3GPP TR 25.921 V4.8.0 (2004-06)

Technical Report

3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Guidelines and principles for protocol description and error handling (Release 4)

The present document has been developed within the 3rd Generation Partnership Project (3GPP ™) and may be further elaborated for the purposes of 3GPP.

The present document has not been subject to any approval process by the 3GPP Organisational Partners and shall not be implemented.
This Specification is provided for future development work within 3GPP only. The Organisational Partners accept no liability for any use of this Specification.
Specifications and reports for implementation of the 3GPP ™ system should be obtained via the 3GPP Organisational Partners' Publications Offices.

Keywords
UMTS, radio

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

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© 2004, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TTA, TTC).
All rights reserved.

Contents

Foreword

This Technical Report (TR) has been produced by the 3rd Generation Partnership Project (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 the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater 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 document.

1 Scope

The present document provides a guideline for protocol specification of UMTS stage 2 and 3 including the usage of formal languages and rules for error handling. The present document covers control-plane and user-plane protocols specified in TSG-RAN such as RRC, RLC, RANAP, RNSAP, NBAP and SABP.

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. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] ITU-T Recommendation X.680: "Information technology - Abstract Syntax Notation One (ASN.1): Specification of the basic notation".

[2] ITU-T Recommendation X.681: "Information technology - Abstract Syntax Notation One (ASN.1): Information object specification".

[3] ITU-T Recommendation X.682: "Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification".

[4] ITU-T Recommendation X.690: "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

[5] ITU-T Recommendation X.691: "Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)".

[6] CSN.1: "specification, version 2.0".

[7] ITU-T Recommendation Z.100: "Specification and description language (SDL)".

[8] ITU-T Recommendation Z.105: "SDL Combined with ASN.1 modules (SDL/ASN.1)".

[9] ITU-T Recommendation Z.120: "Message Sequence Chart (MSC)".

[10] ISO/IEC 9646-3: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)".

[11] 3GPP TR 21.801 (Release 4): "Specification drafting rules".

[12] ETSI EG 202 106 (DEG/MTS-00050): "Methods for Testing and Specification (MTS); Guidelines for the use of formal SDL as a descriptive tool".

[13] 3GPP TS 24.008: "Mobile radio interface layer 3 specification; Core Network Protocols; Stage 3".

3 Void

4 Principles to ensure compatibility

4.1 Introduction

The rules intend to prevent incompatibilities between several phases of UMTS evolution (in analogy to what happened from GSM phase 1 to GSM phase 2).

4.2 Level 1 of principles: Protocol level

It shall be possible to inter-work different versions of any protocol specification.

An unknown protocol shall not cause problems to any entity that terminates the protocol.

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

The protocols shall specify a mechanism such that new message types shall be able to be introduced without causing any unexpected behaviour or damage.

The protocols may define a mechanism that allows a different behaviour, on received messages that are not understood, 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.

4.3.2 Partial decoding

PDU extensions are allowed in a compatible way, e.g. by utilising partial decoding or other mechanisms that allows the decoder to skip the extensions. Partial decoding means that a PDU can be decoded in parts. One part forms a complete value that can be separated from other parts.

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 Void

4.4.3 Adding mandatory IE

For backward compatibility reasons, addition of mandatory IE shall be avoided.

4.4.4 Absent optional IE

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

4.4.5 Comprehension required

"Comprehension required" requirement can be associated with an IE or a message. It means that the IE or message is tagged with "criticality" information (explicit in the message or implicit based on the type of IE or message). Any action performed by the receiver if the IE or message is not understood ("comprehended") is based on this "criticality" information.

4.4.6 Partial Decoding

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

4.5 Level 4 of principles: Values level

4.5.1 Spare values and spare fields

In case the protocol allows for the transfer of spare information element values, the behaviour of the receiving entity not comprehending these values shall be specified.

In case the protocol allows for the transfer of spare fields, the behaviour of the receiving entity not comprehending the spare fields shall be specified. Facilities may be needed to allow receivers that do not comprehend the spare fields to decode the other message parts.

4.5.2 Unspecified values

In case the protocol allows for the transfer of undefined information element values, the behaviour of the receiving entity not comprehending these values shall be specified.

4.5.3 Void

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 of MSCs as one of the formal methods.

MSCs are 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) [12] as closely as possible.

The groups themselves should decide how SDL is used.

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

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

In this version the specifications shall not use SDL for the normative part of the specifications.

7 Protocol procedure specification rules

7.1 General

- A protocol specification shall contain a 'Procedures' clause, which specifies the functional behaviour, using "procedures". A procedure is typically a sequence of events, with a start and an end, which can be observed in the protocol and/or in the interfaces to other layers (upper and/or lower layers).
- The procedure specification shall be made using text and verbal forms.
- The verbal forms, such as "shall", "should" and "may" are used in conformance with [11] Annex E.
- The procedures should be specified in an asymmetric way, by concentrating on the behaviour on one side of the interface. As guidance, the "controlled" side, rather than the "controlling" side of the interface should be specified.
- The procedures should be specified using the externally observable behaviour, to ease writing of test specifications.
- All normal cases shall be covered. Normal cases are straightforward cases, branches of procedures and combinations of procedures.
- All error cases shall be specified, either explicitly or implicitly. The error cases are all cases that are not considered as normal cases. The error handling should be divided between error handling global to the protocol layer and procedure specific error handling. The procedure specific error cases should be put after the normal cases in each procedure.
- Redundancy/duplication shall be avoided, in order to avoid problems with later CR, even if this makes the specification initially less readable.
- States and state variables should be used when it provides unambiguity, a way to describe nested procedures and colliding cases.
- Timers, variables and constants and usage of them must be specified.
- Explicit explanation when the action shall be performed is specified in the procedure itself.
- When there are procedural differences between the FDD and TDD modes these should be clearly pointed out using a consequent notation, e.g. "FDD only", "In TDD, …".
- When optional IEs are possible in a given message, the meaning of the presence (i.e.: which "function" are activated with the given IE) shall be specified in the procedure for the receiving entity. The requirements on when to include a given IE shall be specified for the transmitting entity. An exception for this rule is when the requirements on the entity is not specified by the protocol.
- Requirements on the content of a message at the sending entity are put before analysis of the message at the receiving entity.
- References to IEs that are parts of another IE is allowed.
- When referring to an IE a formal notation shall be used.
- When referring to a message, a formal notation shall be used.

7.1a Specification of algorithms and formulas

When algorithms or formulas are used in the specifications, a formal notation shall be used and mathematical expressions should be used to reduce ambiguity.

- The notations "OP1 div OP2" and "OP1 mod OP2" shall be interpreted according to the following:
  - OP1 = OP2 * A + B;
  - 0 =< B =< OP2-1;
  - OP1 div OP2 corresponds to A;
  - OP1 mod OP2 corresponds to B.

