

**TSG-RAN Working Group 2 (Radio layer 2 and Radio layer 3)  
Sophia Antipolis 16<sup>th</sup> to 20<sup>th</sup> August 1999**

*TSGR2#6(99)715*

**Agenda Item:** 4.2

**Source:** Editor

**Title:** 3GPP TR 25.921 1.1.0 Guidelines and principles for protocol description and error handling

**Document for:** Decision

**3rd Generation Partnership Project;  
Technical Specification Group Radio Access Network;  
Guidelines and Principles for protocol description and  
error handling  
(3G TR 25.921 version 1.10.0)**

---



Reference

---

DTS/TSGR-0325921 U

Keywords

---

<keyword[, keyword]>

**3GPP**

Postal address

---

3GPP support office address

---

650 Route des Lucioles - Sophia Antipolis  
Valbonne - FRANCE  
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet

---

<http://www.3gpp.org>

---

# Contents

<b><u>1 SCOPE .....</u></b>	<b><u>10</u></b>
<b><u>2 REFERENCES.....</u></b>	<b><u>10</u></b>
<b><u>3 DEFINITIONS, SYMBOLS AND ABBREVIATIONS .....</u></b>	<b><u>10</u></b>
3.1 DEFINITIONS .....	10
3.2 SYMBOLS .....	10
3.3 ABBREVIATIONS.....	10
<b><u>4 PRINCIPLES TO ENSURE COMPATIBILITY .....</u></b>	<b><u>11</u></b>
4.1 INTRODUCTION .....	11
4.2 LEVEL 1 OF PRINCIPLES: PROTOCOL LEVEL.....	11
4.3 LEVEL 2 OF PRINCIPLES: MESSAGE LEVEL .....	11
4.3.1 NEW MESSAGES.....	11
4.3.2 PARTIAL DECODING.....	11
4.4 LEVEL 3 OF PRINCIPLES: INFORMATION ELEMENT LEVEL .....	11
4.4.1 NEW IE.....	11
4.4.2 OPTIONAL IE .....	11
4.4.3 ADDING MANDATORY IE.....	11
4.4.4 MISSING OPTIONAL IE.....	12
4.4.5 COMPREHENSION REQUIRED .....	12
4.4.6 PARTIAL DECODING.....	12
4.5 LEVEL 4 OF PRINCIPLES: VALUES LEVEL .....	12
4.5.1 RESERVED VALUES AND SPARE FIELDS.....	12
4.5.2 UNSPECIFIED VALUES .....	12
4.5.3 MISSING OPTIONAL VALUE.....	12
4.5.4 EXTENSION OF VALUE SET .....	12
<b><u>5 MESSAGE SEQUENCE CHARTS .....</u></b>	<b><u>12</u></b>
<b><u>6 SPECIFICATION AND DESCRIPTION LANGUAGE .....</u></b>	<b><u>12</u></b>
<b><u>7 MESSAGE SPECIFICATION .....</u></b>	<b><u>13</u></b>
7.1 SUMMARY OF WHAT HAS BEEN AGREED .....	13
7.2 DEFINITIONS .....	13
7.3 LOGICAL DESCRIPTION .....	13
7.4 MESSAGE CONTENTS DESCRIPTION.....	13
7.5 COMPILABILITY OF THE TRANSFER SYNTAX .....	13

<b>7.6 EFFICIENCY/COMPACTNESS .....</b>	<b>14</b>
<b>7.7 EVOLVABILITY/EXTENSIBILITY .....</b>	<b>14</b>
<b>7.8 INTER IE DEPENDENCY .....</b>	<b>14</b>
<b>7.9 INTRA IE DEPENDENCY .....</b>	<b>14</b>
<b>7.10 SUPPORT OF ERROR HANDLING .....</b>	<b>14</b>
<b><u>8 USAGE OF TABULAR FORMAT .....</u></b>	<b><u>14</u></b>
<b>8.1 TABULAR DESCRIPTION OF MESSAGES AND IES .....</b>	<b>14</b>
8.1.1 MESSAGE DESCRIPTION .....	14
8.1.1.1 The general description.....	14
8.1.1.2 The Information Element table .....	14
8.1.1.2.1 Presence and range columns .....	15
8.1.1.2.1.1 Mandatory .....	15
8.1.1.2.1.2 Optional .....	15
8.1.1.2.1.3 Conditional.....	15
8.1.1.2.1.4 Choice .....	16
8.1.1.2.1.5 Sets.....	16
8.1.1.2.2 IE type and reference column .....	16
8.1.1.2.3 Semantics description .....	17
8.1.1.3 Explanatory clauses .....	17
8.1.2 IE TYPE DESCRIPTION.....	17
<b>8.2 BASIC TYPES.....</b>	<b>17</b>
8.2.1 ENUMERATED.....	17
8.2.2 BOOLEAN.....	17
8.2.3 INTEGER .....	17
<b><u>9 USAGE OF ASN.1 .....</u></b>	<b><u>18</u></b>
<b>9.1 MESSAGE LEVEL .....</b>	<b>18</b>
9.1.1 MESSAGES .....	18
9.1.2 MESSAGE DEFINITION .....	19
9.1.3 MESSAGES AND ASN.1 MODULES .....	20
9.1.4 MESSAGES AND SDL.....	21
<b>9.2 INFORMATION ELEMENT LEVEL .....</b>	<b>22</b>
9.2.1 MESSAGE CONTENTS.....	22
9.2.2 OPTIONAL IES AND DEFAULT VALUES .....	22
9.2.3 NEW IES .....	23
9.2.4 COMPREHENSION REQUIRED.....	23
9.2.5 PARTIAL DECODING.....	23
9.2.6 ERROR SPECIFICATION .....	23
<b>9.3 VALUE LEVEL.....</b>	<b>24</b>
9.3.1 EXTENSIBILITY .....	24
9.3.2 COMPREHENSION REQUIRED.....	24
9.3.3 PARTIAL DECODING.....	24
9.3.4 BOOLEAN.....	24
9.3.5 INTEGER .....	24
9.3.6 ENUMERATED.....	25
9.3.7 BIT STRING .....	25
9.3.8 OCTET STRING .....	26
9.3.9 NULL .....	26
9.3.10 SEQUENCE.....	26
9.3.11 SEQUENCE-OF.....	27

9.3.12 CHOICE.....	27
9.3.13 RESTRICTED CHARACTER STRING TYPES.....	28
9.3.14 IES AND ASN.1 MODULES.....	28

## **10 MESSAGE TRANSFER SYNTAX SPECIFICATION ..... 30**

<b>10.1 SELECTION OF TRANSFER SYNTAX SPECIFICATION METHOD.....</b>	<b>30</b>
10.1.1 TRANSFER SYNTAX SPECIFICATION METHOD ALTERNATIVES.....	30
10.1.2 COMPARISON OF METHODS .....	30
10.1.3 OTHER ISSUES .....	31
<b>10.2 CSN.1 ENCODING FOR ASN.1 TYPES.....</b>	<b>31</b>
10.2.1 MESSAGE STRUCTURES .....	31
10.2.2 BOOLEAN.....	31
10.2.3 INTEGER .....	32
10.2.4 ENUMERATED.....	32
10.2.5 BIT STRING .....	33
10.2.6 OCTET STRING .....	33
10.2.7 NULL .....	34
10.2.8 SEQUENCE.....	34
10.2.9 SEQUENCE-OF.....	36
10.2.10 CHOICE.....	37
10.2.11 RESTRICTED CHARACTER STRINGS.....	37
<b>10.3 SPECIALISED ENCODING.....</b>	<b>38</b>
10.3.1 GENERAL NOTATION .....	38
10.3.2 SHORTHAND NOTATION.....	39

## **1 SCOPE..... 7**

## **2 REFERENCES..... 7**

## **3 DEFINITIONS, SYMBOLS AND ABBREVIATIONS..... 7**

<b>3.1 DEFINITIONS .....</b>	<b>7</b>
<b>3.2 SYMBOLS.....</b>	<b>7</b>
<b>3.3 ABBREVIATIONS.....</b>	<b>7</b>

## **4 PRINCIPLES TO ENSURE COMPATIBILITY..... 8**

<b>4.1 INTRODUCTION .....</b>	<b>8</b>
<b>4.2 LEVEL 1 OF PRINCIPLES: PROTOCOL LEVEL.....</b>	<b>8</b>
<b>4.3 LEVEL 2 OF PRINCIPLES: MESSAGE LEVEL .....</b>	<b>8</b>
4.3.1 NEW MESSAGES.....	8
4.3.2 PARTIAL DECODING.....	8
<b>4.4 LEVEL 3 OF PRINCIPLES: INFORMATION ELEMENT LEVEL .....</b>	<b>8</b>
4.4.1 NEW IE.....	8
4.4.2 OPTIONAL IE.....	8
4.4.3 ADDING MANDATORY IE.....	8
4.4.4 MISSING OPTIONAL IE.....	9
4.4.5 COMPREHENSION REQUIRED.....	9

