



open telematics interface

## **Standardized Message Structure**

### **Structure Specification Version 1**



<b>Title</b>	Specification optiSMS open telematics interface Standardized Message Structure
<b>Publisher</b>	DB Systel GmbH IT/TK Innovation and Evaluation Center Rundeturmstraße 10  D-64283 Darmstadt
<b>Contact</b>	Michael Baranek
<b>Authors</b>	Michael Baranek, DB Systel GmbH Olivier Binet, Saphymo Benoit Petit-Forestier, Saphymo Klaus-Peter Thiele, DB Systel GmbH Felix Schwarz, Fela Management AG Hans-Peter Wepfer, Fela Management AG
<b>Editorial</b>	Michael Baranek, DB Systel GmbH

## Index of content

<b>1 Introduction</b>	<b>6</b>
1.1 Preface	6
1.2 Goals	6
1.3 License Agreement	7
1.4 References	7
1.5 Abbreviations	7
1.6 Scope	7
1.7 Notation	7
<b>2 Requirements</b>	<b>8</b>
2.1 Req-01 Information shall be transported between MU and CS in both directions	8
2.2 Req-02 Data transport volume low to medium	8
2.3 Req-03 optiSMS shall support communication in GSM mobile networks	8
2.4 Req-04 Security	8
2.5 Req-05 Quality of Service	8
2.6 Req-06 Data integrity, consistency	8
2.7 Req-07 Customization, flexibility (Customer/project specific elements can be added)	8
2.8 Req-08 MU shall be able to communicate with one CS to different AS	8
2.9 Req-09 MU shall be accessible from different AS with one CS	8
2.10 Req-10 MU shall be able to transmit information immediately	8
2.11 Req-11 MU shall be accessible depending on its mode	8
2.12 Req-12 MU shall be able to transfer foreign information in both directions	9
2.13 Req-13 Networks and standards	9
<b>3 Terms and definitions</b>	<b>10</b>
3.1 Element	10
3.2 Identifier	10
3.3 Attribute	10
3.4 Application Field	10
3.5 Transaction	10
3.6 Operation	10
3.7 IP Technology	10
3.8 GSM-SMS Technology	11
3.9 Node	11
3.10 Encoder/Decoder	11
3.11 Mobile Unit	11
3.12 Communication Server	11



3.13 Application Server	11
3.14 SET/GET	11
3.15 Application Designer	11
<b>4 General</b>	<b>12</b>
4.1 System Overview	12
4.2 Typical Architecture	12
4.2.1 Transmission technologies	12
4.2.2 Connection to CS	12
4.2.3 General Use Cases of optiSMS	13
<b>5 Architecture</b>	<b>14</b>
5.1 Overview	14
5.2 Communication server	15
5.2.1 Required Features	15
5.2.2 Optional Features	15
5.3 Connector	16
5.3.1 Message Structure	16
5.3.2 Messages	17
5.3.3 Mechanisms	17
5.4 Node	18
<b>6 Payload</b>	<b>19</b>
6.1 Elements	19
6.2 Attribute Structure	20
6.3 Transactions and Operations	20
6.4 Operations	22
6.4.1 GetRequest	22
6.4.2 GetResponse	22
6.4.3 SetRequest	22
6.4.4 SetResponse	22
6.4.5 Element related errors	23
6.4.6 General error	23
6.5 Element Naming Schema (Identifier)	23
6.6 Protocol Extensibility and Versioning	23
6.7 OptiSMS Specification Version 1 according ASN.1	25
<b>7 optiSMS Layer Stack</b>	<b>27</b>
7.1 General	27
7.2 HTTP Sequence Diagram	28
7.3 Application Layer	29
7.4 Presentation Layer	29
7.4.1 Description	29
7.4.2 Version and coding with IP technology	29

7.4.3 Version and coding with GSM-SMS technology	30
7.4.4 Error Handling	30
7.5 Session Layer	31
7.5.1 Description	31
7.5.2 IP Technology	31
7.5.3 GSM-SMS Technology	33
7.5.4 Error Handling	33
7.6 Transport Layer	34
7.6.1 Description	34
7.6.2 IP Technology	34
7.6.3 GSM-SMS Technology	34
7.6.4 Error Handling	34
7.7 Network Layer	35
7.7.1 GSM-SMS technology	35
7.8 Examples	35
7.8.1 IP Example SetRequest→SetGetResponse CS→MU→CS	35
7.8.2 IP Example SetRequest→SetResponse MU→CS→MU	35
7.8.3 GSM-SMS Example SetRequest→SetResponse MU→CS→MU	35
<b>8 Appendix</b>	<b>36</b>
8.1 Application Content Example	36
8.2 Complete Examples	40
8.2.1 CS→MU SetRequest/SetResponse with IP	40

## 1.1 Preface

In 2007 the IT/TK Innovation and Evaluation Center of DB Systel GmbH started an international project to define and develop a standardized interface for telematics applications. The goal is to define a standardized and generic protocol. This protocol is to enable the exchange between servers and decentralized telematics systems in order to build systems through the combination of devices from different suppliers. The name of this upcoming industry standard is optiSMS – open telematics interface – Standardized Message Structure.

For the first time the market leaders for telematics systems in the rail cargo industry could be brought together to ensure interoperability by developing an open industry standard with a high potential for the future. Furthermore, upon publication optiSMS will be introduced and used by the project partners for all new deliveries.

The protocols used today by the suppliers of according telematics systems are usually proprietary, company-individual implementations. These are primarily oriented towards an optimized usage of carrier technology like GSM-SMS as well as towards a user-specific and static implementation of message types. For this the message types use a bit-oriented communication of user data and are tightly coupled with the actual functional application implementation within the autonomous telematics system.

For the potential user this means that he has to carry a certain effort for adapting these proprietary protocols to his special functional requirements for mapping his business logic, which usually means much time for coordination and engineering. If the potential user wants to integrate solutions from multiple suppliers, he has to arrange for special agreements for disclosing the implemented protocols and he should also plan for intensive functional- and integration tests. It should be noted that this is not an exotic scenario but current common practice.

The project team has decided to disregard implemented proprietary protocols for developing a standardized interface for telematics applications and instead to pursue a completely new and future-proof approach. Besides optimizing the user data transmission to reduce communication costs, the logical separation of the protocol from its data transport layer was essential. Another requirement, which – according to the project team – is essential for the success of a future-proof protocol, is to provide the standardized interface for telematics applications free of license fees, so that it can be used by anyone.

After reviewing the market and the telematics projects implemented by the project partners, a strong trend can be identified. While projects implemented so far have relied on using GSM-SMS as a data transport service, the development in the upcoming months and years is moving towards package oriented data transfer using GSM-GPRS.

---

## 1.2 Goals

The main goal is to define a standardized and generic protocol which ensures interoperability by developing an open industry standard with a high potential for the future.

This protocol has to enable the data exchange between CS and decentralized MUs in order to build systems through the combination of devices from different suppliers.

From a strategic point of view, standardized protocol has created an environment in which innovative telematics services can be developed and delivered cost effectively and hence to increase the range of economic telematics services available to manufacturers and users.

Regarding the uses case of railway freight telematics applications and especially the autarky use cases for rail freight cars additional goals were defined. One of these additional goals is that the communication between MU and CS shall be energy efficient. This means, that it is necessary to reduce the size of data structures and overhead to a minimum. In the same context of reducing the size of data structures and overhead a further additional goal is to provide a cost efficient communication between MU and CS.

Besides optimizing the user data transmission to reduce communication costs, the logical separation of the protocol from its data transport layer is essential.

Another additional goal is essential for the success of a future-proof protocol, is to provide the standardized interface for telematics applications free of license fees, so that it can be used by anyone.

From a functional point of view, additional goals were defined. The goals regarding the communication layer is that future developments in communication technology (while honoring the OSI layer model) can be used as transport medium for this protocol. This explicitly includes the use of satellite based data transfer services.

### 1.3 License Agreement

The open telematics interface - Standardized Message Specifications, whether in paper or electronic format and for use, are made available subject to the terms of the license agreement, which can be downloaded on the web-side [www.opentelematcis.info](http://www.opentelematcis.info).

### 1.4 References

- [1] ISO/IEC 7498-1:1994, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*
- [2] ISO/IEC 8824-1:2002... ISO/IEC 8825-4:2002, *Information technology — Abstract Syntax Notation One (ASN.1) and ASN.1 Encoding Rules*  
<http://asn1.elibel.tm.fr/en/standards/index.htm>
- [3] RFC 3986, *Information technology — Uniform Resource Identifier (URI)*
- [4] RFC 2616, *Information technology — Hypertext Transfer Protocol HTTP/1.1*
- [5] ASN.1 Online Syntax Checker: <http://asn1.elibel.tm.fr/tools/asnp/>
- [6] Free ASN.1 Syntax Checker: <http://www.oss.com/products/syntax1.html>
- [7] Open Source ASN.1 Compiler: <http://lionet.info/asn1c/>
- [8] NRC-TR-2006-005, *Information technology — Providing HTTP Access to Web Servers Running on Mobile Phones*

### 1.5 Abbreviations

AP	Application Server
API	Application Protocol Interface – Information technology
BER	Basic Encoding Rule, ASN.1 encoding and decoding scheme (see [2])
CS	Communication Server
MU	Mobile Unit
PER	Packed Encoding Rule, ASN.1 encoding and decoding scheme (see [2])
UPER	Unaligned Packed Encoding Rule, ASN.1 encoding and decoding scheme (see [2])
XER	XML Encoding Rule, ASN.1 encoding and decoding scheme (see [2])

### 1.6 Scope

This technical specification defines the mechanism for the information exchange over a Cellular Network (CN) between Mobile Unit (MU) and Communication Server (CS) for railway freight wagon telematics applications.

The exchanged information defines the behaviour of MU and/or informs the CS about status and events of the MU.

This document specifies the structure of the information that has to be exchanged and the dependant architecture (“how”).