7.2 RRC specific rules

- The specification shall focus on the UE behaviour.
- Only UE timers are normative (when UTRAN timers are present, it is for information).
- The procedure specification text shall specify how the UE shall handle the IEs.
- As much as possible of the UE behaviour shall be tied to reception and non-reception of IEs and included in the subclause "Generic actions upon receipt and absence of an information element", to avoid duplication of text.
- "UTRAN shall" shall be only used when UTRAN behaviour is normative.
- It shall be specified whether timers shall be started when RRC sends the message to lower layers or when the message is effectively sent at the radio interface.
- When referring to messages in the procedure text, the notation "EXAMPLE message" is used (excluding the quotation marks). For example: "The UE shall transmit an RRC CONNECTION REQUEST message".
- When referring to IEs in the procedure text, the tabular description of IEs should be used as basis. The notation "IE "Example"" is used (including the inner but not the outer quotation marks). Values of IEs are put within quotation marks. For example: "The UE set the IE "Protocol error indicator" to "FALSE" in the RRC CONNECTION REQUEST message".
- UE performance requirements are considered to be TSG RAN WG2 work. These must be specified only if they are testable.
- The following rules apply to the procedure text:
  - "n>" shall be used as an indentation level specifier, where, n denotes the level of indentation, n = 1….10.

7.3 RLC specific rules

- The behaviour of an RLC entity is specified by means of one or more elementary procedures. Each elementary procedure involves a Sender and a Receiver. In this respect the Sender is the RLC protocol entity that initiates the procedure, while the Receiver is the peer entity of the Sender. For each elementary procedure there are two possible configurations: one in which the UE is the Sender and UTRAN is the Receiver and another one in which UTRAN is the Sender and the UE is the Receiver.
- An RLC entity is normally configured to be either Sender or Receiver. In some cases however e.g. the AM RLC entity, the RLC entity consists of a transmitting and a receiving side. Such RLC entities can be Sender for some elementary procedures as well Receiver for other elementary procedure(s).
- The procedure specifications shall be specified independently as elementary procedures. Inter- relations between elementary procedures may be described in a general subclause describing the initiation of the procedure.
- The procedure specification text shall specify both how the Sender shall set the PDU parameters and how the Receiver shall handle them.

- It shall be specified when the timers are started, for example whether a timer shall be started when RLC submits a PDU to lower layers or when the successful or unsuccessful transmission of the PDU on the radio interface is indicated by lower layer.

- When referring to messages in the procedure text, the notation "EXAMPLE PDU" is used (excluding the quotation marks). For example: "The receiver shall transmit a RESET ACK PDU".

- When referring to PDU parameters in the procedure text, double quotation marks are used. The notation "PDU parameter "Example"" illustrates this usage (including the inner but not the outer quotation marks). Values of PDU parameters are also put within double quotation marks. For example: "If the PDU parameter "Length indicator" equals "0000000" the Receiver shall...".

- When referring to configuration parameters in the procedure text, double quotation marks are used. The notation "parameter "Example" is configured as" illustrates this usage (including the inner but not the outer quotation marks). Values of configuration parameters are also put within double quotation marks. For example: "If the Sender has the parameter "Delivery of erroneous PDUs" configured for "No Detect" …".

- When referring to state variables in the procedure text, the notation "VT(EXAMPLE)" and "VR(EXAMPLE)" is used (without the quotation marks) for state variables used in the RLC transmitter and RLC receiver respectively. For example: "If the value of VT(DAT) is equal to...".

- When referring to timers in the procedure text, the notation "Timer_Example" is used (without the quotation marks). For example: "If the timer Timer_Poll_Prohibit has expired the sender shall...".

- When referring to RLC protocol states in the procedure text, the notation "EXAMPLE_NAME state" is used (without the quotation marks). For example: "If the sender is in RESET_PENDING state, the sender shall...".

- Cross-references to other clauses are referred to as "clauses", subclauses are referred to as "subclauses". For example: "see subclause 9.3".

- Procedures which include conditional behaviour use the convention "if" … "else if" … "otherwise" to specify each condition.

8 Message specification

8.1 Void

8.2 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 ("abstract syntax"); and

- a message encoding, which defines the encoded messages (i.e. what is carried as a bit string, "transfer syntax").

8.3 Logical description

The logical description of messages shall be done using tabular format specified in clause 9 of the present document, Message contents description.

8.4 Message contents description

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

8.5 Compilability of the transfer syntax

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

8.6 Efficiency/Compactness

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

8.7 Evolvability/Extensibility

The message contents description shall allow the evolution of the protocol.

8.8 Inter IE dependency

The message contents description 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.

8.9 Intra IE dependency

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

8.10 Support of error handling

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

9 Usage of tabular format

A protocol specification should include a 'Tabular description' subclause, including:

- a message description subclause;
- an IE description subclause.

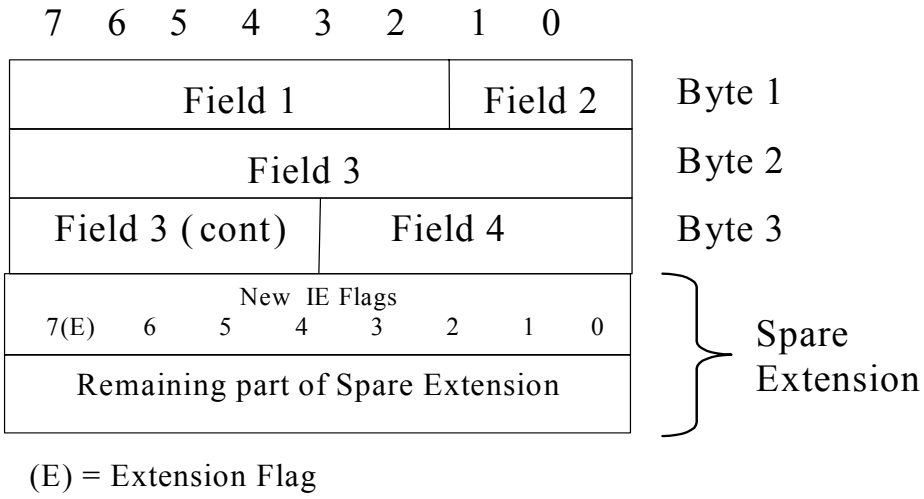
9a Usage of Iub/Iur Frame Protocol

The following clauses contain guidelines for specification of frame protocols.

9a.1 Extensions for future releases in Data Frame

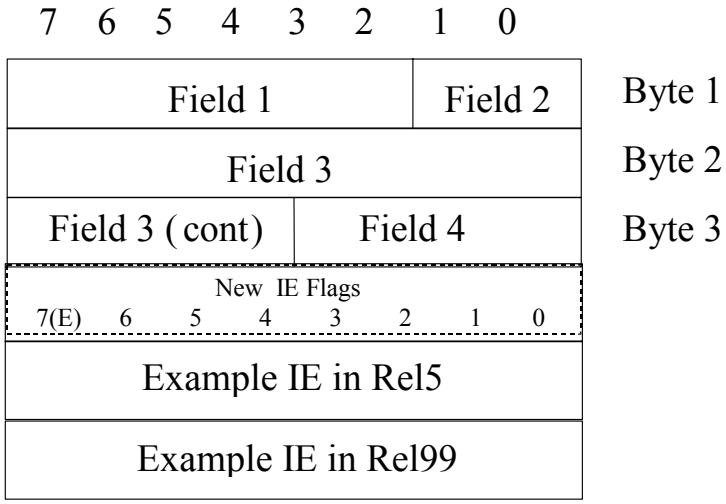
Spare Extension is used to define New IEs in the future. When the first IE is added in the Spare Extension in the Data Frame, *New IE Flags* IE shall be added in the first byte of the Spare Extension to indicate the validity of the value of the IEs in the Spare Extension. The last bit position of the *New IE Flags* IE is used as the Extension Flag to allow the

extension of the *New IE Flags* IE in the future. The IEs in the Spare Extension will be added in the order in which the IEs are introduced regardless of the release.