4.4.6 PARTIAL DECODING.....	9
<b>4.5 LEVEL 4 OF PRINCIPLES: VALUES LEVEL .....</b>	<b>9</b>
4.5.1 RESERVED VALUES AND SPARE FIELDS.....	9
4.5.2 UNSPECIFIED VALUES .....	9
4.5.3 MISSING OPTIONAL VALUE.....	9
4.5.4 EXTENSION OF VALUE SET .....	9
<b><u>5 MESSAGE SEQUENCE CHARTS .....</u></b>	<b><u>9</u></b>
<b><u>6 SPECIFICATION AND DESCRIPTION LANGUAGE .....</u></b>	<b><u>9</u></b>
<b><u>7 MESSAGE SPECIFICATION .....</u></b>	<b><u>10</u></b>
7.1 SUMMARY OF WHAT HAS BEEN AGREED .....	10
7.2 DEFINITIONS .....	10
7.3 DESCRIPTION LEVEL.....	10
7.4 COMPILABILITY OF THE TRANSFER SYNTAX .....	10
7.5 EFFICIENCY/COMPACTNESS .....	10
7.6 EVOLVABILITY/EXTENSIBILITY .....	10
7.7 INTER IE DEPENDENCY .....	10
7.8 INTRA IE DEPENDENCY .....	11
7.9 SUPPORT OF ERROR HANDLING .....	11
<b><u>8 USAGE OF ASN.1 .....</u></b>	<b><u>11</u></b>
<b>8.1 MESSAGE LEVEL .....</b>	<b>11</b>
8.1.1 MESSAGES.....	11
8.1.2 MESSAGE DEFINITION .....	12
8.1.3 MESSAGES AND ASN.1 MODULES .....	13
8.1.4 MESSAGES AND SDL.....	14
<b>8.2 INFORMATION ELEMENT LEVEL .....</b>	<b>14</b>
8.2.1 MESSAGE CONTENTS .....	15
8.2.2 OPTIONAL IES AND DEFAULT VALUES .....	15
8.2.3 NEW IES .....	15
8.2.4 COMPREHENSION REQUIRED .....	15
8.2.5 PARTIAL DECODING.....	16
8.2.6 ERROR SPECIFICATION .....	16
<b>8.3 VALUE LEVEL.....</b>	<b>16</b>
8.3.1 EXTENSIBILITY .....	17
8.3.2 COMPREHENSION REQUIRED .....	17
8.3.3 PARTIAL DECODING.....	17
8.3.4 BOOLEAN.....	17
8.3.5 INTEGER .....	17
8.3.6 ENUMERATED.....	18
8.3.7 BIT STRING .....	18
8.3.8 OCTET STRING .....	19
8.3.9 NULL .....	19
8.3.10 SEQUENCE.....	19
8.3.11 SEQUENCE OF.....	20
8.3.12 CHOICE.....	20

8.3.13 RESTRICTED CHARACTER STRING TYPES.....	21
8.3.14 IES AND ASN.1 MODULES.....	21
<b>9 MESSAGE TRANSFER SYNTAX SPECIFICATION .....</b>	<b>23</b>
<b>9.1 SELECTION OF TRANSFER SYNTAX SPECIFICATION METHOD.....</b>	<b>23</b>
9.1.1 TRANSFER SYNTAX SPECIFICATION METHOD ALTERNATIVES.....	23
9.1.2 COMPARISON OF METHODS.....	23
9.1.3 OTHER ISSUES.....	24
<b>9.2 CSN.1 ENCODING FOR ASN.1 TYPES.....</b>	<b>24</b>
9.2.1 MESSAGE STRUCTURES.....	24
9.2.2 BOOLEAN.....	24
9.2.3 INTEGER.....	25
9.2.4 ENUMERATED.....	25
9.2.5 BIT STRING.....	26
9.2.6 OCTET STRING.....	26
9.2.7 NULL.....	27
9.2.8 SEQUENCE.....	27
9.2.9 SEQUENCE OF.....	29
9.2.10 CHOICE.....	30
9.2.11 RESTRICTED CHARACTER STRINGS.....	30
<b>9.3 SPECIALISED ENCODING.....</b>	<b>31</b>
9.3.1 GENERAL NOTATION.....	31
9.3.2 SHORTHAND NOTATION.....	32



---

## Foreword

This Technical Report has been produced by the 3GPP.

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of this TR, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version 3.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 Indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the specification;

---

# 1 Scope

The present document provides a guideline for using formal languages in protocol description of UMTS stage 2 and 3 and rules for error handling. This document covers all interfaces involved in radio access protocols such as Uu, Iu, Iur and Iub.

---

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1]	X.680 : "Abstract Syntax Notation One (ASN.1): Specification of the basic notation"
[2]	X.681 : "Abstract Syntax Notation One (ASN.1): Information object specification"
[3]	X.682 : "Abstract Syntax Notation One (ASN.1): Constraint specification"
[4]	X.690 : "ASN.1 Encoding Rules : Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)"
[5]	X.691 : "ASN.1 Encoding Rules - Specification of Packed Encoding Rules (PER)"
[5]	CSN.1 : "specification, version 2.0"
[6]	Z.100 : " Specification and description language (SDL)"
[7]	Z.105 : " SDL Combined with ASN.1 (SDL/ASN.1)"
[8]	Z.120 : "Message Sequence Chart (MSC)"
[9]	ISO/IEC 9646-3 : "The Tree and Tabular Combined Notation"

---

# 3 Definitions, symbols and abbreviations

## 3.1 Definitions

## 3.2 Symbols

## 3.3 Abbreviations

---

## 4 Principles to ensure compatibility

### 4.1 Introduction

The rules edicted intends to prevent incompatibilities between several phases of UMTS evolution (analog to what happened from GSM phase 1to GSM phase 2).

### 4.2 Level 1 of principles: Protocol level

It shall be possible to discriminate different versions of any protocol.

An unknown protocol shall not cause problems to any entity that terminates the protocol. The messages using this protocol discriminator shall be discarded by the receiving entity.

As a consequence, introduction of new protocol shall not disturb any receiving entity

### 4.3 Level 2 of principles: Message level

#### 4.3.1 New messages

New message types shall be able to be introduced without causing any damage. New messages not understood shall be discarded by the receiving entity.

As an exception to this principle it can be possible to define a mechanism that allows a different behaviour when a specific reaction is requested from the receiving entity. This mechanism has to be implemented from the beginning. A special care has to be taken into account when defining broadcast messages and the associated Error handling. Further refinement on this paragraph is needed.

Such a mechanism is not required inside the network part.

#### 4.3.2 Partial decoding

Partial decoding means that a PDU can be decoded in parts. One part forms a complete value that can be separated from other parts. A decoding error in a part does not invalidate previously decoded parts. Subsequent parts are however invalidated because if an error has occurred one can not be sure whether the trailing values are really valid.

Example: A multipurpose PDU contains a list of four PDUs. The two first PDUs are valid but the third one is invalid. The two first are decoded but the third and fourth ones are ignored.

### 4.4 Level 3 of principles: Information element level

#### 4.4.1 New IE

New elements shall generally be discarded when not understood.

In some cases new elements might be taken into account when specific behaviour is requested from the receiving side (e.g. a rejection of the message is expected when the element is not understood: «comprehension required»).

#### 4.4.2 Optional IE

Optional IE should be located after mandatory ones.

#### 4.4.3 Adding mandatory IE

For backward compatibility reasons, addition of mandatory IE shall be avoided. In the first stage of UMTS, a set of functionality is available for each class of UE. Mandatory IE may be added only if they are mandatory for further classes of UE.

#### 4.4.4 Missing optional IE

Missing optional element may be understood as having a certain default value hence a defined meaning.

See also missing values in Values level.

#### 4.4.5 Comprehension required

"Comprehension required" requirement can be associated with an IE. It means that after an IE value has been decoded then the value is validated according to some specified criteria. Failure in validation causes rejection of the message.

Example: A broadcast message contains a list of recipient addresses. If a recipient's address is not included in the list then a recipient ignores the whole message.

#### 4.4.6 Partial decoding

The notion of partial decoding shall also be applied at the IE level.