A complete optiSMS application specification document defines all the content of the information that has to be exchanged (“what”).

### 1.7 Notation

This document uses the ASN.1 syntax notation for the definition of elements and operations [1].

## 2 Requirements

The optiSMS structure must fulfil the following requirements.

---

### 2.1 Req-01 Information shall be transported between MU and CS in both directions

The protocol shall support both directions.

---

### 2.2 Req-02 Data transport volume low to medium

The protocol supports the following data volume per operation depending on transmission technology:

- IP: Data size is limited by hardware capabilities but shall be at minimum 1kByte.
- GSM-SMS: 1 SMS payload (140Byte in PDU mode)

---

### 2.3 Req-03 optiSMS shall support communication in GSM mobile networks

The protocol must not use any special network feature.

---

### 2.4 Req-04 Security

The following measurements shall be taken to increase the system security:

- Uses binary format
- The address of the origination must be known by the MU in advance. Data from unknown or unauthorised CS is rejected.

---

### 2.5 Req-05 Quality of Service

The protocol shall support an acknowledge mechanism.

---

### 2.6 Req-06 Data integrity, consistency

The data shall be checked for integrity.

This shall be done with CRC on transport layer and with the binary decoder on application level. For the CRC on GSM-SMS a proprietary mechanism shall be implemented.

---

### 2.7 Req-07 Customization, flexibility (Customer/project specific elements can be added)

The protocol shall supports extensibility of the element that can be added at any later time without effect to prior projects or users.

---

### 2.8 Req-08 MU shall be able to communicate with one CS to different AS

The data from the CS to different AS may be routed via optiXML.

---

### 2.9 Req-09 MU shall be accessible from different AS with one CS

Different AS shall be able to communicate with a MU. OptiXML may be used to communicate from the AS to the CS. The optiSMS protocol shall be used for communication from the CS to the MU.

---

### 2.10 Req-10 MU shall be able to transmit information immediately

The MU shall be able to transmit information immediately or store and transmit them later. The storage capacity depends on hardware capability of the MU.

---

### 2.11 Req-11 MU shall be accessible depending on its mode

- MU is always online (if GSM is available).

- MU is accessible when it initiates the communication due to pending data. In this case the MU is accessible for a variable duration.

---

**2.12 Req-12 MU shall be able to transfer foreign information in both directions**

The MU shall be able to address internal modules or external devices (nodes) to route information.

---

**2.13 Req-13 Networks and standards**

GSM-SMS and/or GSM-GPRS shall be supported.

## 3 Terms and definitions

In the optiSMS protocol the following terms and definitions are used in the described context.

---

### 3.1 Element

An element is the base data container. It consists of:

- Unique identifier
- One or several attributes

---

### 3.2 Identifier

The unique identifier of an element.

---

### 3.3 Attribute

An attribute is either an atomic data value (e.g second, pressure, distance) or a structured set of such values. Sets of attributes may be used to build a new attribute.

One or multiple attributes build an element.

---

### 3.4 Application Field

Application fields (e.g Railway Telematics, Container Telematics, Remote Metering, etc) have different needs to the content exchanged with optiSMS. The use of one recommended default element set for such an application field helps to achieve interoperability on the air and to reduce the implementation effort.

---

### 3.5 Transaction

A transaction describes the chronological sequence of request and answer. The concatenation of the following operations are transactions:

- GetRequest→GetResponse
- SetRequest→SetResponse

The optiSMS implementation shall handle timeouts and if necessary perform a retry depending on the application.

---

### 3.6 Operation

An operation is the entity that is transported from the sender to the receiver. An operation holds data, acknowledge or error information.

optiSMS supports the following operations:

- GetRequest
- GetResponse
- SetRequest
- SetResponse

Transactions are build out of operations.

---

### 3.7 IP Technology

The data exchange is performed with the standardised protocol HTTP over TCP/IP.

---

### **3.8 GSM-SMS Technology**

The data exchange is performed with the SMS (Short Message Service)/GSM technology.

---

### **3.9 Node**

A functional part of the MU or a separated module (e.g. a pressure sensor module) linked to the MU with a dedicated communication link (e.g. RF). OptiSMS allows to address such a node.

Therefore nodes communicate with the CS through the MU and vice-versa.

---

### **3.10 Encoder/Decoder**

The functionality (software component) that converts the binary data stream coming from the presentation layer into objects of the application layer and vice-versa. The encoding/decoding is performed according the rules of ASN.1 and the optiSMS specification.

---

### **3.11 Mobile Unit**

The mobile unit (MU) is the mobile telematic unit placed on the vehicle.

---

### **3.12 Communication Server**

The communication server (CS) is a part of the central backend system managing the wireless communication. The CS interacts with the application server and MU.

---

### **3.13 Application Server**

The application server handles the central business logic. To communicate with the MU it uses the communication server.

---

### **3.14 SET/GET**

SET: The originator requests to modify some data on the destination.

GET: The originator requests to receive some data from the destination.

---

### **3.15 Application Designer**

The application designer is a person specifying the elements according the application/project requirements.

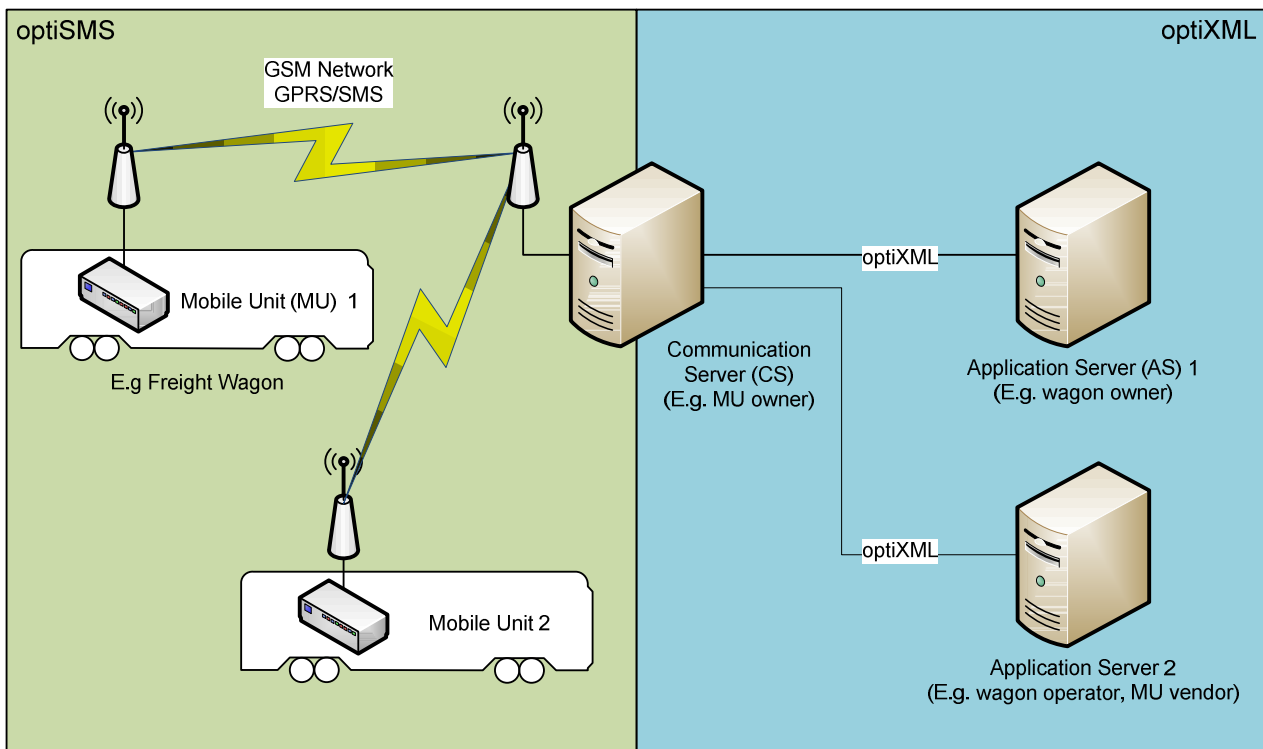
## 4.1 System Overview

Several mobile units (MU) are logically linked with one or several application servers (AS). Both counterparts can exchange information in both directions (MU↔AS).

Physically each MU is linked over a wireless GSM network with one communication server (CS). For this link optiSMS shall be used.

The application servers communicate with the CS over an IP network. The optiXML protocol may be used for this link and thus is out of scope of this specification.

Figure 1: System Overview



The GSM communication between MU and CS is performed either with IP/GSM-GPRS or with GSM-SMS technology.

## 4.2 Typical Architecture

### 4.2.1 Transmission technologies

The MU (e.g. a freight wagon) communicates directly over GSM network with a CS. Therefore mixed technology based on HTTP/TCP/IP/GSM-GPRS or GSM-SMS is used. The CS may implement one or both transmission technologies.

### 4.2.2 Connection to CS

The CS acts as a concentrator for multiple access on the left from different MU (optiSMS) and on the right from different AS (optiXML). The CS is in charge of allowing or refusing an application server to access the MU for security or cost reason. Communication originated from the MU must be passed through the CS. The CS acts as a protocol converter from optiSMS to optiXML and vice-versa. The CS may store the access information of all operating mobile units. This decoupling of application servers and MU is handled by the CS.