In the example below, it is assumed that after *Example IE in Rel5* IE was introduced, *Example IE in Rel99* IE is introduced.

In this example, New IE Flags (0) indicates the validity of *Example IE in Rel5* IE and the New IE Flags (1) indicates the validity of *Example IE in Rel99* IE. The IEs are added in the order of their introduction in the Spare Extension. For the Rel99 and Rel4 nodes, New IE Flags (0) and *Example IE in Rel5* IE will always be seen as Spare Bits while for Rel5 nodes, all the IEs (i.e., New IE Flags (0), New IE Flags(1)), *Example IE in Rel5* IE and *Example IE in Rel99* IE can be used.



Example of Spare Extension Usage

9.1 Tabular description of messages and IEs in RRC

9.1.1 Message description

A "Message description" subclause includes one subclause 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.

9.1.1.1 The general description

9.1.1.2 The Information Element table

The table is composed of 6 columns, labelled and presented as shown below.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version

NOTE: Indentations are used to visualise the embedding level of an "IE/Group".

Indentations are explicitly written with the character ">", one per level of indentation. Indentations of lines can be found in the IE/Group Name column.

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 Need and Multi columns are meaningless and should be left empty.

9.1.1.2.0 IE/Group Name column

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 to refer to the IE or the group of IEs in the procedure specification described in clause 7.

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

9.1.1.2.1 Need and multiplicity (Multi) columns

These columns provide most of the information about the presence, absence and number of instances 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.

The meaning of the 'need' column is summarised below:

MP Mandatorily present.

A value for that information is always needed, and no information is provided about a particular default value. If ever the transfer syntax allows absence (e.g.: due to extension), then absence leads to an error diagnosis.

MD Mandatory with default value.

A value for that information is always needed, and a particular default value is mentioned (in the "Semantical information" column). This opens the possibility for the transfer syntax to use absence or a special pattern to encode the default value.

CV Conditional on value.

The need for a value for that information depends on the value of some other IE or IEs, and/or on the message flow (e.g.: channel, SAP). The need is specified by means of a condition, which result may be that the information is mandatorily present, mandatory with default value, not needed or optional.

If one of the results of the condition is that the information is mandatorily present, the transfer syntax must allow for the presence of the information. If in this case the information is absent an error is diagnosed.

If one of the results of the condition is that the information is mandatory with default value, and a particular default value is mentioned (in the "Semantical information" column), the transfer syntax may use absence or a special pattern to encode the default value.

If one of the results of the condition is that the information is not needed, the transfer syntax must allow encoding the absence. If in this case the information is present, it will be ignored. In specific cases however, an error may be diagnosed instead.

If one of the results of the condition is that the information is optional, the transfer syntax must allow for the presence of the information. In this case, neither absence nor presence of the information leads to an error diagnosis.

CH Conditional on history.

The need for a value for that information depends on information obtained in the past (e.g.: from messages received in the past from the other party). The need is specified by means of a condition, which result may be that the information is mandatorily present, mandatory with default value, not needed or optional.

The handling of the conditions is the same as described for CV.

OP Optional.

The presence or absence is significant and modifies the behaviour of the receiver. However whether the information is present or not does not lead to an error diagnosis.

9.1.1.2.1.1 Mandatory

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Name	MP				
Name	MD			(default value is indicated)	

The multiplicity column may be left empty (see subclause 9.1.1.2.1.5).

For mandatory IEs, the rules are as follows, applied on the number of instances given by the multiplicity column (leaving the multiplicity column empty means one and only one instance).

For an IE not belonging to a group MP indicates that the number of instances as given by the multiplicity column of 'Name IE' is necessary in the message.

For a group not belonging to another group, MP means that the number of instances as given by the multiplicity column of the 'Name group' is necessary in the message.

For an IE or a group belonging to another group, MP means that if the parent group is present, then the number of instances as given by the multiplicity column of the 'Name group' or 'Name IE' is necessary in the embedding group.

For an IE not belonging to a group MD indicates that the number of instances as given by the multiplicity column for information 'Name IE' is necessary in the message, and that a special value (the default value) exists, for all instances or individual instances, and is mentioned in the 'Semantics description' column.

For a group not belonging to another group, MD means that the number of instances as given by the multiplicity column for information structure 'Name group' is necessary in the message, and that a special value (the default value) exists, for all instances or individual instances, and is mentioned in the 'Semantics description' column.

For an IE or a group belonging to another group, MD means that if the parent group is present, then the number of instances as given by the multiplicity column for information structure 'Name group' or information 'Name IE' is necessary in the embedding group, and that a special value (the default value) exists and is mentioned in the 'Semantics description' column.

The default value might be fixed by the standard, or conditional to the value of some other IE or IEs, or conditional on information obtained in the past. In case the default value depends on information obtained in the past, variables may be used to specify it.

9.1.1.2.1.2 Optional

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Name	OP				

The multiplicity column may be empty (see subclause 9.1.1.2.1.5).

This indicates that the number of instances as given by the multiplicity column of the 'Name IE' or 'Name group' is not necessary in the message or the embedding group, and that the sender can choose not to include it.

9.1.1.2.1.3 Conditional

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
	CV *cond*				
	CH				

The multiplicity column may be empty (see subclause 9.1.1.2.1.5).

CV indicates that the requirement for presence or absence of the number of instances as given by the multiplicity column 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). In the CV case, 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. In the CH case, the condition is described in the procedural section.

The expression *cond* should explicitly cover all cases. A typical *cond* expression is "The IE is *CS1* if *TE* and *CS2* otherwise", where *CS1* and *CS1* are condition statements and *TE* is an expression used in the if test. For example: "The IE is mandatory if the IE "XXX" has the value "YYY" and not needed otherwise".

The result of evaluating the condition (if the condition is met or not) may mean that the IE is:

- mandatorily present, when the condition statement says "mandatory";
- mandatory with default value, when the condition statement says "mandatory with default value";
- not needed, when the condition statement says "not needed";
- optional, when the condition statement says "optional".

The error handling shall be specified in the protocol for the cases when the requirement for presence or absence of an IE indicated by the condition is not followed.

9.1.1.2.1.4 Choice

This is particular group of at least two children.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Choice *name*	NOTE 1				
>Name1					
>Name2					

NOTE: The Need column shall take one of the values "MP", "MD", "OP", "CV *cond*" or "CH *cond*".

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

The Need column shall and the Multi columns may be filled 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 (depending on the value of some other IE or IEs, and/or on the message flow) exist for the choice, they are explained in an explanatory clause.

9.1.1.2.1.5 Sets

In general, this indicates that more than one instance of an IE/Group might be necessary in the message.

The two lines below indicate different allowed alternatives.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Name	MP	nn..pp			
Name	MP	nn..indefinite			
Name	MP	nn..sym2			
Name	MP	sym1..pp			
Name	MP	sym1..sym2			
Name	MD	nn..pp			
Name	MD	nn..indefinite			
Name	MD	nn..sym2			
Name	MD	sym1..pp			
Name	MD	sym1..sym2			
Name	OP	nn..pp			
Name	OP	nn..indefinite			
Name	OP	nn..sym2			
Name	OP	sym1..pp			
Name	OP	sym1..sym2			
Name	Cx cond	nn ..pp			
Name	Cx cond	nn..indefinite			
Name	Cx cond	nn..sym2			
Name	Cx cond	sym1..pp			
Name	Cx cond	sym1..sym2			

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

The notation '..' can be replaced with the same meaning by 'to'.