### 4.5 Level 4 of principles: Values level

#### 4.5.1 Reserved values and spare fields

Reserved values shall be forbidden. Otherwise entity receiving such a value shall reject the message. This would create difficulties when provided on broadcast channel.

Spare field shall be forbidden. Otherwise entity receiving such a spare field shall not make any decoding on that field and shall not reject the message.

#### 4.5.2 Unspecified values

As far as possible default understanding shall be provided for unspecified values.

#### 4.5.3 Missing optional value

A default value may be specified for the receiver when the sender did not include a field containing this value.

#### 4.5.4 Extension of value set

There are cases when a data field may originally contain only a definite set of values. In the future the set of values grows but the number new values can be anticipated. There are two alternative ways to specify extension of a value set:

- 1) Infinite extension of a value set. Example: The first version of a data field may contain only values 0-3. In the future the field may contain any positive integer value.
- 2) Finite extension of a value set. Example: The first version of a data field may contain only values 0-3. In the future values 4-15 shall also be used.

---

## 5 Message Sequence Charts

It is agreed to recommend the use MSCs as one of the formal methods.

MSCs is adapted for description of normal behaviour of protocol layers between peer entities and/or through SAPs. So it may be used in stage 2 of protocol description.

---

## 6 Specification and Description Language

The groups are encouraged to use of SDL where appropriate. The SDL code included in the standards should follow the descriptive SDL guidelines from ETSI TC-MTS (DEG MTS-00050) as closely as possible.

The groups themselves should decide how SDL is used.

In some protocol parts, text is more adapted (eg : algorithm or multiplexing), in some other parts SDL is better.

SDLs is adapted for describing the observable behaviour of a protocol layer.

[Editor's note : the following text is appended and comes from the minutes of the ETSI SMG2 Specification Methods AdHoc #1. However, it could not be agreed whether there is always normative text description for everything and SDL may be there in addition, or whether part of the normative standard can use SDL only. Recommendation for the groups to use descriptive SDL was also discussed but cannot be agreed before the documentation (DEG MTS/00050) has been studied more.]

## 7 Message specification

### 7.1 Summary of what has been agreed

- 1) use subset of ASN.1 (compatible with Z.105) for definition of message contents description~~abstract syntax~~ of protocol messages
- 2) there is a need for a default encoding, which can be applied in most cases
- 3) there is a need for a special encoding e.g. by means of CSN.1.
- 4) how to link the message contents description~~abstract syntax~~ to the different encoding rules needs to be specified.
- 5) ASN.1 definitions can be used within SDL and TTCN parts of the specifications.

### 7.2 Definitions

Message descriptions are divided into three levels:

- a logical description, which describes messages and relevant information elements in an easily understandable, semi-formal fashion;
- a message contents description, which describes the messages formally and completely in an abstract fashion; and
- a message encoding, which defines the encoded messages (i.e. what is carried as a bit string).

~~Description of message is divided into two levels : an abstract description which defines data to be sent to the peer entity as descriptive values (analog to parameter passing), and a concrete description (also named transfer syntax) which defines the encoded PDU (i.e. : what is carried as a bit string).~~

### 7.3 Logical description~~Description level~~

~~The logical description of messages shall be done using tabular format specified in clause 8 of this document. The description of the protocol should utilise an abstract description of the manipulated elements which is independent from transfer syntax.~~

~~A subset of ASN.1 (compatible with Z.105) should be used for definition of abstract syntax of protocol messages.~~

### 7.4 Message contents description

The message contents descriptions shall be written using ASN.1. The message encoding shall be based on the ASN.1 description.

### 7.5 Compilability of the transfer syntax

The transfer syntax should allow as automatic as possible compilers which transform a sequence of received bits into a sequence of IEs which can be utilised by the protocol machine. CSN.1 may be used. A link between message contents description~~abstract syntax~~ and transfer syntax needs to be specified.

## 7.6 Efficiency/Compactness

The transfer syntax should allow to minimise the size of messages if so necessary. It should allow protocol dependant optimisations.

## 7.7 Evolvability/Extensibility

The message contents description ~~abstract syntax~~ shall allow the evolution of the protocol.

The transfer syntax shall keep the same level of compactness as the initial design.

## 7.8 Inter IE dependency

The message contents description ~~abstract syntax~~ shall allow that presence of IEs depends on values in previous IEs.

The description of messages should avoid dependency between values in different IE. Indeed, it would mean that values are not independent and that there is a redundancy.

## 7.9 Intra IE dependency

The abstract and transfer syntaxes shall allow that, within an IE, some fields depend on previous ones.

## 7.10 Support of error handling

The syntax used should support optional IEs, default values, partial decoding, "comprehension required" and extensibility as defined above.

# 8 Usage of tabular format

## 8.1 Tabular description of messages and IEs

A protocol specification should include a 'Tabular description' sub-clause, including

- A message description sub-clause;
- An IE description sub-clause

### 8.1.1 Message description

A 'Message description' sub-clause includes one sub-clause per message.

A message is described with, in this order:

- A general description, including the flow the message belongs to (e.g., SAP, direction,...); this indirectly points to the message header description, which is not described again for each message;
- A table describing a list of information elements;
- Explanatory clauses, mainly for describing textually conditions for presence or absence of some IEs.

#### 8.1.1.1 The general description

TBI

#### 8.1.1.2 The Information Element table

The table is composed of 5 columns, labelled and presented as shown below.

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>

The leading spaces and tab of the text in the 'IE/Group name' column are significant. They determine the indentation of the line.

Each line corresponds either to an IE or to a group. A group includes all the IEs in following lines until, and not including, a line with the same indentation as the group line.

Dummy groups can be used for legibility: the following IE/Group has the same indentation. For such dummy groups, the presence/range columns are meaningless and should be left empty.

The 'IE type and reference' column is not filled in the case of a group line and must be filled for IEs. IE/Group Name column

In this column, the leading spaces or tabs are significant. They are referred hereafter as the indentation of the line.

This column gives the local name of the IE or of a group of IEs. This name is significant only within the scope of the described message, and must appear only once in the column at the same level of indentation. It is a free text, which should be chosen to reflect the meaning of the IE or group of IEs. This text is to be used followed by the key word IE, the whole enclosed between quotes [or in italics] to refer to the IE or the group of IEs in the procedural description.

The first word 'choice' has a particular meaning, and must not be used otherwise.

#### 8.1.1.2.1 Presence and range columns

These columns provide most of the information about the presence, absence and number of copy of the IE (in the message or in the group) or group of IEs. The different possibilities for these columns are described one by one.

[Editor's note : the word range seems ambiguous as it does not define a range in terms of a value range but the min and max number of this IE (repetition) that may be present in a message. It is closely related to Presence column as in the title.]

##### 8.1.1.2.1.1 Mandatory

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
<u>Name</u>	<u>M</u>			

The range column must be left empty.

For an IE not belonging to a group this indicates the 'Name IE' is always necessary in the message.

For a group not belonging to another group, this means that the 'Name group' is always necessary in the message, i .e., that the presence/range indications of the children IEs/groups apply directly.

For an IE or a group belonging to another group, this means that if the parent group is present, then the 'Name group' is necessary in the message.

##### 8.1.1.2.1.2 Optional

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
<u>Name</u>	<u>O</u>			

The range column is empty.

This indicates that the 'Name IE' or 'Name group' is not necessary in the message, and that the sender can choose not to include it.

##### 8.1.1.2.1.3 Conditional

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
	<u>C cond</u>			

The range column is empty.

This indicates that the presence of absence of the IE or group of IE depends on the value of some other IE or IEs, and/or on the message flow (e.g., channel, SAP). The condition is to be described in a textual form in an explanatory clause. *cond* stands for a free text that is used as a reference in the title of the explanatory clause.

8.1.1.2.1.4 Choice

This is particular group of at least two children.

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
<u>Choice name</u>				
<u>Name1</u>				
<u>Name2</u>				

A ‘choice’ group is distinguished from standard groups by the use of ‘choice’ as first word in the name.

The presence/range columns are filled normally for the group line. They are not filled for the children lines: the implicit value is conditional, one condition being that one and only one of the children is present if the group is present.

If additional conditions exist for the choice, they are explained in an explanatory clause.

8.1.1.2.1.5 Sets

The two lines below indicate different allowed alternatives.