### **4.2.3 General Use Cases of optiSMS**

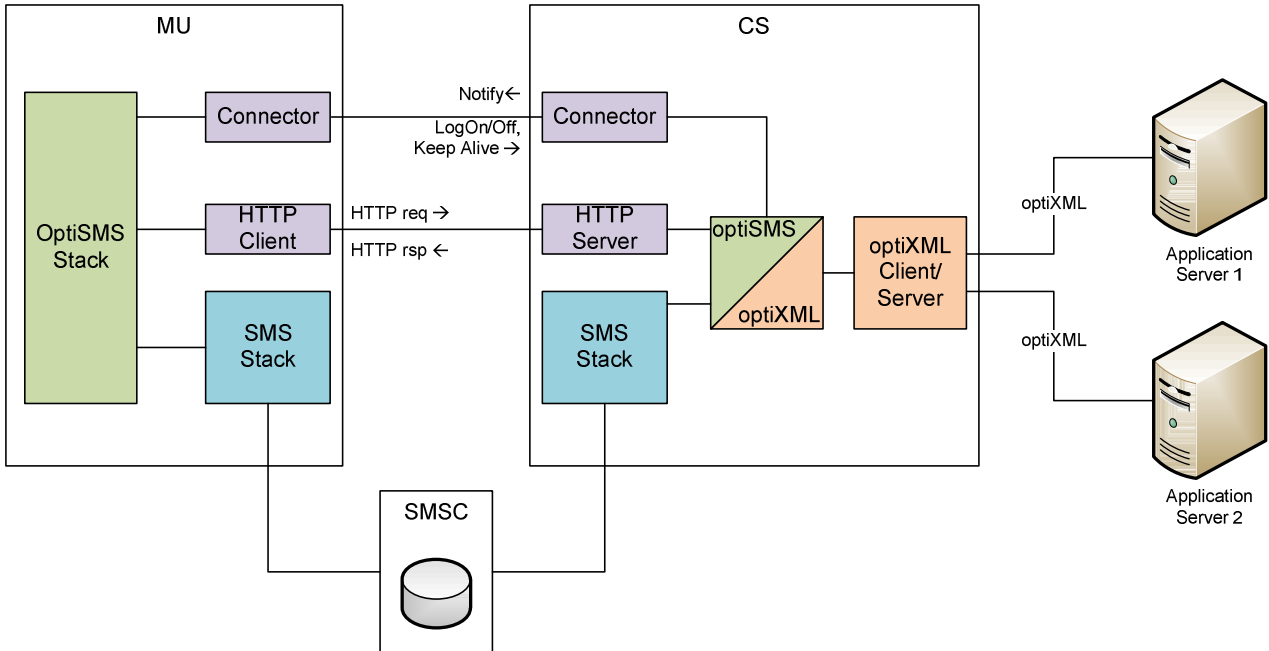
The following general use cases are applicable with optiSMS:

- (1) A communication server requests information from the MU
- (2) A communication server requests to modify information on the MU.
- (3) A MU requests to inform the communication server about state changes.

## 5.1 Overview

The following figure gives a more detailed view of the MU and CS architecture.

Figure 2: Detailed Architecture



When using IP technology the optiSMS architecture requires on the MU side a HTTP client and on the CS side a HTTP server. The transmission is always originated from the MU.

One use case is that the CS needs to start a transaction to the MU. For the time being connecting to a mobile device over GSM-GPRS is not applicable due to GSM-GPRS IP address translation schema. The connecting has always to be initiated by the MU. Therefore the architecture requires on both sides a so-called "Connector". The connector on the MU side initiates the connection when it gets online. The CS can notify the MU over this connection when a transaction is pending.

When using GSM-SMS technology the architecture requires a proprietary GSM-SMS stack on both sides.

If the MU sends a request to a CS, the CS (not the AS) shall respond to the MU.

---

## 5.2 Communication server

### 5.2.1 Required Features

Apart the HTTP server and GSM-SMS stack modules optiSMS requires the following features on the CS:

#### 5.2.1.1 Accessibility

- 1) The communication server shall handle the accessibility from the CS to the MU.
- 2) This shall be done with the connector concept (see paragraph 5.3)
- 3) The communication server shall handle the state of all the MUs in operation (e.g. online/offline)

#### 5.2.1.2 Addressability

The communication server has to handle how to address all the MUs in operation. This requires a directory of:

- Unique Identifier of the MU
- Phone number of MU

### 5.2.2 Optional Features

The CS may implement the following features depending on the application requirements.

#### 5.2.2.1 Notification Services for optiXML

Application servers may register to the CS to be notified on state changes of the MU. This allows signalling asynchronously events like MU online or offline.

#### 5.2.2.2 Security

The CS is an additional layer of defence and can protect against some OS and WebServer specific attacks (e.g. DOS, IP spoofing).

#### 5.2.2.3 Cost Control

The CS may implement some cost control features. This gives the owner of the SIM card the possibility to control the cost e.g. when the monthly limit of byte transfer is reached.

An other possibility is to keep account the data volume used by an operator and to charge them.

#### 5.2.2.4 Load Balancing

Communication Load balancing may be applied e.g. to split requests from the different MUs to the CS.

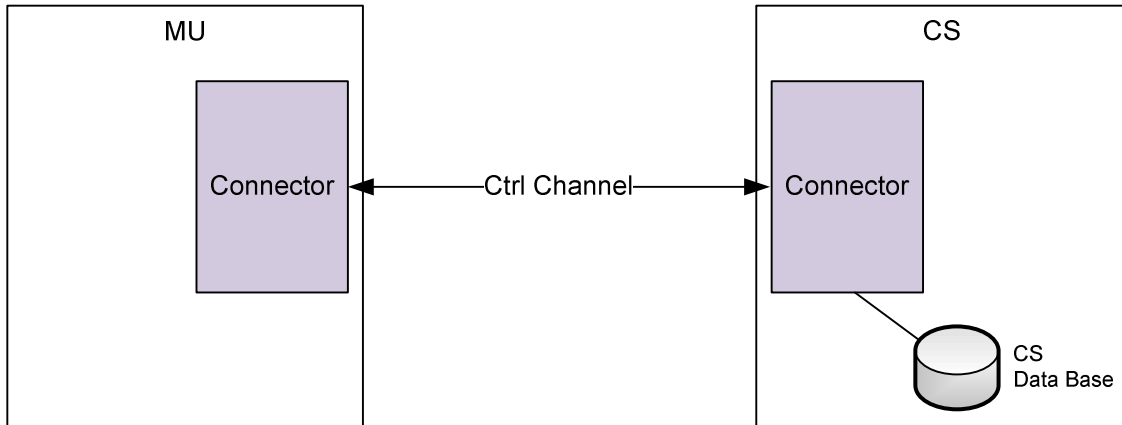
#### 5.2.2.5 Encryption

The CS may implement encryption techniques (e.g. SSL) to encrypt the traffic between CS and MU or AS.

## 5.3 Connector

This concept of connectors is based on the document [8]. The following figure shows a diagram of the connectors:

Figure 3: Connectors



The two connectors interact over the so-called “control channel”. The control channel shall be implemented with a connected TCP socket originated by the MU.

The control channel handles the following:

- MU logs on to the CS with its unique identifier
- MU logs off
- MU sends “keep alive” message to the CS with time indication when next keep alive shall be expected.
- MU responds to the messages from the CS (acknowledge or error)
- CS notifies the MU about a pending transaction
- CS responds to the messages from the CS (acknowledge or error)

The logon and logoff messages are used to connect/disconnect from the MU to the CS. With a logon the unique identifier of the MU shall be given CS. The CS shall accept or reject the logon.

The purpose of the keep alive message is on one hand to make sure the GSM network operator does not break the connection and on the other hand to have the ability to update the state (online/offline) of the MU in the CS data base. For the second case on the reception of a keep alive message the CS may start a timer (according the time indication) to wait for the next keep alive message. If no keep alive or other message arrives in time the MU has been lost.

On a notify message the HTTP client (MU) requests the pending transaction from the CS.

The response upon a request message is either an acknowledge or an error with the according error number.



The “connector concept” is introduced to fulfil requirement Req-11 (see paragraph 2.11) in the “MU always online” mode.

Systems not having this use case (e.g battery powered, autarkic mobile units) the implementation of this mechanism is not necessary.

This concept is related to IP but not to GSM-SMS technology.

### 5.3.1 Message Structure

The messages for this protocol have a very simple structure. Each message type is specified with one single ASCII character. A parameter is added as an ASCII character sequence (except “/n”).

The message shall be terminated with the end delimiter “newline” (“\n”, ASCII character 0x0a). For this protocol no coding is used. Plane text shall be transmitted as shown in the examples below.

### 5.3.2 Messages

The following messages are required for the connectors (“connector protocol”):

Table 1: Connector Protocol Messages

Message Type	Identifier	Parameter	Example
Logon	“L”	Unique Identifier of MU[ASCII without newline]	“LUnit1234/n”
Logoff	“O”	n.a.	“O/n”
Keep Alive	“K”	Time indication when next keep alive shall be expected by the CS [Seconds]	“K300/n”
Notify	“N”	n.a.	“N/n”
Acknowledge	“A”	n.a.	“A/n”
Error	“E”	Error number according the HTTP protocol (see [4]) [Integer]	“E401/n”

### 5.3.3 Mechanisms

#### 5.3.3.1 General

Either MU or CS can send a request message to the counterpart.

If the requesting message can be successfully achieved, an acknowledge (“A/n”) shall be responded.

If the message type is unknown the following error “400 Bad Request“ („E400/n“) shall be responded.

For each request the response shall be received. Several request messages must not be concatenated.

#### 5.3.3.2 Logon

The MU sends a logon request to the CS. The MU shall provide its unique identifier to the CS as the parameter (e.g. “LUnit1234/n”).

If the CS accepts the logon it shall respond with an acknowledge. If the CS rejects the logon it shall respond with the error “401 Unauthorized” (“E401/n“)

#### 5.3.3.3 Logoff

The MU sends a logoff request to the CS (e.g. “O/n”).

The CS shall respond with an acknowledge.

#### 5.3.3.4 Keep Alive

The MU sends a keep alive request to the CS. The MU shall add the parameter with the time indication when next keep alive shall be expected by the CS (e.g. “LUnit1234/n”).

If the CS receives the keep alive it shall respond with an acknowledge.

#### 5.3.3.5 Notify

The CS sends a notify request to the CS (e.g. “N/n”).

The MU shall respond with an acknowledge.