This indicates that a number of instances of the IE/Group are necessary in the message/embedding group. The order is significant. The reference should use the bracket notation (e.g.: 'Name[1] IE') to refer to a specific instance; numbering starts by 1.

The *nn..pp* case indicates that the number of instances is between nn and pp, inclusively. This means that *nn* instances are necessary in the message, that additional *pp-nn* instances are optional and meaningful, and that instances after the *pp*th are not necessary.

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 instead the Multi column should be left empty to indicate one and only one instance. The 0..1 case combined with MP should be avoided and replaced by an OP indication with the Multi column left empty.

The *nn*..indefinite case indicates that the number of instances is *nn* or greater. This means that *nn* instances are necessary in the message, and that additional instances are optional and meaningful. The number *nn* is positive or null. It is however allowed that the transfer syntax puts some practical limits on the maximum number of instances.

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 'Need' column is set to MP, MD, CV or CH and interpreted as described in subclauses 9.1.1.2.1.1 to 9.1.1.2.1.3 applied to the whole set.

9.1.1.2.2 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 subclause in the Information Element Description clause in the same document; typically the subclause number and titles are given, and if possible this should be a hypertext link;

b) a reference to another document, and to a subclause in the Information Element Description clause in the indicated document; typically only the subclause title is indicated.

9.1.1.2.3 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.

9.1.1.2.4 Expressing differences between FDD and TDD modes

If a PDU or a structured information element contain information elements whose Need value is different for FDD and TDD modes or if a certain structured information element is completely different for the two modes, a choice group should be used.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Choice mode	MP				
>FDD					
>>element1	MP				
>>element2	OP				
>TDD					
>>element3	OP				
>>element4	MP				

9.1.1.2.5 Version column

When an information element is added from one version to a latter one, the version in which the element is added (e.g.: REL-4, REL-5) is included in the version column.

When a new CHOICE group is added from one version to a later one, the version in which the group is added is included in the version column of all new rows. If some of the information elements in the new CHOICE group were included in the older version (but not inside a CHOICE group), the version column is not updated for those information elements (see also the example at the end of this clause).

When an existing CHOICE group is extended from one version to a later one to include more options, the version in which the new options are added is included in the version column of the rows describing the new options.

When the type of an information element is modified from one version to a later one to include more values, a new subrow is created in which the new values are added. The version in which the modification takes place is included in the version column of the new subrow.

When the multiplicity of an IE is extended from one version to another, a new subrow is created in which the extended multiplicity values are included. The version in which the modification takes place is included in the version column of the new subrow.

The example below shows how the version column is used for the cases described above. The first table shows an example of a Release '99 table.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Element1-99	MP		Type1		
Element2-99	MP		Type2		
CHOICE choice1-99	MP				
>first					
>>Element3-99	MP		Type3		
>second					
>>Element4-99	MP		Type4		
Element5-99	MP		Enumerated(a,b)		

The second table shows extensions of the above table in Release 5, and how the REL-4 and REL-5 shall be included in the version column.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Element1-99	MP		Type1		
Element6-r4	MP		Type6		REL-4
CHOICE choice2-r4	MP				REL-4
>old					REL-4
>>Element2-99	MP		Type2		
>new					REL-4
>>Element7-r4	MP		Type7		REL-4
CHOICE choice1-99	MP				
>first					
>>Element3-99	MP		Type3		
>second					
>>Element4-99	MP		Type4		
>third					REL-4
>>Element8-r4	MP		Type8		REL-4
Element5-99	MP		Enumerated(a,b,		
		1 to 2			REL-4
			c)		REL-5

9.1.1.3 Explanatory clauses

This includes the subclauses needed to elaborate conditions. There must be one explanatory clause for each named condition. The text must give the information sufficient to decide whether the IE/group is to be included or not.

9.1.2 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. A type should as far as possible not be defined in multiple places in a specification.

An 'IE description' subclause includes one subclause per IE type.

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

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version

The different columns are filled as message description columns are filled with the addition that in the IE Type and reference column also the use of basic types defined in subclause 9.3 in the present document is allowed. These basic types shall be considered as pre-defined and for those no reference is necessary. For IE type descriptions, explanatory clauses should also be used as described in subclause 9.1.1.3.

9.1.3 Extension for further releases

9.1.3.1 Basic principle

Added elements or choice branches are included where they fit most naturally according to their semantics, independently from the version in which they were added.

9.1.3.2 Critical or non-critical

A case-by-case guideline is provided by informal text after the table itself.

Spare values shall not be defined for critical information elements. Extension of the value range of critical information elements must be done by creating a critical extension.

Spare values may be defined for non-critical information elements. For spares that result after mapping of the original information element value the same handling and/or guidelines apply.

In case spares are defined, the tabular notation should indicate the number of spare values that is needed. Within the ASN.1 spare values should only be used to increase the encoded size of an IE. This means that the ASN.1 should only include spares if the number of spare values that is needed exceeds the number of undefined values within the transfer syntax of the information element.

For downlink messages, spare values may be defined for non-critical information elements for which the need is specified to be MD or OP (or CV case leading to MD or OP). In this case, a receiver not comprehending the received a spare value shall consider the information element to have the default value or consider it to be absent respectively.

For uplink messages spares values may be defined for all information elements, including those for which the need is specified to be MP (or CV case leading to MP).

In all cases at most one spare should be defined for choices.

9.1.3.3 Topics left unresolved

Other extensions like removing a component, changing the 'Needed' status of a component, changing the 'multiplicity' status of a component (i.e. extending or reducing the range), adding or removing values to an enumerated, extending or reducing the range of a bit or octet string, extending or reducing the range of an integer are FFS.

Whether, and if so how, the tables indicate the critical or non-critical status of the addition in the coding requires FS. One issue is that for an extension within sub-structures (i.e. not a message structure) the status may differ from one message to another.

9.1a Tabular description of messages and IEs in RANAP, RNSAP, NBAP, and SABP

9.1a.1 Message description

A 'Message description' subclause includes one subclause per message.

A message is described with, in this order:

- a table describing a list of information elements;
- explanatory clauses, mainly for describing textually conditions for presence or absence and range bounds for some IEs/IE groups.

9.1a.1.1 The Information Element table

The table used in RANAP, RNSAP, NBAP, and SABP is composed of 7 columns, labelled and presented as shown below.

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality

NOTE: Indentations are used to visualise the embedding level of an "IE/Group".

Indentations are explicitly written with the character ">" as well as by use of ruler indentations, one per level of indentation. Indentations of lines can be found in the IE/Group Name column.

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

9.1a.1.1.1 IE/Group Name column

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 to refer to the IE or the group of IEs in the procedure specification as described in clause 7.

The name of an IE group shall be given in bold font.

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

9.1a.1.1.2 Presence and Range columns

These columns provide most of the information about the presence, absence and number of instance 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.

At least one of the Presence and Range columns shall be filled.

The meaning of the Presence column is summarised below:

M Mandatorily present.

A value for that information is always needed, and no information is provided about a particular default value.

C Conditional.

The IE/IE group is required to be present when a condition is met that can be evaluated on the sole basis of the content of the message. If the condition is not met, the IE/IE group shall not be included.