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
<u>Name</u>		<u>nn..pp</u>		
<u>Name</u>		<u>nn..indefinite</u>		
<u>Name</u>		<u>nn..sym2</u>		
<u>Name</u>		<u>sym1..pp</u>		
<u>Name</u>		<u>sym1..sym2</u>		
<u>Name</u>	<u>C cond</u>	<u>nn ..pp</u>		
<u>Name</u>	<u>C cond</u>	<u>nn..indefinite</u>		
<u>Name</u>	<u>C cond</u>	<u>nn..sym2</u>		
<u>Name</u>	<u>C cond</u>	<u>sym1..pp</u>		
<u>Name</u>	<u>C cond</u>	<u>sym1..sym2</u>		

Where *nn* and *pp* stand for positive integers, and *sym1* and *sym2* for symbolic names. The presence column can be empty or C.

This indicates that a number of copies of the element can appear. The order is significant. The reference should use the bracket notation (e.g., ‘Name[1] IE’) to refer to a specific copy; numbering starts by 1.

The *nn..pp* case indicates that the number of copies is between *nn* and *pp*, inclusively. The number *nn* is positive or null. The number *pp* must be equal or greater than *nn*. The 1..1 case should be avoided, and a M indication used instead. Similarly, the 0..1 case should be avoided and replaced by an O indication.

The *nn..indefinite* case indicates that the number of copies is *nn* or greater. The number *nn* is positive or null. This indicates that there is no limit on the requirement on the number of copies. It is however allowed that the transfer syntax puts some practical limits on the maximum number of copies.

The use of a symbolic name for one or the other of the range bounds indicates that the value is given in a textual clause. This is necessary the case when the bound depends is conditional to the value of some other IE or IEs.

The presence column is set to C followed by a condition name to indicate that the number of copies is conditional to the value of some other IE or IEs, or on the flow. An explanatory clause describes the condition. Otherwise, the column is left empty.

8.1.1.2.2 IE type and reference column

This column is not filled for groups and must be filled for IEs.

This column includes the reference to a more detailed abstract description of the IE. This includes:



- a) A reference to a sub-clause in the Information Element Description clause in the same document; Typically the sub-clause number and titles are given, and if possible this should be a hypertext link;
- b) A reference to another document, and to a sub-clause in the Information Element Description clause in the indicated document; typically only the sub-clause title is indicated;
- c) A reference to a sub-clause of <this document: i.e., the TS where the tabular notation is described>, with possibly additional information as described.

### 8.1.1.2.3 Semantics description

Filling this column is optional. It should be use to clarify the meaning of the IE or group of IE, as a summary of their use as described in the procedural part.

### 8.1.1.3 Explanatory clauses

This includes the sub-clauses needed to elaborate conditions and symbolic names (e.g., range bounds). There must be one explanatory clause for each named condition, and for each symbolic name. The text must give the information sufficient to decide whether the IE/group is to be included or not, or the value of the symbolic name.

## 8.1.2 IE type description

This describes IE types referred elsewhere, either in the description of a message or in the description of another IE type. The description of an IE type must be as generic as possible, i.e., independent of any specific use.

An 'IE description' sub-clause includes one sub-clause per IE type.

The description of an IE type is done as a table similar to that used for the description of messages.

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
----------------------	-----------------	--------------	------------------------------	------------------------------

The different columns are filled exactly as message description columns are filled.

## 8.2 Basic types

To reduce the text in tabular descriptions, some basic abstract types of IE are defined in this document.

### 8.2.1 Enumerated

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
			<u>Enumerated (<i>c1, c2, c3</i>)</u>	

Where *c1, c2, c3* stands for a list of 2 or more symbolic names separated by commas.

This indicates that the value of the IE when present takes one and only one of the values indicated in the list.

### 8.2.2 Boolean

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
			<u>Boolean</u>	

This is shorthand for:

			<u>Enumerated (<i>False, True</i>)</u>	
--	--	--	--	--

### 8.2.3 Integer

The different lines below indicate different alternatives.

<u>IE/Group Name</u>	<u>Presence</u>	<u>Range</u>	<u>IE Type and reference</u>	<u>Semantics description</u>
			<u>Integer</u>	
			<u>Integer (<i>nn..pp</i>)</u>	
			<u>Integer (<i>nn..indefinite</i>)</u>	
			<u>Integer (<i>sym1..pp</i>)</u>	

			<u>Integer (nm..sym2)</u>	
			<u>Integer (sym1..sym2)</u>	

Where nm and pp stand for positive, negative or null integers, and sym1 and sym2 for symbolic names. The presence column must be left empty.

This corresponds to whole or a subset of the set of positive, negative or null integers, as defined by usual mathematics.

The range notation is self-explanatory. In the two unbounded cases, practical bounds may be imposed by the transfer syntax.

## 9 Usage of ASN.1

The following clauses contain guidelines for specification of protocol messages with ASN.1. The purpose of ASN.1 is to make it possible to specify message contents description ~~abstract syntax~~ of a message (i.e. what is the contents of a message) separately from its transfer syntax (i.e. how a message is encoded for transmission). The features that ASN.1 provides include specification of:

- Extensibility (both structural and extension of value set)
- Optional IEs and values (see the clauses 9.2.2 and 9.3.10)
- Default values (see the clauses 9.2.2 and 9.3.10)
- Comprehension required (see the clause 9.2.4)
- Inter/Intra IE dependency (see the clause 9.3.10)
- Specification of partial decoding (see the clause 9.2.5)

The clause 11 specifies how message transfer syntax is specified. It should be noted that importance of some transfer syntax properties must be determined early during specification because of their effect on ~~message~~ message contents description ~~abstract syntax~~ specification possibilities. The properties are **compactness** and **extensibility**. If extreme compactness is required then extensibility must be restricted. If good extensibility is required then compromises must be done regarding compactness. The sections concerning these issues are marked in the following clauses as **COMPACTNESS** and **EXTENSIBILITY**.

### 9.1 Message level

#### 9.1.1 Messages

It is presumed that messages share the same structure, namely that they contain an identification part and a contents part. An identification part contains an IE that identifies a message among all messages in some context. A contents part contains message specific IEs.

Example: A protocol layer XYZ contains three messages: A, B and C. The structure of the messages is as presented in the figure 3-1.

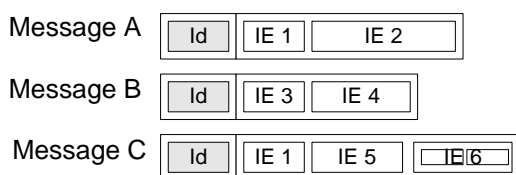


Figure 3-1: Three example messages

Messages are specified using ASN.1 [1]. There are three ASN.1 types, *MessageA*, *MessageB* and *MessageC*, which contain definitions for the contents of the above messages. The mapping between the message contents types and message identifiers is as follows:

Message id	Type of message contents
1	MessageA
2	MessageB
3	MessageC

New message types will be introduced in the future.

In cases where different PDUs have different identification schemes it is possible to apply this categorisation for a set of PDUs that share the same identification scheme.

## 9.1.2 Message definition

In order to capture information in the previous clause the following three things must be defined:

1. A structure for the table
2. The table itself
3. A generic message structure which can contain both message identifier IE and message contents IEs (i.e. id 1 + *MessageA*, id 2 + *MessageB*, id 3 + *MessageC*)

The table structure is defined as follows using ASN.1 classes [2]:

```

XYZ-MESSAGE ::= CLASS {
    &id      MessageId,
    &Type
}
WITH SYNTAX {
    &id &Type
}

MessageId ::= INTEGER (0..63)

```

The table is defined as follows:

```

XYZ-Messages XYZ-MESSAGE ::= {
    { messageA-id MessageA } |
    { messageB-id MessageB } |
    { messageC-id MessageC } |
    ...
    -- Extension marker => additional messages
    -- can be introduced.
}

messageA-id MessageId ::= 1
messageB-id MessageId ::= 2
messageC-id MessageId ::= 3

```

The following type represents the generic message structure that can carry values of the messages specified in the *XYZ-Messages* table.

```

XYZ-Message ::= SEQUENCE {
    id          XYZ-MESSAGE.&id    ({XYZ-Messages}),
                -- MessageId: 1, 2 or 3

    contents   XYZ-MESSAGE.&Type  ({XYZ-Messages}@id)
                -- id=1 => MessageA, id=2 => MessageB, id=3 => MessageC
}

```

The above definition means that if *id* is 1 then the *Message* type is equivalent to the following type:

```

XYZ-Message ::= SEQUENCE {
    id          MessageId,          -- 1
    contents   SEQUENCE {
        ie1     IE1,
        ie2     IE2
    }
}

```

If *id* is 2 then the type is equivalent to the following type:

```

XYZ-Message ::= SEQUENCE {
    id          MessageId,          -- 2
    contents   SEQUENCE {
        ie3     IE3,
        ie4     IE4
    }
}

```

### 9.1.3 Messages and ASN.1 modules

ASN.1 definitions shall be placed in ASN.1 modules such that definitions in a module form a logical unit. For example PDUs definitions for one protocol layer could be in one ASN.1 module and IE definitions in another.