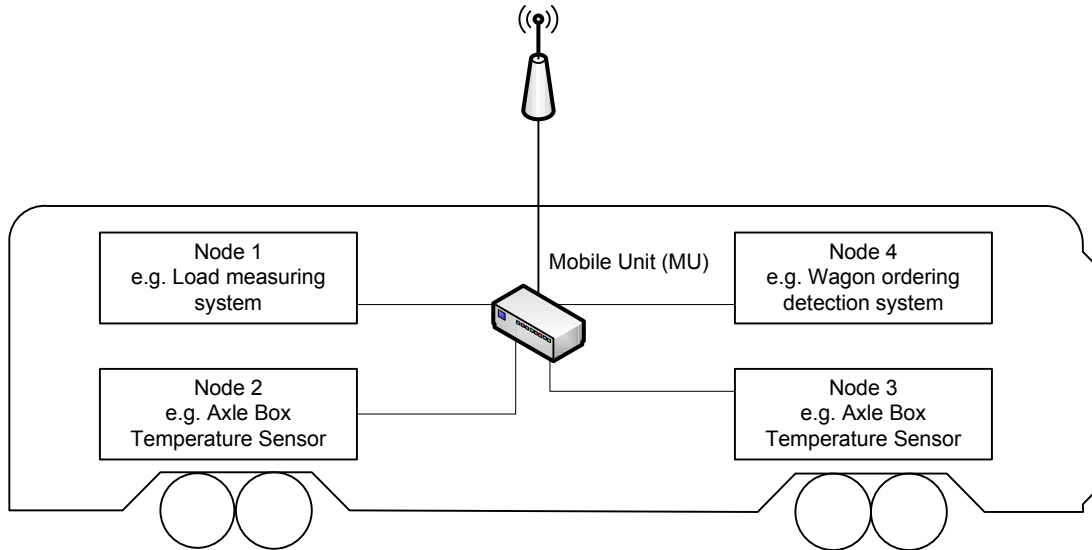
#### 5.3.3.6 Connector Sequence Diagrams

The mechanisms above are shown in the Figure 7.

## 5.4 Node

OptiSMS allows a CS to talk directly to a sensor, actor, or other external device. Therefore the protocol defines a mechanism how to address so called nodes. Depending on the application the MU may provide routing capability to access the nodes (e.g. on intelligent freight wagon, see figure below).

Figure 4: Nodes on intelligent freight wagon



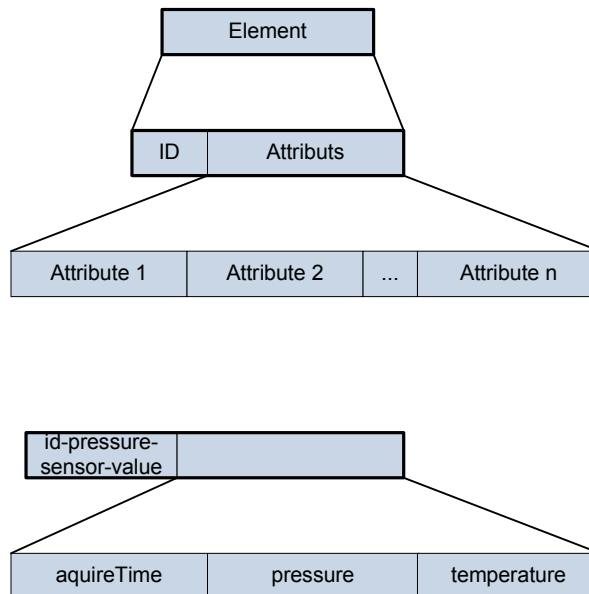
## 6 Payload

### 6.1 Elements

In optiSMS the basic data item that is transmitted is named “element”. Such elements can be exchanged between MU and CS with the transactions “GET” and “SET”. Transactions are performed with operations “Request” and “Response”.

An operation contains a sequence of elements. Each element has a unique identifier and one or several attributes. The following diagram gives a simplified view of this structure and an example:

Figure 5: The Element



```
id-pressure-sensor-value ::= INTEGER (123)
```

```
PressureSensorValue ::= SEQUENCE {
  acquireTime Timestamp, -- UTC time when the pressure is acquired
  pressure INTEGER, -- Pressure in milliBar
  temperature INTEGER -- air temperature in grad deciCelsius
}
```

The application designer is free to define the needed elements for a specific application.



#### Recommendation:

- Use of a recommended default set of elements regarded to the application field. A default set of elements has to be built for each sector of industry (or type of application) and if such a set does not exist we shall take the default set of elements which is the most appropriate, if possible.
- For application specific elements which are not part of a default set the element identifier (Number) should start at 1.000 increasing.

---

## 6.2 Attribute Structure

The application designer can define and combine the attributes with all the concepts of ASN.1 [2]. Some of them are:

- SEQUENCE (of different attributes), e.g. see paragraph 6.
- SEQUENCE OF (list of the same attribute), e.g.:

LogFile ::= SEQUENCE OF LogFileEntry
--------------------------------------

- CHOICE (alternative attribute type)
- ENUMERATION

---

## 6.3 Transactions and Operations

The following use cases can be handled with optiSMS:

- The communication server (CS) provides information to the MU (SET)
- The CS requests information from the MU (GET)
- The MU provides information to the CS (SET)
- The MU requests information from the CS (GET)

OptiSMS does not differentiate the direction of the data flow. So the use cases are reduced to two principle transactions GET and SET.

Each transaction has two parts. The first part is the request and the second part is the response. The party originating a transaction sends a request, the counterpart answers with the response.

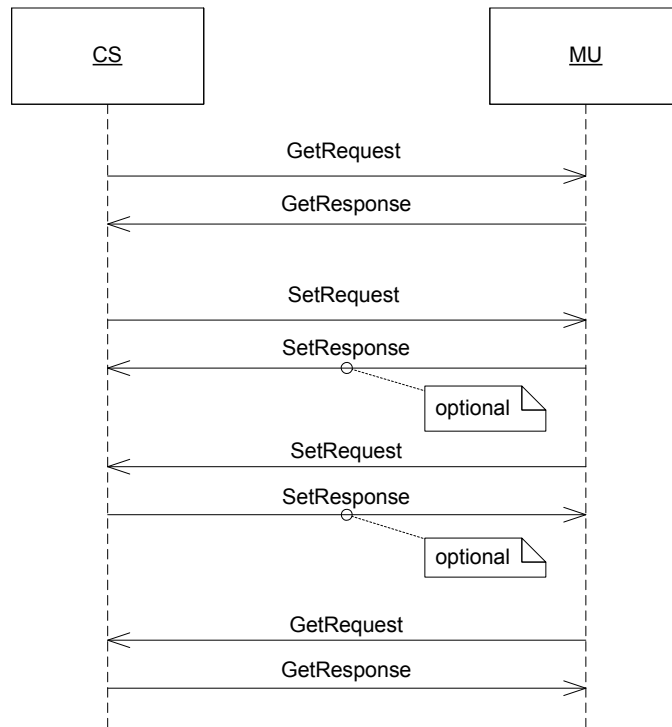
Each part of the transaction is called operation. Therefore the following operations are defined:

- GetRequest
- GetResponse
- SetRequest
- SetResponse

A GetRequest is answered with GetResponse, a SetRequest is normally answered with SetResponse. The SetResponse is optional to save data volume.

The following diagram shows all possible transactions:

Figure 6: Possible Transactions



## 6.4 Operations

This paragraph gives an overview on the operations. The detailed specification can be found in the ASN.1 specification (see paragraph 6.7).

### 6.4.1 GetRequest

The content of a GetRequest message consists in a list of either:

- Option 1: an element identifier.
- Option 2: a subset of an element identifier.

The subset of an element identifier is defined by a start index and the amount of elements in a sequence of the same element type. The purpose of the subset is to address several instances of the same element type.

### 6.4.2 GetResponse

The contents of a GetResponse consist in either:

- a general error (for all options of GetRequest in case of protocol error)
- the response to the element identifiers in a list of either:
  - GetRequest option 1: an element
  - GetRequest option 2: a sequence of elements of the same ID (subset) and a starting index.

The response contains either the requested elements and subsets or an error related to the requested element. In case of option 2 the element related error can only be indicated for the element and not for several instances of the same element.

In case of an element related error the GetResponse can be either a general error or an element specific error.



The GetResponseItem table (see paragraph 6.7) shall only contain elements that are readable. Elements that are write-only shall be omitted in this table. Caution: Some elements are writeable on the CS side but not on the MU side and vice-versa.

### 6.4.3 SetRequest

The contents of a SetRequest message consist in:

- sendResponse-Flag (tells the receiver that it has to answer the request with a response resp. an acknowledgement).
- a list of either:
  - Option 1: an element (and its attributes).
  - Option 2: a sequence of elements of the same ID (subset) and a starting index.



The SetRequestItem table (see paragraph 6.7) table only contains elements that are writeable. Elements that are read-only shall be omitted in this table. Caution: Some elements are writeable on the CS side but not on the MU side and vice-versa.

### 6.4.4 SetResponse

The contents of a SetResponse consist in either:

- an acknowledgement
- a general error (for all options of SetRequest in case of protocol error)
- a list of element related errors

An acknowledgement is a message back. The receiver notifies that the operation was fulfilled successfully.

### 6.4.5 Element related errors

Several different errors may be specified for each set element. Additional parameters may be transmitted with an element specific error. At runtime only one error can be transmitted for each set element.

The use of element specific error messages is not mandatory. The response to an element specific error may be a general error.

### 6.4.6 General error

The general error signals failure on:

- Access security policy
- General request processing
- A general error with one or several elements

Example:

```
GeneralError ::= ENUMERATED {
  no-access,      -- The receiver has refused the access due to security policy
  read-only-access, -- The receiver has refused a write access due to security policy
  inconsistent-data, -- The receiver has detected a data inconsistency
  insufficient-memory, -- The receiver has insufficient memory to respond
  busy,          -- The receiver is busy and not able to respond at the moment
  element-error, -- The receiver has detected an unspecified error on one or several elements
  ...
}
```

## 6.5 Element Naming Schema (Identifier)

The identifier shall be defined in the application specification.

