Zur Erweiterung werden jeweils Bytes mit dem Inhalt ‚00' (hex) verwendet.

(142) Als Escape-Zeichenfolge wird die Byte-Kette ‚1b 1b 1b 1b' (Angabe in hex.) festgelegt.

(143) Folgende Escape-Merkmale sind definiert:

(144)

Pos.	Escape-Merkmal (Angabe in hex)	Bedeutung / Hinweis
1	1b 1b 1b 1b	Kennzeichnet den Fall, dass die Escape-Sequenz selbst im Nutzdatenstrom enthalten ist.
2	01 01 01 01	Leitet Übertragung der Version 1 als Datenstrom ein. Kennzeichnet das Merkmal ‚Beginn einer Nachricht'.
3	02 TT UU VV	Leitet Übertragung der Version 2 mit Blocktransfer ein, siehe Kapitel 8.2.
4	03 00 RR RR	Wird nur in Zusammenhang mit der Version 2 verwendet, und legt das zu verwendende Timeout fest, siehe Kapitel 8.2.
5	04 00 SS SS	Wird nur in Zusammenhang mit der Version 2 verwendet, und legt die zu verwendende Blocksize fest, siehe Kapitel 8.2.
6	1a XX YY ZZ	Kennzeichnet das Merkmal ‚Ende einer Nachricht'. ‚XX' ⇔ Kann Werte des Bereichs ‚00', ‚01', ‚02' oder ‚03' annehmen und liefert die Anzahl von Bytes, die am

[10] Da die Escape-Zeichenfolge aus 4 Bytes besteht, wird die Anzahl der Bytes der Nutzlast ebenfalls immer so erweitert, dass eine Division modulo 4 genau 0 liefert. Damit muss der Absender entweder kein Byte, oder ein, zwei oder drei Bytes anfügen.

Pos.	Escape-Merkmal (Angabe in hex)	Bedeutung / Hinweis
		Ende der Nutzlast angefügt worden sind, um die Anzahl der Bytes der Nutzlast restfrei durch die Anzahl der Bytes einer Escape-Zeichenfolge teilen zu können. ‚YY ZZ' ⇔ Kann Werte im Bereich ‚00..FF' annehmen und enthält die Prüfsumme über die ganze Nachricht. YY ist dabei das Most Significant Byte und ZZ das Least Significant Byte der Prüfsumme.
7	Alle anderen Kombinationen	Reserviert für künftige Erweiterungen.

Tab. 7: Escape-Merkmale zum SML-Transport-Protokoll.

(145) Die Prüfsumme ist nach CCITT-CRC16 zu berechnen. Sie wird über alle Bytes des Datenstroms im SML-Transportprotokoll mit Ausnahme der letzten beiden Bytes (und damit ohne die Bytes der Prüfsumme selber) berechnet.

(146) Die Berechnung erfolgt gemäß DIN EN 62056-46.

(147) Erfolgt die Datenbeschaffung über TCP- oder UDP-Verbindungen, wird das SML-Transport-Protokoll zur Kennzeichnung zusammenhängender SML-Dateien verwendet.

8.2. Version 2

(148) Zur Übertragung von SML-Dateien über ungesicherte Halbduplex-Verbindungen kann das SML-Transport-Protokoll der Version 2 verwendet werden. Dieses bietet einen simplen Mechanismus zur Fluss-Steuerung, so dass der Sender seine Ausgaben an die Anfordernisse von – möglicherweise mit wenig Ressourcen ausgestattete – Clients anpassen kann.

(149) Das SML-Transport-Protokoll der Version 2 ist damit ebenfalls anzuwenden, wenn SML-Dateien über optischen Strecken, vergleichbar DIN EN 62056-21 zu übertragen sind.

(150) Das zur Lösung dieser Anforderungen realisierte Konzept teilt die zu übertragende SML-Datei in Blöcke auf und verwendet Timeouts zum Restart bei Fehlern. Sowohl die Blockgröße als auch die Timeouts werden bei Beginn der Übertragung zwischen Sender und Empfänger ausgehandelt. Als Timeout und minimale Blockgröße für das Aushandeln werden verwendet:
Inital zu verwendendes Timeout: 5 s;
Initial zu verwendende minimale Blockgröße: 32 Bytes.