The tagging mode for the modules shall be "AUTOMATIC TAGS".

Example: A message definition module for the XYZ protocol layer.

```

XYZ-Messages DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

XYZ-Messages XYZ-MESSAGE ::= {
  { messageA-id MessageA } |
  { messageB-id MessageB } |
  { messageC-id MessageC } |
  ... -- Additional messages can be introduced.
}

MessageA ::= SEQUENCE {
  -- Message contents
}

messageA-id MessageId ::= 1

MessageB ::= SEQUENCE {
  -- Message contents
}

messageB-id MessageId ::= 2

MessageC ::= SEQUENCE {
  -- Message contents
}

messageC-id MessageId ::= 3

END

```

### 9.1.4 Messages and SDL

The identifiers *messageA-id*, *MessageA*, *messageB-id*, etc. can be used in descriptive SDL when protocol behaviour is specified. Note that classes and objects can not (yet) be referenced in SDL. Types and values however can be imported to SDL definitions. The figures below contain some examples about usage of ASN.1 in SDL specifications.

```

imports
  MessageA, messageA_id,
  MessageId
from SomeASN1Module;

signal XYZ_MessageA(
  MessageId, MessageA);

dcl aVariable MessageA;

```

Figure 3-2: Import and use of ASN.1 definitions in SDL.

```

XYZ_MessageA(
  messageA_id,
  aVariable)

```

Figure 3-3: Sending of a message id and contents.

## 9.2 Information element level

Messages consist of information elements.

The following ASN.1 message types are used in the following clauses.

```

MessageA ::= SEQUENCE {
    ie1 IE1,          -- A mandatory IE.

    ie2 IE2 OPTIONAL,      -- An optional IE.

    ...                -- An extension marker.
}

MessageB ::= SEQUENCE {
    ie3 IE3
    (CONSTRAINED BY {-- ComprehensionRequired(is for receiver) --}
    !comprehensionRequiredFailure) ,

    ie4 IE4 DEFAULT 0,    -- An optional IE with a default value.

    ...
}

MessageC ::= SEQUENCE {
    ie1 IE1
    (CONSTRAINED BY {-- PartialDecoding(OnErrorIgnoreRest) --}
    !partialDecodingFailure)
    OPTIONAL,

    ie5 IE5
    (CONSTRAINED BY {-- PartialDecoding(OnErrorIgnoreRest) --}
    !partialDecodingFailure)
    OPTIONAL,

    ... ,                -- An extension marker

    ie6 IE6
    (CONSTRAINED BY {-- PartialDecoding(OnErrorIgnoreRest) --}
    !partialDecodingFailure)
    OPTIONAL             -- A new IE
}

-- Error codes
comprehensionRequiredFailure INTEGER ::= 1
partialDecodingFailure      INTEGER ::= 2

```

### 9.2.1 Message contents

A message contents structure is defined using a sequence type (9.3.10).

Example: *MessageA*, *MessageB* and *MessageC* are message contents structures.

### 9.2.2 Optional IEs and default values

An IE can be marked as optional.

**COMPACTNESS:** Optional IEs shall be after mandatory ones.

Example: *MessageA.ie2* is an optional IE.

ie2	IE2	<b>OPTIONAL</b>
-----	-----	-----------------

An IE can be marked as being optional and having a default value. In those cases a missing optional IE may be understood as having a certain value hence a defined meaning.

Example: *MessageB.ie4* is an optional IE with a default value.

```
ie4 IE4 DEFAULT 0
```

### 9.2.3 New IEs

**EXTENSIBILITY:** If new IEs will be added to a message then the message contents structure must be specified as extensible using the ellipsis notation (...). New IEs shall be added after the extension marker. New IEs shall be optional or shall have default values.

Example: *MessageC.ie6* is an additional optional IE.

```
... ,
ie6 IE6 OPTIONAL
```

### 9.2.4 Comprehension required

"Comprehension required" requirement can be associated with an IE. It means that after an IE value has been decoded then the value is validated. Failure in validation causes rejection of the message.

The requirement is specified as an extension to ASN.1 by using user defined constraints [3]. The comment part of the constraint shall be of the form:

*ComprehensionRequired(<additional constraint>)*

where <additional constraint> specifies the rule that the IE must satisfy.

Example: The *MessageB* is a broadcast message. The *ie3* IE contains recipient addresses. It is not until the addresses have been decoded when a receiver can decide whether it should decode the rest of the message or not.

```
ie3 IE3
(CONSTRAINED BY {-- ComprehensionRequired(is for receiver) --}
!comprehensionRequiredFailure) ,
```

### 9.2.5 Partial decoding

"Partial decoding" means that a PDU can be decoded in parts. One part forms a complete value that can be separated from other parts. A decoding error in a part does not invalidate previously decoded parts. Subsequent parts are however invalidated.

"Partial decoding" is specified as an extension to ASN.1 using user defined constraints. The comment of constraint shall be of the form:

*PartialDecoding(<OnErrorClause>)*

where <OnErrorClause> specifies action in case of a decoding error. The possible alternatives are:

- OnErrorIgnoreRest: End decoding, ignore rest of the message

Example: The *MessageC* is a multipurpose message. The IEs *ie1*, *ie5* and *ie6* are independent of each other.

```
ie1 IE1
(CONSTRAINED BY {-- PartialDecoding(OnErrorIgnoreRest) --}
!partialDecodingFailure)
```

### 9.2.6 Error specification

An error specification can be associated with user defined constraints.

A simple integer value can be associated with an exception specification or as elaborate structured value as needed.

Example: If decoding of *ie1* fails then decoder returns the error code *partialDecodingFailure*.

```
ie1 IE1
  (CONSTRAINED BY {-- PartialDecoding(OnErrorIgnoreRest) --}
   !partialDecodingFailure)
```

## 9.3 Value level

Information elements consist of values.

If the CSN.1 specified default syntax (see the clause 10) is used as a transfer syntax then only the following ASN.1 types can be used in the value level:

- Boolean (9.3.4)
- Integer (9.3.5)
- Enumerated (9.3.6)
- Bit string (9.3.7)
- Octet string (9.3.8)
- Null (9.3.9)
- Sequence (9.3.10)
- Sequence-of (9.3.11)
- Choice (9.3.12)
- Character string types (9.3.13)

Otherwise there are no restrictions on usage of ASN.1 types.

### 9.3.1 Extensibility

**COMPACTNESS:** In the value level use of ASN.1 extensibility is forbidden unless otherwise stated in the following clauses.

### 9.3.2 Comprehension required

"Comprehension required" can be applied to components of sequence types, alternatives of choice types and elements of sequence-of types. See 9.2.4

### 9.3.3 Partial decoding

"Partial decoding" can be applied to components of sequence types, alternatives of choice types and elements of sequence-of types. See 9.2.5

### 9.3.4 Boolean

Example: A simple boolean type.

```
Flag ::= BOOLEAN

setFlag Flag ::= TRUE
```

### 9.3.5 Integer

An integer type should be constrained.

**COMPACTNESS:** An integer type shall be constrained to have a finite value set. The value set can be either continuous or non-continuous.

Named numbers can be associated with an integer type.



**COMPACTNESS, EXTENSIBILITY:** If an integer type needs to be extended in the future then two value sets must be defined:

- A value set that specifies the values that can be sent in the current protocol version.
- A value set that specifies all the possible values that can be received now and in the future.

The former value set is specified in a user-defined constraint. The comment part shall be of the form:

*Send(<value set>)*

The latter form is specified using a normal constraint, e.g. a value range constraint.

Examples: Integer types and values.

```
Counter      ::= INTEGER (0..255)  -- 0 <= Counter value <= 255

SparseValueSet ::= INTEGER (0|3|5|6|8|11)

SignedInteger ::= INTEGER (-10..10)

-- idle stands for value 0.
Status       ::= INTEGER { idle(0), veryBusy(3) } (0..3)

-- Send values 0..3 but be prepared to receive values 0..15.
Extensible   ::= INTEGER (0..15)(CONSTRAINED BY {-- Send(0..3) --})

initialCounter Counter      ::= 0

zero         SparseValueSet ::= 0

initialStatus Status       ::= idle
```

### 9.3.6 Enumerated

An enumerated type shall have a continuous finite value set. The enumeration value of the smallest enumeration shall be 0. The list of enumerated values specifies the value set for an enumerated type.