It may be either an automatic enumeration or a fixed list of integer. The application designer shall assign unique identifiers.

## 6.6 Protocol Extensibility and Versioning

ASN.1 defines rules how to extend an existing protocol. Therefore an application designer may add some new elements to the application specification and create e.g. a version 2. ASN.1 decoder/encoder can handle different application specification version at run-time (e.g. client has v1 and server has v2) and manages to ignore unknown elements and overcome not existing elements without skipping the whole data.

The extensibility is limited to extend the protocol at marked places in the structures.

Example:

```
Waypoint ::= SEQUENCE {
  locationCode    UINT16, -- Location code TBDF (station code, coordinate, ect.)
  eta             Timestamp, -- Estimated time (UTC) of arrival
  actionCodeOnTime UINT16, -- Action code to perform if arrival in time
  ...
  [[2: actionCodeLate UINT16 ]], -- Action code if arrival to late [added with version 2]
  [[4: state        UINT16 ]], -- State code of waypoint [added with version 4]
}
```

Value constraints may be extended if destined in initial version.

Example:

```
LoadWeight-Attributes ::= INTEGER (0..65535, ..., 0..200000) -- Unit: kg (0 .. 65.535t, extended to 200t with version 3)
```



It is not possible to change the type of attributes from one to the next version. Therefore application designer must consider extension from the beginning.



OptiSMS distinguishes between application specification extensions as described above and incompatible optiSMS protocol versions. Incompatibility means that the decoder is not able to process the input data at all (e.g. introduction of a new operation). In a transmission optiSMS provides this protocol version in the presentation layer (see 7.4). The receiver of an operation must refuse it if he does not support that version.

## 6.7 OptiSMS Specification Version 1 according ASN.1

```

OptiSmsProtocolModule DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
IMPORTS ElementId, GetResponseItem, SetRequestItem, SetResponseItem FROM OptiSmsContentModule;

-- Operation --
-- The following operations are supported:
-- - GetRequest and GetResponse
-- - SetRequest and SetResponse
-- All operations are symmetric and can be performed in
-- both directions (Communication Server (CS)/ Mobile Unit (MU))
--
-- Possible transactions:
-- Server      Mobile
-- 1a |====GetRequest====>|
-- 1b |<====GetResponse====|
-- |
-- 2a |====SetRequest====>|
-- 2b |<====SetResponse====| (optional)
-- |
-- 3a |<====SetRequest====|
-- 3b |====SetResponse====>| (optional)
-- |
-- 4a |<====GetRequest====|
-- 4b |====GetResponse====>|
Operation ::= CHOICE {
  getRequest  GetRequest,
  getResponse GetResponse,
  setRequest  SetRequest,
  setResponse SetResponse,
  ...
}

-- GetRequest --
-- CS (resp. MU) requests to read a list of elements from MU (resp. CS)"
GetRequest ::= SEQUENCE OF CHOICE {
  single ElementId,
  subset GetRequestSubsetItems
}

GetRequestSubsetItems ::= SEQUENCE {
  startIndex INTEGER (0..MAX) DEFAULT 0,
  amount    INTEGER (1..MAX) DEFAULT 1,
  items     ElementId
}

-- GetResponse --
-- The GetResponse provides the requested elements defined in the GetRequest.
GetResponse ::= CHOICE {
  generalError GeneralError,
  responseList SEQUENCE OF CHOICE {
    single GetResponseItem,
    subset GetResponseSubsetItems
  }
}

GetResponseSubsetItems ::= SEQUENCE {
  startIndex INTEGER (0..MAX) DEFAULT 0,
  items      SEQUENCE OF GetResponseItem
}

-- SetRequest --
-- CS (resp. MU) requests to write a list of elements to the MU (resp. CS)"
SetRequest ::= SEQUENCE {
  sendResponse BOOLEAN DEFAULT TRUE, -- The receiver must send a response
  requestList  SEQUENCE OF CHOICE {
    single SetRequestItem,
    subset SetRequestSubsetItems
  }
}

SetRequestSubsetItems ::= SEQUENCE {
  startIndex INTEGER (0..MAX) DEFAULT 0,
  items      SEQUENCE OF SetRequestItem
}

```

```

-- SetResponse --
-- The SetResponse acknowledges the SetRequest.
SetResponse ::= CHOICE {
  successful  NULL,           -- The SetRequest performed successfully
  generalError  GeneralError, -- A general error happened
  responseList SEQUENCE OF SetResponseItem -- An element error happened -> list of failed IDs with its error code
}

-- GeneralError --
-- On an operation a general error may occur. This indicates the failure on:
-- * Access security policy
-- * General request processing
-- * Unspecified error on one or several elements
GeneralError ::= ENUMERATED {
  no-access,           -- The receiver has refused the access due to security policy
  read-only-access,   -- The receiver has refused a write access due to security policy
  inconsistent-data,  -- The receiver has detected a data inconsistency
  insufficient-memory, -- The receiver has insufficient memory to respond
  busy,              -- The receiver is busy and not able to respond at the moment
  element-error,     -- The receiver has detected an unspecified error on one or several el.
  version-error,     -- The receiver has detected version error and was not able to decode
  coding-error,      -- The receiver has detected an unknown coding type
  ...
}
END

```

## 7 optiSMS Layer Stack

### 7.1 General

The optiSMS protocol covers several layers of the OSI “Seven Layer” Model [1]. Depending on the transmission technology more or less layers of the OSI model are affected. The following table gives an overview:

Table 2: OSI Layer Stack

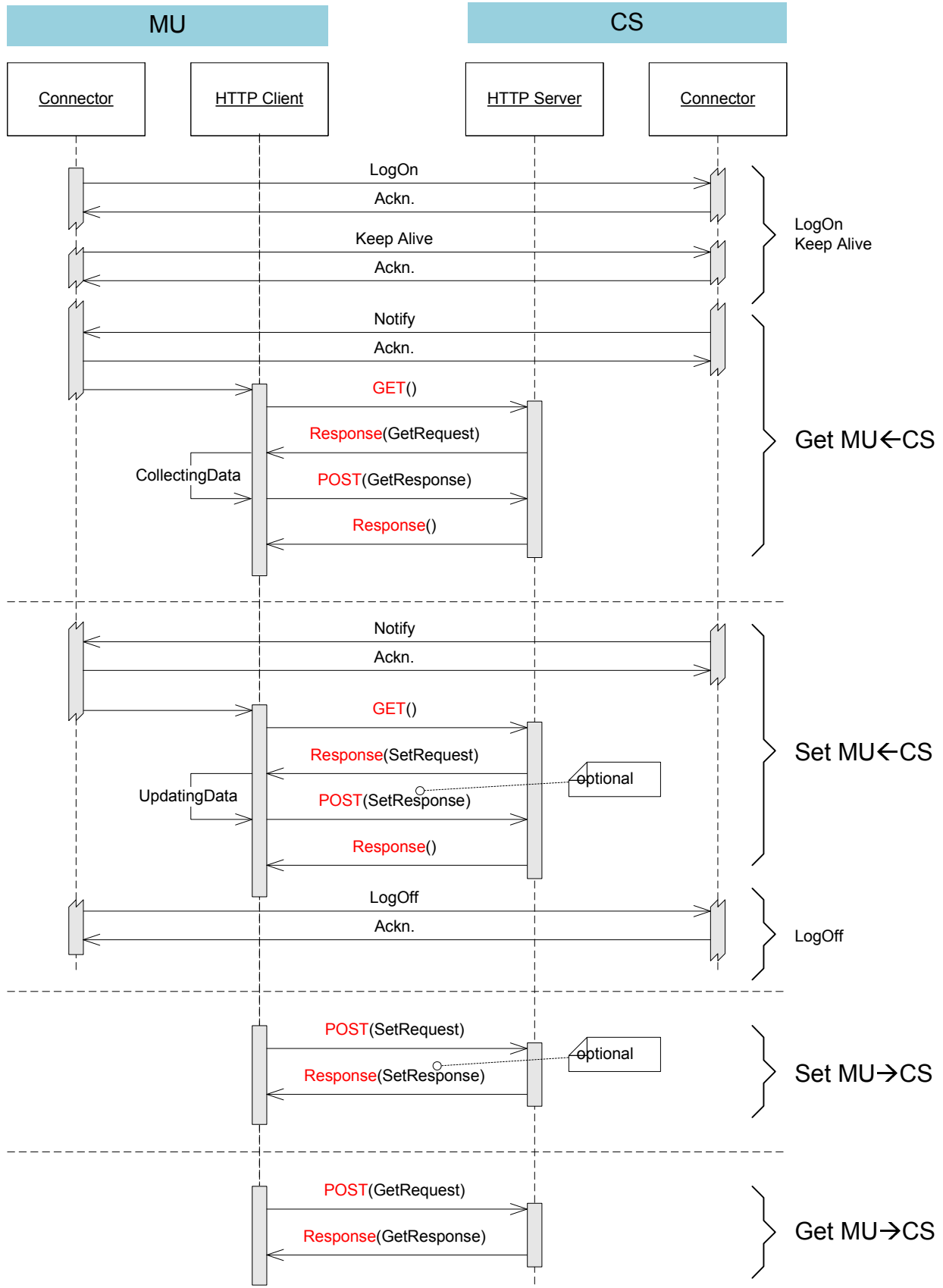
Layer	IP Technology	GSM-SMS Technology
<a href="#">7. Application</a>	Operations (Elements, Errors)	
<a href="#">6. Presentation</a>	Encoding and Decoding	
	<ul style="list-style-type: none"> <li>• Coding type</li> <li>• Protocol version</li> </ul>	<ul style="list-style-type: none"> <li>• Coding type</li> <li>• Protocol version</li> </ul>
<a href="#">5. Session</a>	<ul style="list-style-type: none"> <li>• Node identifier<sup>1)</sup></li> <li>• Source address<sup>2)</sup></li> <li>• Transaction pending</li> <li>• Length</li> </ul>	<ul style="list-style-type: none"> <li>• Node identifier<sup>1)</sup></li> <li>• Source Address<sup>2)</sup></li> <li>• Transaction control (Transaction reference number)</li> </ul>
	HTTP POST/RESPONSE	
<a href="#">4. Transport</a>	TCP	Packer (Optional) CRC
<a href="#">3. Network</a>	GPRS	SMS
2. Data Link	GSM	GSM
1. Physical		

- 1) MU→CS request: Node ID indicates the source, CS→MU request: Node ID indicates the destination.
- 2) The source address indicates the address of either the MU or the CS. Originated from the MU the source address holds the MU's unique identifier (with IP) or the MU's phone number (with GSM-SMS). In the other direction the source address holds the ULR of the CS (with IP) or the phone number of the CS (with GSM-SMS).
- 3) Fields marked in green relate to optiSMS.