O Optional.

The presence or absence is significant and modifies the behaviour of the receiver.

9.1a.1.1.2.1 Mandatory

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality
Name	M					
Name		1				

For an IE M indicates that one and only one instance of *Name* IE shall be present in that part of the message.

For an IE group 1 in the Range column indicates that one and only one instance of *Name* IE group shall be present in that part of the message.

9.1a.1.1.2.2 Optional

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality
Name	O					
Name		0..1				

For an IE O indicates that one and only one instance of *Name* IE may be present in that part of the message and that the sender can choose not to include it.

For an IE group 0..1 in the Range column indicates that one and only one instance of *Name* IE group may be present in that part of the message and that the sender can choose not to include it.

9.1a.1.1.2.3 Conditional

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality
Name	C - *cond*					
Name	C - *cond*	nn..<sym>				

For an IE/IE group C indicates that the requirement for presence or absence of the IE/IE group depends on a condition 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.

The result of evaluating the condition (if the condition is met or not) may mean that the IE is:

- mandatorily present, where nn is giving the minimum number of instances that shall be present and "sym" is a symbolic name giving the maximum number of instances than may be present.
- mandatorily absent.
- optional, where nn is giving the minimum number of instances and "sym" is a symbolic name giving the maximum number of instances than may be present.

9.1a.1.1.2.4 Choice

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality
Choice *name*						
>Name1						
>Name2						

A 'choice' is distinguished from IEs/IE groups by the use of 'choice' as first word in the name.

The Presence columns are filled normally for the group line (Choice *name*). They are not filled for the choice tags, e.g.: "*Name1*".

9.1a.1.1.2.5 Sets

In general, this indicates that more than one instance of an IE/IE group may be present in the message.

The two lines below indicate different allowed alternatives.

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality
Name		nn..pp				
Name		nn..sym				

Where *nn* and *pp* stand for positive integers (non zero) and *sym* for a symbolic name. The number pp must be greater than nn.

The use of a symbolic name for the upper range bound indicates that the value is given in a textual clause.

The notation '..' can be replaced with the same meaning by 'to'.

This indicates that a number of instances of the IE/IE group shall be present in the message/embedding IE group. The order is significant.

The *nn..pp* case indicates that the number of instances is between nn and pp, inclusively. This means that *nn* instances shall be present in the message, that up to *pp* instances may be present.

The *nn..sym* case indicates that the number of instances is between nn and the value represented by "sym", inclusively. This means that *nn* instances shall be present in the message, that up to *sym* instances may be present.

9.1a.1.1.3 IE Type and reference column

This column is not filled for IE 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 subclause in the "Information Element Description" clause in the same document; typically the subclause number and titles are given, and if possible this should be a hypertext link. Titles need only be given if the name of the type is different from the name of the IE;

b) a reference to another document, and to a subclause in the Information Element Description clause in the indicated document; typically only the subclause title is indicated.

9.1a.1.1.4 Semantics description column

Filling this column is optional. It should be use to clarify the meaning of the IE/IE group.

9.1a.1.1.5 Expressing differences between FDD and TDD modes

Differences between FDD and TDD can be expressed either by separate tabular description of the messages or by comments in the semantics description column. The former alternative should be used for messages with major differences and the latter for messages with minor differences between FDD and TDD.

9.1a.1.1.6 Criticality column

Each IE or IE group may have criticality information applied to it. The following cases are possible.

–	No criticality information is applied explicitly.
YES	Criticality information is applied. 'YES' is usable only for non-repeatable information elements.
GLOBAL	The information element and all its repetitions together have one common criticality information. 'GLOBAL' is usable only for repeatable information elements.
EACH	Each repetition of the information element has its own criticality information. It is not allowed to assign different criticality values to the repetitions. 'EACH' is usable only for repeatable information elements.

9.1a.1.1.7 Assigned Criticality column

This column provides the actual criticality information as defined in subclause 10.3.2 in RANAP, RNSAP, NBAP, and SABP.

If an IE/IE group is not understood or missing, the receiving node shall take different actions depending on the value of the Criticality Information. The three possible values of the Criticality Information for an IE/IE group are:

1. Reject IE;
2. Ignore IE and Notify Sender;
3. Ignore IE.

9.1a.1.2 Explanatory clauses

This includes the subclauses 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/IE group is to be included or not, or the value of the symbolic name. The text shall be given in separate tables for the conditions and range bounds.

9.1a.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. A type should as far as possible not be defined in multiple places in a specification.

An 'IE description' subclause includes one subclause per IE type.

The description of an IE type is done as a table similar to that used for the description of messages. In RANAP, RNSAP, NBAP, and SABP this table has the layout as shown below.

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality

The different columns are filled as message description columns are filled with the addition that in the IE Type and reference column also the use of a basic types defined in subclause 9.2 in the present document is allowed. These basic types shall be considered as pre-defined and for those no reference is necessary. For IE type descriptions, explanatory clauses should also be used as described in subclause 9.1a.1.2. The inclusion of the criticality and assigned criticality columns is optional, but shall be included if separate criticality needs to be indicated.

9.1a.3 Extension for further releases

9.1a.3.1 Basic principle

Added elements or choice branches are included where they fit most naturally according to their semantics if the ASN.1 allows (e.g. there exist an extension container in this place). For further information on handling of extensions in RANAP, RNSAP, NBAP, and SABP see subclause 10.5.

9.2 Basic types

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

NOTE: The tabular description in this subclause used to describe different formats of the basic types follow the layout applicable to RRC. However, the basic types as such are applicable also to RANAP, RNSAP, NBAP, and SABP.

9.2.1 Enumerated

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
			Enumerated (*c1*, *c2*, *c3*)		
			Enumerated (x1..x*n*)		
			Enumerated (*c1*, *c2*, *c3, …*)		
			Enumerated (*c1*, *c2*, *c3, …, c4, c5*)		

In the first format, *c1*, *c2*, *c3* stands for a list of 2 or more symbolic names separated by commas.

In the second format, *x* is some character string, possibly empty, *n* is an integer, and indicates a list of *n* different values, with no particular property except for being distinct.

In the third format the IE value range is c1, c2, and c3 with an infinite extension possibility.

In the fourth format the IE value range is c1, c2, c3, c4, and c5 with an infinite extension possibility. The values c4 and c5 have been added in a backward compatible way, typically in a release later than Release '99.

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

9.2.2 Boolean

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
			Boolean		

This is shorthand for:

			Enumerated (*False, True*)		

The 'semantics description' column should in this case give the meaning of the two alternatives.

NOTE: Boolean should be preferably replaced by an enumerated with two values, with expressive names.

9.2.3 Integer

The type is indicated by the word 'Integer' followed possibly by a list of values or ranges between parentheses.

The different lines below indicate different alternatives.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
			Integer	unit indication	
			Integer (*nn..pp*)	unit indication	
			Integer (*nn*..indefinite)	unit indication	
			Integer (*sym1..pp*)	unit indication	
			Integer (*nn..sym2*)	unit indication	
			Integer (*sym1..sym2*)	unit indication	
			Integer (*b1..b2* by step of *st)*	unit indication	

This indicates some quantity of something, possibly limited to some range. This typically enters in computations, such as additions or other arithmetic. The unit should be indicated in the 'Semantics description' column when applicable.