(151) Die Version 2 verwendet drei gegenüber der Version 1 zusätzlich definierte ESC-Sequenzen.

8.2.1. Einleitung der Übertragung nach Version 2

(152) Zur Unterscheidung der Version 2 des SML-Transport-Protokolls von der Version 1 (siehe Kapitel 8.1) wird als Einleitung die Sequence

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

verwendet (alle Angaben in hex, siehe Tab. 7).

(153) Die Elemente „TT UU VV“ dienen dabei Kennzeichnung von Blöcken, und SML-Dateien.

8.2.1.1. Kennzeichnung von Blöcken

(154) Blöcke werden per Element „TT“, siehe Tab. 7, gekennzeichnet. Die Kennzeichnung verwendet folgende Merkmale:

- Bit 7, MSB: 0 ⇔ Sendeblock,
  1 ⇔ Empfangen als ACK;
- Bit 6: 0 ⇔ weitere Blöcke folgen,
  1 ⇔ letzter Block der SML-Datei.
- Bit 5 … Bit 0, LSB: Blocknummer,
  beginnend mit 0x00, rollierend bei 0x3F auf 0x01.

8.2.1.2. Kennzeichnung von SML-Dateien

(155) SML-Dateien werden per Element „UU“, siehe Tab. 7, gekennzeichnet. Die Kennzeichnung verwendet folgende Merkmale:

- Die erste über die Strecke zu übertragende SML-Datei erhält die Kennzeichnung ‚0x00’;
- Mit jeder weiteren über die Strecke zu übertragenden SML-Datei wird das Merkmal um eins inkrementiert;
- Wird der Wert ‚0xFF’ erreicht, ist danach wieder mit ‚0x00’ zu beginnen.

8.2.1.3. Merkmal ‚VV’

(156) Das Element „VV“, siehe Tab. 7, ist für künftige Erweiterungen reserviert und immer auf ‚0x01’ zu setzen.

8.2.2. Vereinbarung des zu verwendenden Timeouts

(157) Mit Beginn einer Übertragung schlägt der Sender das zu verwendende Timeout vor. Er verwendet dazu eine ESC-Sequenz der Art

1b 1b 1b 1b 03 00 RR RR

wobei per „RR RR“ das von ihm vorgeschlagene Timeout in ‚ms’ anzugeben ist.

(158) Das Timeout ist in der Form

1b 1b 1b 1b 03 00 High-Byte Low-Byte

in die ESC-Sequenz einzutragen.

(159) Der Empfänger beantwortet diese ESC-Sequenz mit dem von ihm tatsächlich gewählten Timeout, wobei er nur denselben oder einen größeren Zahlenwert wählen darf. Der Sender hat das vom Empfänger abschließend festgelegte Timeout zu verwenden.

8.2.2.1. Vereinbarung der maximal zulässigen Blocksize

(160) Mit Beginn einer Übertragung schlägt der Sender die maximal zulässige Blocksize vor. Er verwendet dazu eine ESC-Sequenz der Art

1b 1b 1b 1b 04 00 SS SS

wobei per „SS SS“ die von ihm vorgeschlagene Blocksize in ‚Byte’ anzugeben ist.

(161) Die Blocksize ist in der Form

1b 1b 1b 1b 04 00 High-Byte Low-Byte

in die ESC-Sequenz einzutragen.

(162) Der Empfänger beantwortet diese ESC-Sequenz mit der von ihm tatsächlich gewählten Blocksize, wobei er nur denselben oder einen kleineren Zahlenwert wählen darf. Der Sender hat die vom Empfänger abschließend festgelegte Blocksize zu verwenden.