## 7.2 HTTP Sequence Diagram

The following diagram gives an overview of how the embedded operations are transmitted with the HTTP protocol.

Figure 7: HTTP Sequence Diagram



1) Text in red marks the HTTP methods. The text inside the brackets

---

### 7.3 Application Layer

ASN.1 is used to specify the application content (see appendix 8.1).

---

### 7.4 Presentation Layer

#### 7.4.1 Description

The presentation layer handles the encoding/decoding of the application content. Therefore the receiver of a request or response must know about the coding type and incompatible protocol version. These two information are part of the presentation layer.

- The version information is mandatory.
- The coding information is optional.

The ASN.1 is used to specify the coding. The following coding rule shall be supported according [2]:

- UPER (unaligned packed encoding rule) (best packing performance)

The following coding rules are optionally for diagnostic purpose:

- PER (packed encoding rule)
- BER (basic encoding rule)
- XER (XML encoding rule)

#### 7.4.2 Version and coding with IP technology

The protocol version shall be added as a parameter to the Request-URI (POST) or to the “Location” tag (HTTP response).

This version parameter is mandatory for the request.

The parameter shall be named “ver”.

The value shall be an integer between 0 and 31 according the supported protocol version.

Example:

`/index/?ver=1`

The coding type shall be added in the POST method as a parameter to the HTTP request-URI [4].

The parameter shall be named “code”.

Following values of the parameter shall be supported:

- “uper”
- “per”
- “ber”
- “xer”

If this parameter is not added to the Request-URI the receiver shall assume the coding is UPER (default).

Example:

`POST /index/?source=unit1234&node=123&ver=2&code=per HTTP/1.1\r\n`

In the HTTP response message the coding parameter shall be added to the “Location” tag.

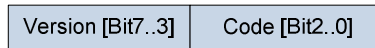
Example:

HTTP/1.1 200 OK\r\n  
Location: /index/?ver=2&code=per

### 7.4.3 Version and coding with GSM-SMS technology

The header has the following structure:

Figure 8: GSM-SMS Presentation Layer Header



The version is packed in Bit7..3 of the header. Bit7 is the most significant bit. The value shall be an integer between 0 and 31 according to the supported protocol version.

The coding is packed in Bit2..0 of the header. Bit2 is the most significant bit. The following decimal representation shall be used:

Table 3: GSM-SMS Coding Representation

Encoding/Decoding Rule	Decimal Representation
<b>UPER</b>	0x0
<b>PER</b>	0x1
<b>BER</b>	0x2
<b>XER</b>	0x3
<b>Reserved for future use</b>	0x4..0x6
<b>Undefined</b>	0x7

If no coding needs to be transmitted the value shall be set to “Undefined”

### 7.4.4 Error Handling

The receiver may encounter the following problems on this layer:

- Unknown protocol version
- Unknown coding type

With IP technology the receiver shall decode with best effort with its default values for version and coding. If the decoder fails, the application layer shall respond with a general error.

With GSM-SMS technology no response on the receiver side shall be generated. The decoder will try with best effort with its default values for version and coding.

---

## 7.5 Session Layer

### 7.5.1 Description

With optiSMS the session layer handles the connection and transaction control. Thus the node identifier and the source address shall be added to the content of the upper layer. Depending on the transmission technology the transaction pending flag for IP and a transaction reference number shall be added (GSM-SMS).

#### 7.5.1.1 Node identifier

In this layer the node identifier may be transmitted. This identifier is optional, but shall only be used in the request message. The response message implicitly is coming from or going to the referenced node.

Depending on the transaction direction the meaning of the node identifier is different.

- MU→CS Get-/SetRequest: Node ID indicates the source
- CS→MU Get-/SetRequest: Node ID indicates the destination

#### 7.5.1.2 Source Address

In this layer the identification of the MU shall be transmitted. Therefore the source identifier is used.

The source information is mandatory on both directions and on both technologies.

Originated from the MU the source address holds the MU's unique identifier (with IP) or the MU's phone number (with GSM-SMS). In the other direction the source address holds the URL of the CS (with IP) or the phone number of the CS (with GSM-SMS).

### 7.5.2 IP Technology

The node identifier shall be added as a parameter to the Request-URI (POST) or to the "Location" tag (HTTP response).

The parameter shall be named "node".

The value shall be an integer between 0 and 255 according to the system address schema.

Example:

`/index/?node=23`

The source address shall be added as a parameter to the Request-URI (POST) or to the "Location" tag (HTTP response).

The parameter shall be named "source".

The value shall be a sequence of characters (URL/unique identifier) according to URL convention.

Example:

(MU→CS): `/index/?source=unit1234`

(CS→MU): `/index/?source=cs.optisms.interrail.com`

### 7.5.2.1 Transaction Pending Flag

With IP technology the CS may add a “Transaction Pending Flag” to each HTTP response. This indicates the MU pending transactions on the CS. In this case the MU may start a new HTTP GET to initiate the pending transaction.

The flag shall be added as a parameter to the “Location” tag (HTTP response).

The parameter shall be named “pending”.

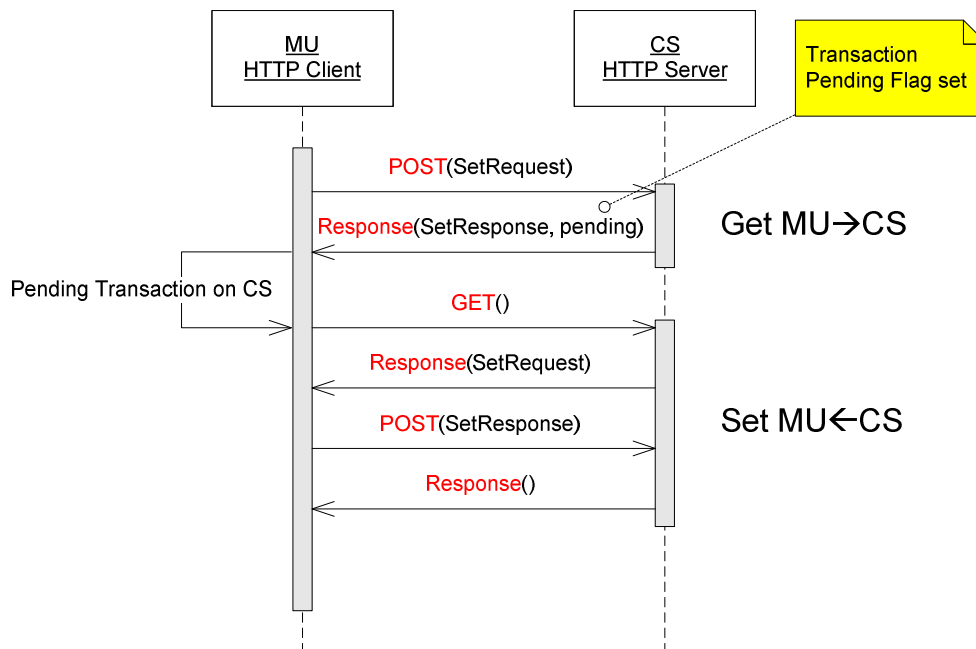
The value shall be empty.

Example: /index/?**pending**

With GSM-SMS technology this flag is not foreseen.

Example:

Figure 9: Example Sequence Diagram - Transaction Pending Flag



This example shows a sequence diagram of a MU signalling an event to the CS. In the HTTP response the CS informs the MU about a pending transaction. Thus the MU initiates a transaction with the HTTP GET method. The CS is now able to transmit the SetRequest.

### 7.5.2.2 HTTP

As session protocol optiSMS uses HTTP for IP technology.

HTTP version 1.1 and further shall be supported.

The HTTP client on the MU shall not close the session (IP socket) until the complete transaction has been performed. The MU shall respond to a Get-/SetRequest on the same session. This allows the CS to manage transaction control for CS originated transactions. Thus the HTTP client must not use the HTTP header field “Connection: close” in-between a transaction.

The length of the HTTP body must be indicated.

The length shall be added in the Content-Length header field.

The value shall be the length of the HTTP response/POST body in bytes, or in other words the length of the ASN.1 encoded application content.

Example: “Content-Length: 348”

### 7.5.3 GSM-SMS Technology

The GSM-SMS header of this layer has the following structure:

Figure 10: GSM-SMS Session Layer Header

Byte 0	Byte 1
Node ID [Bit7..0]	Transaction Ref. No. [Bit7..0]

The node identifier is packed in byte 0 of the header. Bit7 is the most significant bit.

The transaction reference number (see paragraph below) is packed in byte 1 of the header. Bit7 is the most significant bit.

Byte 0 shall be send first.

#### 7.5.3.1 Transaction Control

With GSM-SMS technology the transaction (request→response) is not managed by the transport technology (in contrary to HTTP). Per default the response GSM-SMS cannot be assigned to the request GSM-SMS.

Thus optiSMS introduces the so called “Transaction Reference Number”. With each request GSM-SMS the reference number shall be provided. The receiver of the request shall copy the reference number to the response. With each request the reference number shall be changed to another value.