**COMPACTNESS, EXTENSIBILITY:** If an enumerated type needs to be extended in the future then two value sets must be defined as in case of integer types.

Note: An integer type with named numbers can be used as an alternative to an enumerated type.

Example: Enumerated types and value.

```
Enum         ::= ENUMERATED { a, b, c, d }

-- Send values a, b, c or d but be prepared to receive values
-- a, b, c, d, spare4, spare5, spare6 and spare7.
ExtendedEnum ::= ENUMERATED { a, b, c, d, spare4, spare5, spare6, spare7 }
              (CONSTRAINED BY {-- Send(a/b/c/d) --})

aEnum Enum   ::= a
```

### 9.3.7 Bit string

A size constraint shall be specified. It shall be finite.

Named bits can be associated with a bit string type.

Example: Bit string types and values.

```

FixedLengthBitStr ::= BIT STRING (SIZE (10))
VariableLengthBitStr ::= BIT STRING (SIZE (0..10))
BitFlags ::= BIT STRING { a(0), b(1), c(2), d(3)} (SIZE (4))
fix FixedLengthBitStr ::= '0001101100'B
var VariableLengthBitStr ::= '0'B
flg BitFlags ::= { a, c, d } -- '1011'B

```

### 9.3.8 Octet string

A size constraint shall be specified. It shall be finite.

Example: Octet string types and values.

```

FixedLengthOctetStr ::= OCTET STRING (SIZE (10))
VariableLengthOctetStr ::= OCTET STRING (SIZE (0..10))
UpperLayerPDUSegment ::= OCTET STRING (SIZE (1..512))
fix FixedLengthOctetStr ::= '0102030405060708090A'H
var VariableLengthOctetStr ::= 'FF'H

```

### 9.3.9 Null

A null type has only one value, NULL.

Example: Null type as an alternative type of a choice type.

```

IE ::= CHOICE {
    doThis ThisArg,
    doThat ThatArg,
    doNothing NULL
}

```

### 9.3.10 Sequence

A sequence type is a record. Components of a sequence type can be optional or they can have default values. Optional components and components with default values should be after mandatory components.

Inner subtyping can be used to force an optional component to be present or absent in a derived type.

If an optional component is conditionally present or absent then the condition shall be specified in a user defined constraint of the form:

*Condition(<condition expression>)*

<condition expression> shall be such that both sender and receiver are able to evaluate it before a conditional component is encoded or decoded.

"Comprehension required" can be associated with a component of a sequence type.

"Partial decoding" can be associated with a component of a sequence type.

**EXTENSIBILITY:** A sequence type can be marked as extensible. Example: Sequence types and values.

```

Record ::= SEQUENCE {
    flag      Flag,
    counter   Counter,
    bitFlags  BitFlags      OPTIONAL,
    extEnum   ExtendedEnum  DEFAULT a
}

DerivedRecord ::= Record (WITH COMPONENTS {
    bitFlags      PRESENT
})

RecordWithConditionalComponent ::= SEQUENCE {
    mand      INTEGER (0..7),
    opt       BOOLEAN OPTIONAL,
    cond       BOOLEAN
              (CONSTRAINED BY {--Condition(field 'mand' is 7)--})
              OPTIONAL
}

aRecord Record ::= {
    flag      TRUE,
    counter   100
}

anotherRecord DerivedRecord ::= {
    flag      TRUE,
    counter   1000,
    bitFlags  '0101'B      -- bitFlags must be present
}

```

### 9.3.11 Sequence-of

A sequence-of type is a list of some element type. A size constraint shall be specified. It shall be finite.

"Comprehension required" can be associated with an element of a sequence-of type.

"Partial decoding" can be associated with an element of a sequence-of type.

Example: Sequence-of types and values.

```

FixedLengthList      ::= SEQUENCE (SIZE (10)) OF Record
VariableLengthList  ::= SEQUENCE (SIZE (0..10)) OF Status
UpperLayerPDUSegments ::= SEQUENCE (SIZE (1..10)) OF UpperLayerPDUSegment
aList VariableLengthList ::= { idle, 1, 2, veryBusy, 2, 1, idle }

```

### 9.3.12 Choice

A choice type is a variant record. Only one alternative component can be selected.

Inner subtyping can be used to force an alternative to be selected in a derived type.

"Comprehension required" can be associated with an alternative component of a choice type.

"Partial decoding" can be associated with an alternative component of a choice type.

**EXTENSIBILITY:** A choice type can be marked as extensible.

Example: Choice type and value.

```

VariantRecord ::= CHOICE {
    flag      Flag,
    counter   Counter,
    extEnum   ExtendedEnum
}

aVariantRecord variantRecord ::= flag : FALSE

```

### 9.3.13 Restricted character string types

A size constraint shall be specified. It shall be finite.

Example: Character string types.

```

FixedStr  ::= IA5String (SIZE (10))

VarStr    ::= IA5String (SIZE (1..10))

FixedWStr ::= BMPString (SIZE (10))

VarWStr   ::= BMPString (SIZE (1..10))

```

### 9.3.14 IEs and ASN.1 modules

If an IE or a value field within an IE is a parameter from another protocol layer then type for such a field should be defined in another module. In this way there is a clear separation of definitions that are specific to different protocol layers.

Example: The XYZ protocol message *MessageC* contains an IE, which contains an OPQ protocol layer specific field *parameter1*. Type for the field is imported from OPQ specific module.

```

XYZ-Messages DEFINITIONS AUTOMATIC TAGS ::=

BEGIN

IMPORTS
    OPQParameter  -- OPQParameter is not defined within XYZ-Messages
                  -- module.
FROM OPQ-DataTypes;

MessageC ::= SEQUENCE {
    -- Other IEs.
    ie6 IE6 OPTIONAL
}
-- Other definitions ...

IE6 ::= SEQUENCE {
    parameter1  OPQParameter, -- Imported definitions can be
                                -- referred to.
    parameter2  XYZParameter
}

XYZParameter ::= INTEGER (0..255)

END

```

Example: The OPQ protocol layer specific module exports *OPQParameter* type so that other modules can refer it.

```
OPQ-DataTypes DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
EXPORTS
    OPQParameter
;
OPQParameter ::= INTEGER (0..7)
END
```

## 10 Message transfer syntax specification

### 10.1 Selection of transfer syntax specification method

#### 10.1.1 Transfer syntax specification method alternatives

There are the following alternatives for specification of message transfer syntax. One transfer syntax specification method shall be selected for all the messages of a given protocol.

- BER (Basic Encoding Rules, X.690) [4] (including CER and DER)
- PER (Packed Encoding Rules, X691) [5]
- CSN.1 specified encoding, see the following clauses.
- Tabular format

BER and PER are standard ASN.1 encoding rules.

CSN.1 is not standardised but it is publicly available. Use of CSN.1 with ASN.1 is presented in the clause 11.2.

There are no formal rules for specification of tabular format transfer syntax.

#### 10.1.2 Comparison of methods

The following table contains comparison of transfer syntax specification methods. The numbers indicate the rank of a method.

Criteria	BER	PER	CSN.1	Tabular format
Compactness	4	2	1	3
Extensibility	1	1	2 *	2 *

**BER** produces large octet oriented encodings with a lot of extra control information. For radio protocol messages encodings are too large. Thus BER should not be used.

**PER** produces small bit oriented encodings. **BASIC UNALIGNED PER** produces the most compact encodings whereas **BASIC OCTET-ALIGNED PER** pads some fields. PER provides good support for extensibility. The support causes some growth of messages. PER produced encodings are self-delimiting.

**CSN.1** produces smallest encodings. If the CSN.1 alternative is selected then the **COMPACTNESS** and **EXTENSIBILITY** sections in the clause 9 shall be followed. The following clauses do not support the **EXTENSIBILITY** sections. Such support is for FFS. If message level extensibility is specified then CSN.1 produced encodings are not self-delimiting because no length information is encoded for extended fields. If message level extensibility is not used then CSN.1 produces self-delimiting encodings.

**PER** and **CSN.1** are best suited for cases when the structure of a message is complex, e.g. there are many IEs/value fields, some fields are optional or alternative or repetitive etc. PER and CSN.1 produce similar encodings. Selection between PER and CSN.1 should be done according to the following criteria:

- If **compactness** of encoding is the most important requirement and the restricted extensibility is adequate (message extensions are always added to end of a message as new IEs) then the **CSN.1** alternative should be selected.
- If **extensibility** is the most important requirement and compactness is the second then **BASIC UNALIGNED PER** (or **BASIC OCTET ALIGNED PER**) should be selected.