Where *nn* and *pp* stand for positive, negative or null integers, and *sym1* and *sym2* for symbolic names.

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.

A step indication can be added to any of the range description, meaning that the values are $b1+k*st$, for all integral values of k such that $b1+k*st \leq b2$. The step *st* must be a positive non-null integer. When the step indication is not given, the default is a step of 1.

Some care should be applied not to present as Integer a field carrying a type of information which has nothing to do with integer, i.e. used in additions/subtractions, or as a discrete representation of a continuous data. If those conditions are not met, the bit string is to be preferred.

List of values or list of ranges separated by commas can also be used.

The word 'indefinite' can also appear as the upper bound of a range, or alone to indicate the infinity as a value. Examples are:

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
Some element	MP		Integer(0, 10, 20..25)	In dB	
TImer	MD		Integer(100..500 by step of 100, 1000, 2000, indefinite)	In ms, default is 100 Indefinite means that the timer needs not be started	

9.2.4 Bit string

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
			Bit string		
			Bit string (*nn*)		
			Bit string (*mm..pp*)		
			Bit string (*mm..pp, …*)		

Where *nn*, *mm*, and *pp* are positive non-null numbers indicating the fixed size, lower bound, and upper bound of the number of bits in the string respectively. In the fourth format the number of bits in the string is extensible beyond the upper bound. If no size is given the bit string can have any number of bits.

Bit strings are unstructured as seen by the protocol. They are typically transparent fields, used by other protocols (other layers or others systems), or as containers on which bit-per-bit boolean operations are done (e.g.: ciphered containers).

9.2.5 Octet string

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
			Octet string		
			Octet string (*nn*)		
			Octet string (*mm..pp*)		
			Octet string (*mm..pp, …*)		

Where *nn*, *mm*, and, *pp* are positive non-null numbers indicating the fixed size, lower bound, and upper bound of the number of octets in the string respectively. In the fourth format the number of octets in the string is extensible beyond the upper bound. If no size is given the octet string can have any number of bits.

This is just a shortcut for bit strings with a length a multiple of 8, and the same comments as on bit strings apply.

It should be noted that this does not indicate that the information is 'octet aligned', which is an encoding notion (and hence foreign to the tabular format) according to which in the transfer syntax a field starts at an octet boundary relatively to the beginning of the message (or other container).

9.2.6 Real

The type is indicated by the word 'Real' followed possibly by a list of values or ranges between parentheses.

The different lines below indicate different alternatives.

IE/Group Name	Need	Multi	Type and reference	Semantics description	Version
			Real (by step of *st*)	unit indication	
			Real (*nn..pp* by step of *st*)	unit indication	
			Real (*nn*..indefinite by step of *st*)	unit indication	
			Real (*sym1..pp* by step of *st*)	unit indication	
			Real (*nn..sym2* by step of *st*)	unit indication	
			Real (*sym1..sym2* by step of *st*)	unit indication	

This indicates some quantity of something, possibly limited to some range. This typically enters in computations, such as additions or other arithmetic. The unit must be indicated in the 'Semantics description' column when applicable.

Where *nn* and *pp* stand for positive, negative or null reals (typically expressed with a dot or by fractions), and *sym1* and *sym2* for symbolic names.

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.

The step indication means that the values are *b1+k*st*, for all integral values of k such that *b1+k*st* ≤ *b2*. The step *st* must be a positive non-null real.

List of values or list of ranges separated by commas can also be used.

The word 'indefinite' can also appear as the upper bound of a range, or alone to indicate the infinity as a value.

10 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 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 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 contents description 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**.

Identifiers that could be keywords of some language (e.g.: SDL, C, ASN.1, JAVA, C++, …) should be avoided.

In the current version of the ASN.1 specifications, user-defined constraints are not used.

RANAP, SABP, RNSAP and NBAP specifications refers to the 1997 versions of X.680, X.681 and X.691, but the protocols shall only make use of the feature set available in the 1994 versions of X.680, X.681 and X.691.

10.1 Message level

Void.

10.2 Information element level

Void.

10.3 Component level

Void.

10.4 Extensions for future releases in RRC

10.4.1 Basic principles

All non-critical extensions are shown even if empty as it costs no bits.

10.4.2 Naming convention

The abstract type defining a message provides mechanisms to allow for extending the message in future releases:

- For critical extensions, this is done by defining the message as a CHOICE of two alternatives, one being the intended message structure, and the other being an empty SEQUENCE named "criticalExtensions".
- For non-critical extensions, this is done by defining an OPTIONAL element named "nonCriticalExtensions" of type "SEQUENCE {}" at the end of the message definition.

When extensions are introduced, this is done by replacing one of the empty SEQUENCEs by a new structure, that includes a new type containing the message extensions, and the same extension mechanism recursively for further extensions.

For critical extensions the new elements introduced to specify the extensions should be grouped together in an element with a name showing the release in which the extension was made, and this should be the same as for the new message root. For this naming, "r3" is used for Release '99, "r4" for Release 4, "r5" for Release 5 and so on.

For non-critical extensions the new elements introduced to specify the extensions should be grouped together in an element with a name showing the version of the specification where this extension will first be included, e.g. if the version of the specification being corrected is v3.7.0, then the suffix added to the name will be -v380ext (i.e. the next version).

If non-critical extensions for two different roots happen to be identical in contents, their types are still named differently, possibly with the second being declared as synonymous to the first.

An example is given below to illustrate these principles, on the message named "Test-msg".

```
-- In Release '99, the Test-msg is defined as following:
Test-msg ::= CHOICE {
    r3                              SEQUENCE {
        test-msg-r3                     Test-msg-r3-IEs,
        nonCriticalExtensions           SEQUENCE {} OPTIONAL
    },
    later-than-r3                   SEQUENCE {
        rrc-TransactionIdentifier       RRC-TransactionIdentifier,
        criticalExtensions              SEQUENCE {}
    }
}
-- A later correction to Release '99 adds a non-critical extension in v3.8.0
-- of the specification
Test-msg ::= CHOICE {
    r3                              SEQUENCE {
        test-msg-r3                     Test-msg-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
            test-msg-v380ext                Test-msg-v380ext-IEs,
            nonCriticalExtensions           SEQUENCE {} OPTIONAL
        } OPTIONAL
    },
    later-than-r3                   SEQUENCE {
        rrc-TransactionIdentifier       RRC-TransactionIdentifier,
        criticalExtensions              SEQUENCE {}
    }
}
-- The Test-msg gets the following structure, if only a non-critical
-- extensions is introduced for Release 4 in v4.4.0 of the specification.
Test-msg ::= CHOICE {
    r3                              SEQUENCE {
        test-msg-r3                     Test-msg-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
            test-msg-v380ext                Test-msg-v380ext-IEs,
            laterNonCriticalExtensions      SEQUENCE {
                -- Container for additional Release '99 extensions
                test-msg-r3-add-ext             BIT STRING
                    (CONTAINING Test-msg-r3-add-ext-IEs)        OPTIONAL,
                v440nonCriticalExtensions       SEQUENCE {
                    test-msg-v440ext                Test-msg-v440ext-IEs,
                    nonCriticalExtensions           SEQUENCE {} OPTIONAL
                }   OPTIONAL
            }   OPTIONAL
        } OPTIONAL
    },
    later-than-r3                   SEQUENCE {
        rrc-TransactionIdentifier       RRC-TransactionIdentifier,
        criticalExtensions              SEQUENCE {}
    }
}

-- In Release 5, the Test msg gets the following structure when a critical
-- extension is added
Test-msg ::= CHOICE {
    r3                              SEQUENCE {
        test-msg-r3                     Test-msg-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
```

```
            test-msg-v380ext                    Test-msg-v380ext-IEs,
            laterNonCriticalExtensions          SEQUENCE {
                -- Container for additional Release '99 extensions
                test-msg-r3-add-ext                 BIT STRING
                    (CONTAINING Test-msg-r3-add-ext-IEs)            OPTIONAL,
                v440nonCriticalExtensions           SEQUENCE {
                    test-msg-v440ext                    Test-msg-v440ext-IEs,
                    nonCriticalExtensions               SEQUENCE {} OPTIONAL
                }   OPTIONAL
            }   OPTIONAL
        } OPTIONAL
    },
    later-than-r3                   SEQUENCE {
        rrc-TransactionIdentifier       RRC-TransactionIdentifier,
        criticalExtensions              CHOICE {
            r5                              SEQUENCE {
                test-msg-r5                     Test-msg-r5-IEs,
                nonCriticalExtensions           SEQUENCE {} OPTIONAL
            },
            criticalExtensions              SEQUENCE {}
        }
    }
}
```