This allows the receiver of the response to perform transaction control checks. If the originator of an request does not receive the expected response in time, this layer shall inform the upper layer about the timeout.

The transaction reference number shall be added in the GSM-SMS session layer header.

#### 7.5.4 Error Handling

The receiver in general may encounter the following problems on this layer:

- Unknown node
- Unreachable node
- Not recognized MU
- Unknown transaction number (GSM-SMS) (receiver internal)
- Wrong length (receiver internal)
- CS not reachable
- Response Timeout (no response in expected time)

The receiver of a request shall respond with a general error (IP) or must not respond at all (GSM-SMS).

## 7.6 Transport Layer

### 7.6.1 Description

This layer handles the transport of the content of the upper layer.

### 7.6.2 IP Technology

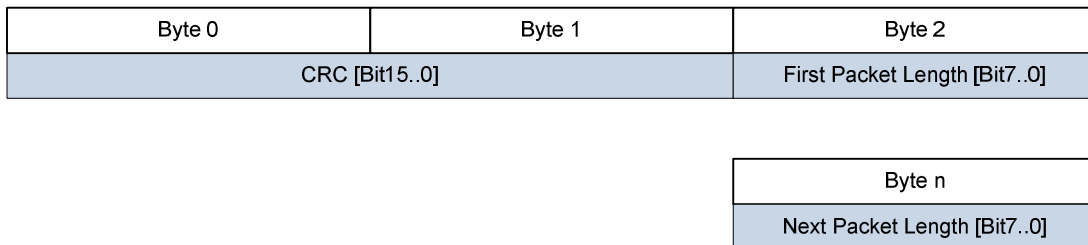
Standard TCP shall be used. In most implementations the HTTP client/server already sets up on TCP.

### 7.6.3 GSM-SMS Technology

With GSM-SMS technology the transport layer handles an optional packing mechanism and an additional CRC mechanism.

The header has the following structure:

Figure 11: GSM-SMS Transport Layer Header



The CRC is packed in byte 0 and 1 of the header. Byte 0 is the most significant byte and shall be send first.

The “First Packet Length” information is packed in byte 2 of the header. Bit7 is the most significant bit. If no packing is performed this information shall be set to zero. If two or more packets are transmitted in the frame, the “Next Packet Length” of the next package shall be added in-between the previous and the following packet.

#### 7.6.3.1 Packer

This optional mechanism collects frames directed to the same mobile number to one collection of frames. This reduces the amount of GSM-SMS.

This mechanism is optional. Detailed specification is not yet defined. Although the packet information in the header shall be foreseen.

#### 7.6.3.2 CRC

A GSM-SMS specific CRC is added to the content of the upper layer. The 2 byte CRC is based on the CRC-16-CCITT polynomial definition:

$$x^{16} + x^{12} + x^5 + 1, \text{ with pre-load value} = 0$$

The CRC shall be calculated over the whole content of the upper layer (including the packet information).

### 7.6.4 Error Handling

The receiver may encounter the following problems on this layer:

- Wrong CRC (GSM-SMS)

If the CRC on the receiver side fails, the receiver shall discard the received GSM-SMS and may signal a CRC error to the upper layer.

## 7.7 Network Layer

### 7.7.1 GSM-SMS technology

The following paragraph specifies the GSM-SMS message.

- 1) The GSM-SMS shall be sent and received in PDU mode.
- 2) The type-of-address of SMSC and sender number shall be the international format (0x91).
- 3) The TP-Protocol-Identifier shall be set to default (0x00).
- 4) The TP-Data-Coding-Schema shall be set to 8bit data, Class 2 (0x06).

## 7.8 Examples

### 7.8.1 IP Example SetRequest→SetGetResponse CS→MU→CS

Destination address of CS: <http://optisms.interrail.com>

- 1) GET /index HTTP/1.1\r\n
- 2) HTTP/1.1 201 Created\r\n
 Location: /index/?ver=2&code=per&source=optisms.interrail.com  
 Content-length: 73\r\n
 \r\n
 490385jkdsföklajkföklajop978234öjlsadfökljöklsdjaflokfj3489075pijasdfökla
- 3) POST /index HTTP/1.1\r\n
 Content-length: 12\r\n
 \r\n
 490385jkdsfö
- 4) HTTP/1.1 200 OK\r\n

### 7.8.2 IP Example SetRequest→SetResponse MU→CS→MU

Destination address of CS: <http://optisms.interrail.com>

- 1) POST /index/?ver=2&code=per&source=unit1234 HTTP/1.1\r\n
 Content-length: 73\r\n
 \r\n
 490385jkdsföklajkföklajop978234öjlsadfökljöklsdjaflokfj3489075pijasdfökla
- 2) HTTP/1.1 201 Created\r\n
 Content-length: 12\r\n
 \r\n
 490385jkdsfö

### 7.8.3 GSM-SMS Example SetRequest→SetResponse MU→CS→MU

The following example shows a Set transaction with GSM-SMS technology.

- 1) GSM-SMS PDU:
  - a. Dest: +39123456789 (CS)
  - b. Length: 33
  - c. Data:
    - i. Pres.: 0x11 (Version 2, PER)
    - ii. Ses.: 0x00, 0xa5 (Node 0, Transaction Ref. 165)
    - iii. Trsp: 0xa4f1, 0x00 (CRC = 42225, First Packet Length = 0)
    - iv. Appl.: 0x123a45d67889f0988aa765454f9999e88812345678  
 → 0x1100a5a4f100123a45d67889f0988aa765454f9999e88812345678
- 2) GSM-SMS PDU:
  - a. Dest: +39456789321 (MU)
  - b. Length: 13
  - c. Data:
    - i. Pres.: 0x11 (Version 2, PER)
    - ii. Ses.: 0x00, 0xa5 (Node 0, Transaction Ref. 165)
    - iii. Trsp: 0xa4f1, 0x00 (CRC = 42225, First Packet Length = 0)
    - iv. Appl.: 0x123a45d6  
 → 0x1100a5a4f100123a45d6

### 8.1 Application Content Example

The content of the optiSMS protocol is application specific.

The following example of an ASN.1 shows some possibilities:

```

OptiSmsContentModule DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

-- All possible elements or element errors that can be transmitted with a GetRequest
-- Remark: Write-only elements can be excluded in this choice
GetResponseItem ::= CHOICE {
  unknown          [id-unknown]          UnknownElement-AttributesOrError,
  loadWeight       [id-load-weight]      LoadWeight-AttributesOrError,
  position         [id-position]         Position-AttributesOrError,
  positionExtended [id-position-extended] PositionExtended-AttributesOrError,
  serverLogonInterval [id-server-logon-interval] ServerLogonInterval-AttributesOrError,
  -- write-only event [id-event]         Event-AttributesOrError,
  waypointByIndex [id-waypoint-by-index] WaypointByIndex-AttributesOrError,
  alarmBehaviour  [id-alarm-behaviour]   AlarmBehaviour-AttributesOrError,
  -- write-only defaultAlarmInfo [id-default-alarm-info] DefaultAlarmInfo-AttributesOrError,
  privatContainer [id-private-container] PrivatContainer-AttributesOrError,
  ... -- Elements of a new version must be placed here!
}

-- All possible elements that can be transmitted with a SetRequest
-- Remark: Read-only elements can be excluded in this choice
SetRequestItem ::= CHOICE {
  -- read-only loadWeight [id-load-weight] LoadWeight-Attributes,
  position [id-position] Position-Attributes,
  positionExtended [id-position-extended] PositionExtended-Attributes,
  serverLogonInterval [id-server-logon-interval] ServerLogonInterval-Attributes,
  event [id-event] Event-Attributes,
  waypointByIndex [id-waypoint-by-index] WaypointByIndex-Attributes,
  alarmBehaviour [id-alarm-behaviour] AlarmBehaviour-Attributes,
  defaultAlarmInfo [id-default-alarm-info] DefaultAlarmInfo-Attributes,
  privatContainer [id-private-container] PrivatContainer-Attributes,
  ... -- Elements of a new version must be placed here!
}

-- All possible element errors that can be transmitted with a SetResponse
SetResponseItem ::= CHOICE {
  unknown [id-unknown] UnknownElement-Error,
  loadWeight [id-load-weight] LoadWeight-Error,
  position [id-position] Position-Error,
  positionExtended [id-position-extended] PositionExtended-Error,
  serverLogonInterval [id-server-logon-interval] ServerLogonInterval-Error,
  event [id-event] Event-Error,
  waypointByIndex [id-waypoint-by-index] WaypointByIndex-Error,
  alarmBehaviour [id-alarm-behaviour] AlarmBehaviour-Error,
  defaultAlarmInfo [id-default-alarm-info] DefaultAlarmInfo-Error,
  privatContainer [id-private-container] PrivatContainer-Error,
  ... -- Elements of a new version must be placed here!
}

-- Element Identifier --
-- The ElementId is a unique identifier of an element
-- This enumeration defines the value of the identifier
-- This values are used in the DefinedValue definition below
ElementId ::= ENUMERATED {
  id-value-unknown-element (0),
  id-value-load-weight (1),
  id-value-position (2),
  id-value-position-extended (3),
  id-value-server-logon-interval (4),
  id-value-event (5),
  id-value-waypoint-by-index (6),
  id-value-alarm-behaviour (7),
  id-value-default-alarm-info (8),
  id-value-private-container (9),
  ... -- IDs for a new version must be placed here!
}