\* The relative order of these two methods can not be definitely stated because they do not provide one fixed way for specification of extensibility.

In case of **tabular** format properties of encoding depend on how a message is specified. This is because there are no formal rules for specification of tabular format transfer syntax. Tabular format is best suited for cases when there are few IEs/value fields and the structure of a message is simple.

### 10.1.3 Other issues

If there is definite size limit for a message (e.g. a broadcast message must fit into one lower layer message) then the **COMPACTNESS** sections in the clause 9 must be followed.<sup>1</sup>

## 10.2 CSN.1 encoding for ASN.1 types

The following clauses specify the CSN.1 [6] specific default encoding for ASN.1 types. The rules specify one-to-one mapping from an abstract syntax to a transfer syntax.<sup>2</sup>

### 10.2.1 Message structures

Message structures shall be encoded as follows:

- A choice of all the messages specified in a message table.
- Selection is done according to the identifier field values.

Example: Encoding of the *XYZ-Message* type in 9.1.2

```
<XYZ_Message> ::=
{
  <id : 000001> <MessageA>
  <id : 000002> <MessageB>
  <id : 000003> <MessageC>
}
;
```

### 10.2.2 Boolean

A boolean type maps to one bit.

Example: Encoding for the boolean type in 9.3.4

<sup>1</sup> Note that it is the responsibility of a specifier to make sure that an message contents description ~~abstract syntax~~ produces a transfer syntax with wanted properties. Additional *user defined constraint* specifications should be considered.

<sup>2</sup> Note that it is possible to specify one set of bits in multiple ways in CSN.1. For example the following descriptions denote the same set of bits:

{000|001|010|011}

0 {0|1} {0|1}

0 bit(2)

This document contains a mapping from an ASN.1 type to an CSN.1 description. Other CSN.1 descriptions that denote the same bit set as presented in the document are also valid.

```

<Flag> ::=
  <BOOLEAN>
;

<BOOLEAN> ::=
  bit
;

```

### 10.2.3 Integer

An integer type is encoded as an UNALIGNED variant of a constrained whole number as specified in PER [5].

Explanation: Let "lb" be lower bound and "ub" be the upper bound of an integer type. A value "n" will be encoded as a value  $e = ("n" - "lb")$  using the minimum number of bits necessary to represent the values in range.

Named numbers do not affect encoding.

If an integer type is marked as extensible as specified in 9.3.5 then the reception and emission value sets are defined separately.

Example: Encodings for integer types in 9.3.5

```

<Counter> ::=
  <INTEGER(8)> -- n = 0..255, e = n-0 = 0..255
;

<SparseValueSet> ::=
  <INTEGER(4)> -- n = 0/3/5/6/8/11, e = n-0 = 0/3/5/6/8/11
  exclude {
    0001|0010|0100|0111|1001|1010|1100|1101|1111
  }
;

<SignedInteger> ::= -- n = -10..10, e = n-(-10) = 0..20
  <INTEGER(5)>
  exclude {
    10101|10110|10111|11000|11001|11010|11011|11100|11101|11110|11111
  }
;

<Status> ::= -- n = 0..3, e = n-0 = 0..3
  <INTEGER(2)>
;

<Extensible> ::= -- n = 0..3, e = n-0 = 0..3, two spare bits
  <INTEGER(4)> = 00 <INTEGER(2)>
;

<INTEGER(nBits)> ::=
  bit(nBits)
;

```

See also 10.3 for specialised encoding.

### 10.2.4 Enumerated

Enumeration values form a value set of 0..(number of enumerations-1). Each enumeration item is encoded as its corresponding numeric value. A value "n" will be encoded using the minimum number of bits necessary to represent all the values in a value set.



If an enumerated type is marked as extensible as specified in 9.3.6 then the reception and emission value sets are defined separately.

Example: Encodings for enumerated types in 9.3.6

```

<Enum> ::=
{
  <a : 00>
  <b : 01>
  <c : 10>
  <d : 11>
}
;

<ExtendedEnum> ::=
{
  <a : 000>
  <b : 001>
  <c : 010>
  <d : 011>
  <spare4 : 100>
  <spare5 : 101>
  <spare6 : 110>
  <spare7 : 111>
}
=
{
  <a : 000>
  <b : 001>
  <c : 010>
  <d : 011>
}
;

```

## 10.2.5 Bit string

A bit string is mapped to a string of bits. If the number of bits may vary then a length field precedes the bit string.

A length field is encoded as an integer field of type INTEGER (lb..ub) where "lb" is the lower bound of size constraint and "ub" is the upper bound.

Named bits do not affect encoding.

Example: Encodings for bit string types in 9.3.7

```

<FixedLengthBitStr> ::=
  bit(10)
;

-- length = 0..10, e = length-0 = 0..10
<VariableLengthBitStr> ::=
  <length : <INTEGER(4)> exclude {1011|1100|1101|1110|1111}>
  bit * val(length)
;

<BitFlags> ::=
  bit(4)
;

```

## 10.2.6 Octet string

An octet string is mapped to a string of bits. If the number of octets may vary then a length field precedes the octet string.

A length field is encoded as an integer field of type INTEGER (lb..ub) where "lb" is the lower bound of the size constraint and "ub" is the upper bound.

Example: Encodings for octet string types in 9.3.8

```

<FixedLengthOctetStr> ::=
  <octet>(10)
;

-- length = 0..10, e = length-0 = 0..10
<VariableLengthBitStr> ::=
  <length : <INTEGER(4)> exclude {1011|1100|1101|1110|1111}>
  <octet> * val(length)
;

-- length = 1..512, e = length-1 = 0..511
<UpperLayerPDUSegment> ::=
  <length : <INTEGER(9)>>
  <octet> * val(length)
;

```

## 10.2.7 Null

The null type is mapped to an empty bit string.

Example: Encoding for the choice type with nested null type in 9.3.9

```

<IE> ::=
  {
    00 <doThis : <ThisArg>>
    01 <doThat : <ThatArg>>
    10 <doNothing : null>
  }
;

```

## 10.2.8 Sequence

Component values are encoded using rules for component types.

Presence or absence of an **optional component** or a **component with a default value** is indicated with a heading bit. Default values do not affect encoding.

If an optional component is forced to be present or absent in a derived type then the heading bit is omitted for the derived type.

If an optional component is conditionally present or absent then the heading bit is omitted. Presence of a conditional component depends on the associated condition expression.

If a sequence type is a message contents type then

- If there is an extension marker then spare bits description follows the last component description.
- Truncation of omitted trailing optional components is allowed.

Example: Encodings for sequence types in 9.3.10

```

<MessageA> ::=
  <ie1 : <IE1>>
  {
    { 0
    | 1 <ie2 : <IE2>>
    }

    <spare bit>(*)
  } // -- Truncation of optional components is allowed
;

<MessageB> ::=
  <ie3 : <IE3>>
  -- ComprehensionRequired(is for receiver)
  -- !comprehensionRequiredFailure

  {
    { 0 -- DEFAULT 0
    | 1 <ie4 : <IE4>>
    }

    <spare bit>(*)
  } // -- Truncation of optional components is allowed
;

<MessageC> ::=
  {
    { 0
    | 1 { <ie1 : <IE1>> ! <PartialDecodingFailure : bit(*) = <no string>>}
    }

    { 0
    | 1 { <ie5 : <IE5>> ! <PartialDecodingFailure : bit(*) = <no string>>}
    }

    { 0
    | 1 { <ie6 : <IE6>> ! <PartialDecodingFailure : bit(*) = <no string>>}
    }

    <spare bit>(*)
  } // -- Truncation of optional components is allowed
;

```

```

<Record> ::=
  <flag : <Flag>>
  <counter : <Counter>>
  {
    0
  | 1 <bitFlags : <BitFlags>>
  }
  {
    0
  | 1 <extEnum : <ExtendedEnum>>
  }
;

<DerivedRecord> ::=
  <flag : <Flag>>
  <counter : <Counter>>
  <bitFlags : <BitFlags>>      -- Note: no heading bit as in <Record>
  {
    0
  | 1 <extEnum : <ExtendedEnum>>
  }
;

<RecordWithConditionalComponent> ::=
  <mand : <INTEGER(3)>>      -- 0..7
  {
    0
  | 1 <opt : <BOOLEAN>>
  }
  {
    null          -- Note: no heading bit
  | <cond : <BOOLEAN>>      -- if 'mand' is 7 then this field is present
  }
;

```

## 10.2.9 Sequence-of

Element values are encoded using rules for the element type. If the number of elements may vary then a length field precedes the element values.