8.2.3. Prozess zum Aufbau der Übertragung

(163) Eine Übertragung nach Version 2 wird durch folgenden Prozess eingeleitet:

- Der Sender versendet den ersten Datenblock, der exakt wie folgt aufzubauen ist:
  1b 1b 1b 1b 02 00 UU VV
  1b 1b 1b 1b 03 00 RR RR (mit RR RR proposed timeout in ms)
  1b 1b 1b 1b 04 00 SS SS (mit SS SS proposed block size in Byte)
  1b 1b 1b 1b 1a xx yy zz
- Der Empfänger quittiert diesen ersten Datenblock:
  1b 1b 1b 1b 02 80 UU VV
  1b 1b 1b 1b 03 00 rr rr (mit rr rr confirmed timeout in ms)
  1b 1b 1b 1b 04 00 ss ss (mit ss ss confirmed block size in Byte)
  1b 1b 1b 1b 1a xx yy zz

8.2.4. Prozess zum Ablauf einer Übertragung

(164) Eine Übertragung nach Version 2 wird wie vorstehend beschrieben eingeleitet und arbeitet danach als „Ping-Pong“ von gesendeten und quittierten Datenblöcken.

(165) Der Zustandsautomat fällt mit einfachem Timeout an den Beginn des aktuellen Übertragungsschritts zurück, falls er keine Rückmeldung auf den zuletzt versendeten Block erhält.

(166) Bei Fehlern ist nach doppeltem Timeout die komplette Dateiübertragung zu wiederholen.

8.2.5. Beispiel zum Ablauf des Übertragungsvorgangs nach Version 2

(167) Als Beispiel eines Übertragungsvorgangs kann nachstehender Ablauf angesehen werden:

- Start der Übertragung mit:
  1b 1b 1b 1b 02 00 UU VV
  1b 1b 1b 1b 03 00 RR RR (mit RR RR proposed timeout in ms)

1b 1b 1b 1b 04 00 SS SS (mit SS SS proposed block size in Byte)
1b 1b 1b 1b 1a xx yy zz

- Wird der Start der Übertragung nicht korrekt dekodiert / empfangen, erfolgt keine Reaktion durch den Empfänger. Der Sender muss den Beginn der Übertragung erneut einleiten.
- Übertragung confirmed mit:
  1b 1b 1b 1b 02 80 UU VV
  1b 1b 1b 1b 03 00 rr rr (mit rr rr confirmed timeout in ms)
  1b 1b 1b 1b 04 00 ss ss (mit ss ss confirmed block size in Byte)
  1b 1b 1b 1b 1a xx yy zz
- Erste Nutzlast senden mit:
  1b 1b 1b 1b 02 01 UU VV
  Nutzlast
  1b 1b 1b 1b 1a xx yy zz
- Erste Nutzlast bestätigen mit ACK:
  1b 1b 1b 1b 02 81 UU VV
  1b 1b 1b 1b 1a xx yy zz
- Erste Nutzlast abweisen mit NAK:
  1b 1b 1b 1b 02 80 UU VV (mit „alter“ Blocknummer in 0x80)
  1b 1b 1b 1b 1a xx yy zz
- Zweite Nutzlast senden mit:
  1b 1b 1b 1b 02 02 UU VV
  Nutzlast
  1b 1b 1b 1b 1a xx yy zz
- Zweite Nutzlast bestätigen mit ACK:
  1b 1b 1b 1b 02 82 UU VV
  1b 1b 1b 1b 1a xx yy zz
- Zweite Nutzlast abweisen mit NAK:
  1b 1b 1b 1b 02 81 UU VV (mit „alter“ Blocknummer in 0x81)
  1b 1b 1b 1b 1a xx yy zz
- Dritte und letzte Nutzlast senden mit:
  1b 1b 1b 1b 02 43 UU VV
  Nutzlast
  1b 1b 1b 1b 1a xx yy zz
- Dritte und letzte Nutzlast bestätigen mit ACK:
  1b 1b 1b 1b 02 C3 UU VV
  1b 1b 1b 1b 1a xx yy zz
- Dritte und letzte Nutzlast abweisen mit NAK:
  1b 1b 1b 1b 02 82 UU VV (mit „alter“ Blocknummer in 0x81)
  1b 1b 1b 1b 1a xx yy zz