```

```

-- Element Identifier DefinedValue --
-- The tags in GetResponseItem, SetRequestItem and SetResponseItem
-- require a "DefineValue" (according ASN.1). The use of the value
-- out of ElementId is not possible.
-- Therefore the following list defines "DefineValue" of the IDs.
id-unknown          ElementId ::= id-value-unknown-element
id-load-weight      ElementId ::= id-value-load-weight
id-position         ElementId ::= id-value-position
id-position-extended ElementId ::= id-value-position-extended
id-server-logon-interval ElementId ::= id-value-server-logon-interval
id-event           ElementId ::= id-value-event
id-waypoint-by-index ElementId ::= id-value-waypoint-by-index
id-alarm-behaviour ElementId ::= id-value-alarm-behaviour
id-default-alarm-info ElementId ::= id-value-default-alarm-info
id-private-container ElementId ::= id-value-private-container

-- Element ServerLogonInterval --
-- Description: Defines the interval how often the MU shall connect the CS
ServerLogonInterval-Attributes ::= UINT16 (10..MAX) -- Interval from 10 seconds to ~18 days

ServerLogonInterval-Error ::= CHOICE {
  outOfRange PossibleRange,
  ...
}

PossibleRange ::= SEQUENCE {min UINT16, max UINT16}

ServerLogonInterval-AttributesOrError ::= CHOICE {
  attributes ServerLogonInterval-Attributes,
  error ServerLogonInterval-Error
}

-- Element Event --
-- Description: Informs the CS about asynchronous events on the MU
Event-Attributes ::= SEQUENCE {
  position Position-Attributes,
  eventTime Timestamp, -- UTC time when the event occurs
  eventCode UINT16 -- Event code (TBDF Enumeration!)
}

Event-Error ::= CHOICE {
  notSupported NULL, -- E.g. on the MU a SET of this element is not supported
  ...
}

-- Event element is write-only, therefore no Event-AttributesOrError required
-- Event-AttributesOrError ::= CHOICE {
--   attributes Event-Attributes,
--   error Event-Error
-- }

-- Element WaypointByIndex --
-- Description: Addressing a waypoint via index out of a list
-- Specific error: invalid-index TBDF
WaypointByIndex-Attributes ::= SEQUENCE {
  waypointIndex UINT16, -- Index to address a waypoint out of a list of waypoints
  waypoint Waypoint
}

WaypointByIndex-Error ::= CHOICE {
  readOnly NULL,
  invalidIndex IndexRange,
  ...
}
IndexRange ::= SEQUENCE {min INTEGER, max INTEGER}

WaypointByIndex-AttributesOrError ::= CHOICE {
  attributes WaypointByIndex-Attributes,
  error WaypointByIndex-Error
}

-- Attribute Waypoint --
-- Description: Defines the attributes of a waypoint.
Waypoint ::= SEQUENCE {
  locationCode UINT16, -- Location code TBDF (station code, coordinate, ect.)
  eta Timestamp, -- Estimated time (UTC) of arrival
  actionCodeOnTime UINT16, -- Action code to perform if arrival in time
  actionCodeLate UINT16, -- Action code to perform if arrival to late
}

```

```

}
...
}
-- Attribute GeoCoordinate --
-- Description: Defines the coordinate of a geographic point in the WGS84.
GeoCoordinate ::= SEQUENCE {
  -- Latitude of a geographic point in micro-degrees, northward treated as positive
  latitude INTEGER (-90000000..90000000) DEFAULT 0,
  -- Longitude of a geographic point in micro-degrees, eastward treated as positive
  longitude INTEGER (-179999999..180000000) DEFAULT 0
}

-- Element Position --
-- Description: This is a generic position type containing the acquisition time
-- and the coordinates
Position-Attributes ::= SEQUENCE {
  acquireTime Timestamp, -- UTC time when the position is acquired
  coordinate GeoCoordinate
}

Position-Error ::= CHOICE {
  readOnly NULL,
  ...
}

Position-AttributesOrError ::= CHOICE {
  attributes Position-Attributes,
  error Position-Error
}

-- Element PositionExtended --
-- Description: Extends the position element with altitude and heading
PositionExtended-Attributes ::= SEQUENCE {
  position Position-Attributes,
  altitude INT16, -- Altitude in meter
  heading INTEGER (0..359), -- Heading in degree, 0 = North, increasing clockwise
  ... -- Extensible for future version
}

PositionExtended-Error ::= CHOICE {
  readOnly NULL,
  ...
}

PositionExtended-AttributesOrError ::= CHOICE {
  attributes PositionExtended-Attributes,
  error PositionExtended-Error
}

-- Element AlarmBehaviour --
-- Description: Defines the actions that must be performed if an alarm occurs.
AlarmBehaviour-Attributes ::= SEQUENCE {
  -- Send the alarm information immediately or store it for later transmission
  sendImmediately BOOLEAN DEFAULT TRUE,
  -- Provide the default and/or the user defined elements
  informationType ENUMERATED {default, user-defined, both } DEFAULT default,
  -- List of user defined element IDs
  userDefinedElements PublishList
}

AlarmBehaviour-Error ::= CHOICE {
  unknownEntry PublishListUnknownEntries,
  ...
}

AlarmBehaviour-AttributesOrError ::= CHOICE {
  attributes AlarmBehaviour-Attributes,
  error AlarmBehaviour-Error
}

-- Attribute PublishList --
-- Description: A PublishList defines a sequence of elements that must be sent to a predefined destination
-- under certain conditions.
-- If an element contains an attribute of this PublishList it should define a specific error handling the
-- problem that an entry of the list can be an unknown ID. Therefore the PublishListUnknownEntries is defined.
PublishList ::= SEQUENCE OF ElementId
PublishListUnknownEntries ::= SEQUENCE OF INTEGER -- Error parameter to give a list of unknown identifiers

-- Element DefaultAlarmInfo --

```

```

-- Description: Defines the alarm information that shall be provided per default.
DefaultAlarmInfo-Attributes ::= SEQUENCE {
  alarmType UINT16, -- Type of alarm TBDF as enumeration or bitfield
  alarmTime Timestamp -- UTC time when the alarm has happened
}

DefaultAlarmInfo-Error ::= CHOICE {
  notSupported NULL, -- E.g. on the MU a SET of this element is not supported
  ...
}

-- DefaultAlarmInfo element is write-only, therefore no DefaultAlarmInfo-AttributesOrError required
-- DefaultAlarmInfo-AttributesOrError ::= CHOICE {
--   attributes DefaultAlarmInfo-Attributes,
--   error DefaultAlarmInfo-Error
--}

-- Element LoadWeight --
-- Description: Weight of the load
LoadWeight-Attributes ::= UINT16 -- Unit: kg (0 .. 65.535t) Is enough? TBDF

LoadWeight-Error ::= CHOICE {
  readOnly NULL,
  ...
}

LoadWeight-AttributesOrError ::= CHOICE {
  attributes LoadWeight-Attributes,
  error LoadWeight-Error
}

-- Element PrivatContainer --
-- Description: Contains vendor specific privat data
PrivatContainer-Attributes ::= OCTET STRING -- Octet sequence

PrivatContainer-Error ::= CHOICE {
  readOnly NULL,
  ...
}

PrivatContainer-AttributesOrError ::= CHOICE {
  attributes PrivatContainer-Attributes,
  error PrivatContainer-Error
}

-- =====
-- == Add new elements here ==
-- =====

-- Element "UnknownElement" --
-- Caution: This element has a special relevance in optiSMS!
-- Description: The element-specific errors specify errors that are strongly linked to a
-- well known element.
-- If the identifier of an element is not known on the receiver side this special
-- element is used to tell the sender what element ID is refused by the receiver.
-- The "UnknownElement" by itself is well known, has no attributes but the content
-- of the error holds the unknown identifier as an integer.
-- It is not required to transmit this element on SetRequest. Therefore no type
-- UnknownElement-Attributes is defined and is not added to SetRequestItem
UnknownElement-Error ::= CHOICE {
  unknownID INTEGER, -- Identifier of the unknown element
  ...
}

UnknownElement-AttributesOrError ::= CHOICE {
  attributes NULL,
  error Event-Error
}

-- Useful data types for attributes --
UINT32 ::= INTEGER (0..4294967295) -- Integer with limited range (encoded size may vary from 32 bits!)
INT16 ::= INTEGER (-32768..32767) -- Integer with limited range (encoded size may vary from 16 bits!)
UINT16 ::= INTEGER (0..65535) -- Integer with limited range (encoded size may vary from 16 bits!)
Timestamp ::= UINT32 -- Timestamp in seconds starting at 1970-1-1 00:00:00
-- add other useful types here

```

END

## 8.2 Complete Examples

### 8.2.1 CS→MU SetRequest/SetResponse with IP

→ SetRequest

#### Application Layer

Operation	setRequest	Operation	581200000033FFFFFFFFC00B400B480000
setRequest		SetRequest	C09000000019FFFFFFFFE005A005A400000
sendResponse	TRUE		80
requestList	2		024000000067FFFFFFFF80168016900000
SetRequestItem 1	waypointByIndex	SetRequestItem	4000000067FFFFFFFF801680168
waypointByIndex		WaypointByIndex-Attributes	000000067FFFFFFFF8016801680
waypointIndex	0	UINT16	0000
waypoint		Waypoint	00067FFFFFFFF8016801680
locationCode	12	UINT16	000C
eta	4294967295	Timestamp	FFFFFFFF
actionCodeOnTime	45	UINT16	002D
actionCodeLate	45	UINT16	002D
SetRequestItem 2	serverLogonInterval	SetRequestItem	200000
serverLogonInterval	10	ServerLogonInterval-Attributes	0000

#### Encoding in UPER

Hex (16Bytes): 581200000033FFFFFFFFC00B400B480000

#### HTTP

- 1) GET /index HTTP/1.1\r\n
- 2) HTTP/1.1 201 Created\r\n
 Location: /index/?ver=2&code=uper&source=optisms.interrail.com  
 Content-length: 16\r\n
 \r\n
 <binary data 16 bytes>
- 3) POST /index HTTP/1.1\r\n
 Content-length: 1\r\n
 \r\n
 <binary data 1 bytes>
- 4) HTTP/1.1 200 OK\r\n

#### Decoding in UPER:

Hex (1 Byte): 60

#### Application Layer

Operation	setResponse	Operation	60
setResponse	successful	SetResponse	00

→ SetResponse