Critical extensions in Release *N* in message "`Test-msg`" should be included in the type "`Test-msg-r`*N*`-IEs`" (*N*=3 is used for Release '99).

If an abstract type is introduced in Release *N* when new elements are included in an extension, it should have a suffix "`-r`*N*". For Release '99 types, no such suffix is used. In case the type that is introduced in Release *N* includes one or more new (nested) types, the additional suffix need not be used for these nested types. In case the type that is introduced in Release *N* includes one or more revisions of existing types, the suffix is needed to distinguish them from the earlier revisions. In case a revision of an abstract type that is introduced in Release *N* includes an IE for which the abstract type already existed in earlier releases, while that IE was not present in the previous revision(s) of the revised abstract type, the IE name should have a suffix "`-r`*N*".

If an abstract type is introduced in a release to extend an already existing type "`TypeX`", it should get the same name with a non-critical extension type suffix ("`-vXYZext`", e.g. "`TypeX-v380ext`") although in this case the final "–IEs" suffix is not added. In case the type that is introduced in Release *N* to extend an already existing type includes one or more new (nested) types that are extensions of an already existing type, the additional suffix should not be used for these nested types. In case the type that is introduced in Release *N* to extend an already existing type includes one or more new (nested) types, the abovely specified rules for new abstract types apply.

The above naming conventions are further illustrated in the example below:

```
Test-msg-v380ext-IEs ::= SEQUENCE {
        existingIE-A-v380ext            ExistingIE-A-v380ext        OPTIONAL,
        newIE-B                         NewIE-B
}

Test-msg-v440ext-IEs::= SEQUENCE {
        newIE-C-r4                      NewIE-C                     OPTIONAL,
        existingIE-D-v440ext            ExistingIE-D-v440ext
}

Test-msg-r5-IEs::= SEQUENCE {
        existingIE-E                    ExistingIE-E
        newUseOfexistingIE-F-rF         ExistingIE-F                OPTIONAL,
        newIE-G-r5                      NewIE-G
        revisionOfExistingIE-H-r5       ExistingIE-H-r5
}
```

The abovely described naming convention means that some IEs introduced in a later release/ version need not apply a specific suffix. This means that it will not allways be clear from the name of an IE whether or not backwards incompatible changes to it are allowed. The Message type is a special case, which can be changed by replacing the empty SEQUENCEs with extensions as shown above, and elements having spare values defined, where the spare value can be replaced with a newly introduced value.

An exception to the above structure can be needed, if there are some elements to be used in a message, which need to be comprehended even in case of critical extensions (e.g. for error handling procedures). In this case, the elements can be placed before one of the criticalExtensions CHOICEs, as shown in the example below:

```
Test-msg ::= CHOICE {
    r3                              SEQUENCE {
        test-msg-r3                     Test-msg-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
            test-msg-v380ext                Test-msg-v380ext-IEs,
            nonCriticalExtensions           SEQUENCE {} OPTIONAL
        } OPTIONAL
    },
    later-than-r3                   SEQUENCE {
        rrc-TransactionIdentifier       RRC-TransactionIdentifier,
        criticalExtensions              SEQUENCE {
            importantElements               ImportantElements,
            rest-of-message                 CHOICE {
                r4                              SEQUENCE {
                    test-msg-r4                     Test-msg-r4-IEs,
                    nonCriticalExtensions           SEQUENCE {} OPTIONAL
                },
                criticalExtensions              SEQUENCE {}
            }
        }
    }
}
```

In the above example, the elements in "importantElements" can be comprehended from a UE implementing this structure, even if a future version of the message including critical extensions is transmitted (i.e. the criticalExtension branch of the second CHOICE is used).

NOTE 1: The structure presented in this clause and the proposed naming rules are one possibility. Further possibilities are FFS.

NOTE 2: When non-critical extensions are introduced in a message that does not have yet a criticalExtension branch, they are introduced in the "Test-msg-v380ext-IEs" type as described above. It is possible, that after this change, another change introduces a critical extension for the same message, thus defining a critical extension branch. In this case, the whole message is redefined in the type "Test-msg-r*N*-IEs", and care is to be taken to include in this new type also all non-critical extensions that were introduced previously, in a way that best fits the new structure of the message.

- To be prepared for such cases, it could be beneficial to define in advance the "Test-msg-r*N*-IEs" whenever a non-critical extension is introduced, which would be an unused type mirroring the actual structure of the message, as long as no critical extensions are introduced, and would be used as the basis of the message if a critical extension is introduced. It is FFS if this concept is feasible, and if it should be introduced in the future.

10.4.3 Recommendations for extensions for further releases in RRC

10.4.3.1 General

When in RRC an information element group is to be extended, the extension cannot be done directly in that IE, but only in the top level of the message, in the extension IEs of the message structure shown in Example 1. For implementing the extension, it has therefore to be investigated, in which messages the element to be extended is included.

Depending on criticality of the extension, this will be done by using the criticalExtension CHOICE branch, or the nonCriticalExtension information element.

The following subclauses provide some recommendations on how to use these elements.

```
MessageA ::=                CHOICE {
    r3                              SEQUENCE {
        messageA-r3                     MessageA-r3-IEs,
        nonCriticalExtensions           SEQUENCE {} OPTIONAL
    },
    criticalExtensions              SEQUENCE {}
}
```

```
MessageA-r3-IEs ::=                 SEQUENCE {
    -- All messageA related information elements are included here.
}
```

Example 1

10.4.3.2 Critical Extensions

When the extension is a critical one (i.e. the receiver has to reject the whole message, and handle according to the error procedures of the protocol), the criticalExtension branch of the top-level CHOICE in the message is used. In this case the message information elements can be updated similar to the tabular, providing a message structure for the new release's information elements, similar to the updated structure in the tabular description.

Example 2 shows the structure of MessageA presented above, how it would become after a critical extension in Release 4.

In this example, in the criticalExtensions branch a new information element is defined (MessageA-r4-IEs) which will contain all messageA specific elements for Release 4, including the extensions in the place they fit naturally according to the semantics.