A length field is encoded as an integer field of type INTEGER (lb..ub) where "lb" is the lower bound the of size constraint and "ub" is the upper bound.

Example: Encodings for sequence-of types in 9.3.11

```

<FixedLengthList> ::=
  <Record>(10)
;

-- length = 0..10, e = length-0 = 0..10
<VariableLengthList> ::=
  <length : <INTEGER(4)> exclude {1011|1100|1101|1110|1111}>
  <Status>*val(length)
;

-- length = 1..10, e = length-1 = 0..9
<UpperLayerPDUSegments> ::=
  <length : <INTEGER(4)> exclude {1010|1011|1100|1101|1110|1111}>
  <UpperLayerPDUsegment >*val(length)
;

```

See also 10.3

## 10.2.10 Choice

A choice value is encoded with a preceding tag, which indicates which alternative has been selected. A tag is encoded as an integer value in range 0..(number of alternatives-1).

Example: Encoding for choice type in 9.3.12

```
<VariantRecord> ::=
{
  00 <flag : <Flag>>
  01 <counter : <Counter>>
  10 <extEnum : <ExtendedEnum>>
}
;
```

See also 10.3

## 10.2.11 Restricted character strings

A character string is mapped to a string of octets (or double octets in case of BMPString). If the number of characters may vary then a length field precedes the character string.

A length field is encoded as an integer field of type INTEGER (lb..ub) where "lb" is the lower bound the of size constraint and "ub" is the upper bound.

Example: Encodings for character string types in 9.3.13

```
<FixedStr> ::=
  <Char>(10)
;

-- length = 1..10, e = length-1 = 0..9
<VarStr> ::=
  <length : <INTEGER(4)> exclude {1010|1011|1100|1101|1110|1111}>
  <Char>*val(length)
;

<FixedWStr> ::=
  <WChar>(10)
;

-- length = 1..10, e = length-1 = 0..9
<VarWStr> ::=
  <length : <INTEGER(4)> exclude {1010|1011|1100|1101|1110|1111}>
  <WChar>*val(length)
;

<Char> ::=
  bit(8)
;

<WChar> ::=
  bit(16)
;
```

## 10.3 Specialised encoding

Specialised encoding can be specified only if the default encoding is specified in CSN.1. If standard ASN.1 encoding rules (BER and PER) are used then specialised encoding definitions have no effect on encoding.

### 10.3.1 General notation

There are three alternatives for specification of specialised encoding:

1. Definition is within ASN.1 definition in a user-defined constraint. The constraint is of the form

*Encoding(<special encoding>)*

2. Definition is stand-alone and there is a reference to it within ASN.1 definition in a user-defined constraint:

*Encoding(<reference to specialised encoding>)*

3. Definition is stand-alone and there is a reference to the corresponding ASN.1 definition

Specialised encoding is defined in CSN.1.

Example of specialised encoding, specialisation within ASN.1 definition:

```
B ::= BOOLEAN
      (CONSTRAINED BY {-- Encoding(<B> ::= 0|1;--})
```

Example of specialised encoding, reference to specialisation within ASN.1 definition:

```
B ::= BOOLEAN
      (CONSTRAINED BY {-- Encoding(specialisation in the clause 2.3.4.5)--})
```

```
<B> ::= 0|1;
```

Example of specialised encoding, reference to message contents description ~~abstract syntax~~ within CSN.1 definition:

```
B ::= BOOLEAN
```

```
-- Specialisation for type B specified in the clause 1.2.3.4
<B> ::= 0|1;
```

The specialised encoding shall be such that all the values of a type can be represented with it, i.e. there shall be a mapping from each abstract value to an encoded value.

Example: An integer value set is not continuous but it is evenly distributed.

```
SparseEvenlyDistributedValueSet ::= INTEGER (0|2|4|6|8|10|12|14)
      (CONSTRAINED BY {
        -- Encoding(
        --   <SparseEvenlyDistributedValueSet> ::=
        --     <INTEGER(3)>
        --   ;
        -- )
        -- Mapping: e = n/2
      }
```

Example: An integer value set is not continuous and evenly distributed.

```

SparseValueSet ::= INTEGER (0|3|5|6|8|11)
  (CONSTRAINED BY {
    -- Encoding(
    -- <SparseValueSet> ::=
    --   000|001|010|011|100|101
    -- ;
    -- )
    -- 0 => 000, 3 => 001, etc.
  })

```

Example: A list type is encoded using more bits instead of explicit length.

```

VariableLengthList ::= SEQUENCE (SIZE (0..10))
  (CONSTRAINED BY {
    -- Encoding(
    -- <VariableLengthList> ::=
    --   { 1 <Status> }(*)
    --   0
    -- ;
    -- )
  })
  OF Status

```

Example: Some alternatives of a choice type are used more frequently as others. Therefore the tags for the frequently used alternatives are specified to be shorter than others.

```

VariantRecord ::= CHOICE {
  flag      Flag,           -- The two first alternatives are mostly used
  counter   Counter,
  extEnum   ExtendedEnum,
  status    Status,
  list      VariableLengthList
}
(CONSTRAINED BY {
  -- Encoding(
  -- <VariantRecord> ::=
  --   { 00 <flag> : <Flag>>
  --     / 01 <counter> : <Counter>>
  --     / 100 <extEnum> : <ExtendedEnum>>
  --     / 101 <status> : <Status>>
  --     / 110 <List> : <VariableLengthList>>
  --   }
  -- ;
})

```

### 10.3.2 Shorthand notation

Some specialised encodings can be specified using a shorthand notation.

If an integer value set is not continuous then the encoding can be compressed by specifying the following shorthand:

*Encoding(compressed)*

Let there be  $m$  values in a value set. For each value  $n_i$ ,  $n_i < n_{i+1}$ ,  $0 = i < m$ . Value  $n_i$  is encoded as value  $i$  of type INTEGER (0..m-1).

Example: A value set is not continuous but it is evenly distributed.

```
SparseEvenlyDistributedValueSet ::= INTEGER (0|2|4|6|8|10|12|14)
    (CONSTRAINED BY {-- Encoding(compressed) --})

-- Value is encoded as INTEGER (0..7)
-- 0 => 000, 2 => 001, 4 => 010, 6 => 011, 8 => 100, 10 => 101,
-- 12 => 110, 14 => 111
```

Example: A value set is not continuous and evenly distributed.

```
SparseValueSet ::= INTEGER (0|3|5|6|8|11)
    (CONSTRAINED BY {-- Encoding(compressed) --})

-- Value is encoded as INTEGER (0..5)
-- 0 => 000, 3 => 001, 5 => 010, 6 => 011, 8 => 100, 11 => 101
```

A list type can be encoded using more bits instead of explicit length indicator by specifying the following shorthand:

*Encoding(morebit)*

Example: A list type is encoded using more bits instead of explicit length.

```
VariableLengthList ::= SEQUENCE (SIZE (0..10))
    (CONSTRAINED BY { -- Encoding(morebit) --})
    OF Status

-- Value is encoded as:
-- <VariableLengthList> ::=
--   { 1 <Status> }(*)
--   0
-- ;
--
```

More shorthand notations are for FFS.



## History

Document history		
V.0.0.0	January 1999	Starting point based on UMTS YY.40 V0.1.0
V.0.0.1	January 1999	version after TSG RAN WG2 #1 based on the presentation of Tdoc TSG RAN WG2 018/99.
V.0.0.2	March 1999	3GPP template included Addition of all accepted in TSG RAN WG2#2, <i>TSGR2#2(99)087 (chapter 10 and 11)</i> Addition of a sentence in §6 about using descriptive SDL and removing the other sentence dealing with PEX group
<u>V.0.1.0</u>	<u>April 1999</u>	<u>All modifications included in V.0.0.2 approved by TSG RAN WG2#3</u>
V1.0.0	April 1999	Version seen for information at TSG RAN#3
V1.1.0	July 1999	Proposals contained in document TSGR2#5(99)669 included with some sentences removed as agreed in RAN2#5  Description of tabular format contained in TSGR25(99)524 included as a new chapter 8 named "Usage of tabular format" as agreed in RAN2#5
Rapporteur for UMTS 25.921 is:		
Jean Dumazy Philips Consumer Communications  Tel. : +33 (0)2 43 18 48 08 Fax : +33 (0)2 43 41 18 18 Email : jean.dumazy@dev-lme.pcc.philips.com		
This document is written in Microsoft Word version 6.0/95.		