Note that in the new structure additional nonCriticalExtensions and criticalExtensions information elements are defined to allow for further extensions in future releases.

```
MessageA ::= CHOICE {
    r3                              SEQUENCE {
        messageA-r3                     MessageA-r3-IEs,
        nonCriticalExtensions           SEQUENCE {} OPTIONAL
    },
    later-than-r3                   SEQUENCE {
        rrc-TransactionIdentifier       RRC-TransactionIdentifier,
        criticalExtensions              CHOICE {
            r4                              SEQUENCE {
                messageA-r4                     MessageA-r4-IEs,
                nonCriticalExtensions           SEQUENCE {} OPTIONAL
            },
            criticalExtensions          SEQUENCE {}
        }
    }
}

MessageA-r3-IEs ::=                 SEQUENCE {
    -- This is not changed compared to the above example. It includes all information
    -- elements used in Release '99 for messageA.
}

MessageA-r4-IEs ::=                 SEQUENCE {
    -- Here, the updated information elements used for MessageA in Release 4 are included.
}
```

Example 2

10.4.3.3 Non-critical Extensions

For non-critical extensions (i.e. the receiver shall just ignore the extensions, and use the rest of the message as if the extensions were not present), the approach is to use the nonCriticalExtensions information element, which is encoded at the end of the message, allowing backward compatibility. For forward compatibility reasons, the transmitter should never include fields reserved for future non-critical extensions. This principle is illustrated using the example defined below: a transmitter conforming to the v380 version of the standard including MessageA as defined in Example 3 (see below), should not include the IE nonCriticalExtensions.

Before that Backward Compatibility is started for the following Release *N+1*, the non-critical extension information elements of the current Release *N* are added at the end of the message. At the point when Backward Compatibility is started for the following Release *N+1*, an optional BIT STRING container should be added before the information elements of the new release. In the case that further non-critical extension information elements need to be added to Release *N* they shall be placed within the BIT STRING container.

For example: As long as Backward Compatibility is not being enforced for Release 4, Release '99 extensions are added "normally" at the end of a message within a nonCriticalExtensions sequence. Once Backward Compatibility is started for Release 4, then new Release '99 specific extensions are introduced within an extension container. An extension container is a "normal" bit string field that encapsulates an extension structure. As a result:

- New extensions can be added **both** in Release '99 and Release 4 in a backward compatible way; and
- Release 4 systems are able to skip over unknown Release '99 extensions.

The extension container can be viewed as a specific type of non-critical extension and it is included in the same way. If the extension container is added to Release *N* before that Backward Compatibility has started for release *N+1*, further non-critical extensions to Release *N* should not be included in the container, but should be placed after it using the usual mechanism. In this way the extension container is not used until necessary, and therefore the corresponding length field overhead is not incurred unnecessarily. Additional guidelines concerning the use of extension containers are provided in subclause 10.4.3.5.

The structure of the message of the example above is shown in Example 3 for Release '99 and 4 messages.

Examples for special non-critical extensions and MessageA-v440ext-IEs are given in the following subclauses.

```
-- This shows the message structure in Release '99 (including one non-critical extension)
-- before backward compatibility is started for Release 4.
MessageA ::=                CHOICE {
    r3                              SEQUENCE {
        messageA-r3                     MessageA-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
            messageA-v380ext                MessageA-v380ext-IEs,
            nonCriticalExtensions           SEQUENCE {} OPTIONAL
        }   OPTIONAL
    },
    criticalExtensions              SEQUENCE {}
}

MessageA-r3-IEs ::=             SEQUENCE {
    -- This is not changed compared to the same IE in Release '99. It includes all information
    -- elements used in Release '99 for MessageA.
}

MessageA-v380ext-IEs :: =       SEQUENCE {
    -- Here are information elements added to Release '99 as extensions to the information
    -- contained in MessageA-r3-IEs.
}



-- This shows the Release '99 message structure once backward compatibility
-- has been started for Release 4.
MessageA ::=                CHOICE {
    r3                              SEQUENCE {
        messageA-r3                     MessageA-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
            messageA-v380ext                MessageA-v380ext-IEs,
            laterNonCriticalExtensions      SEQUENCE {
                -- Container for additional Release '99 extensions
                messageA-r3-add-ext                 BIT STRING
                    (CONTAINING MessageA-r3-add-ext-IEs)            OPTIONAL,
                nonCriticalExtensions       SEQUENCE {} OPTIONAL
            }   OPTIONAL
        }   OPTIONAL
    },
    criticalExtensions              SEQUENCE {}
}

MessageA-r3-IEs ::=             SEQUENCE {
    -- This is not changed compared to the same IE in Release '99. It includes all information
    -- elements used in Release '99 for MessageA.
}

MessageA-v380ext-IEs :: =       SEQUENCE {
    -- Here are information elements added to Release '99 as extensions to the information
    -- contained in MessageA-r3-IEs.
}
```

```
MessageA-r3-add-ext-IEs :: =        SEQUENCE {
    -- Here are information elements added to Release '99 as extensions to the information
    -- contained in MessageA-r3-IEs after backward compatibility was started for Release 4.
}



-- This shows the structure of the Release 4 message
-- (including one Release 4 non-critical extension).
MessageA ::=                CHOICE {
    r3                              SEQUENCE {
        messageA-r3                     MessageA-r3-IEs,
        v380nonCriticalExtensions           SEQUENCE {
            messageA-v380ext                    MessageA-v380ext-IEs,
            laterNonCriticalExtensions          SEQUENCE {
                -- Container for additional Release '99 extensions
                messageA-r3-add-ext                 BIT STRING
                    (CONTAINING MessageA-r3-add-ext-IEs)            OPTIONAL,
                v440nonCriticalExtensions           SEQUENCE {
                    messageA-v440ext                    MessageA-v440ext-IEs,
                    nonCriticalExtensions           SEQUENCE {} OPTIONAL
                }   OPTIONAL
            }   OPTIONAL
        }   OPTIONAL
    },
    criticalExtensions              SEQUENCE {}
}

MessageA-r3-IEs ::=                 SEQUENCE {
    -- This is not changed compared to the same IE in Release '99. It includes all information
    -- elements used in Release '99 for MessageA.
}

MessageA-v380ext-IEs :: =           SEQUENCE {
    -- Here are information elements added to Release '99 as extensions to the information
    -- contained in MessageA-r3-IEs.
}

MessageA-r3-add-ext-IEs :: =        SEQUENCE {
    -- Here are information elements added to Release '99 as extensions to the information
    -- contained in MessageA-r3-IEs after backward compatibility was started for Release 4.
}

MessageA-v440ext-IEs ::=            SEQUENCE {
    -- Here are information elements added to Release 4 as extensions to the information
    -- contained in MessageA-r3-IEs and MessageA-v380ext-IEs.
}
```

Example 3

10.4.3.4 Examples of non-critical extensions

10.4.3.4.1 Addition of a separate IE

If the extension is the addition of an information element (not inside a CHOICE, SEQUENCE OF, SET OF etc.), this new element can be directly included in MessageA-v440ext-IEs.

Example4 shows how the MessageA is extended to include a new element, "element3".

```
MessageA-r3-IEs ::=                 SEQUENCE {
    element1                            Element1,
    element2                            Element2
}

MessageA-v440ext-IEs ::=                SEQUENCE {
    element3                            Element3-r4
}
```

Example 4

10.4.3.4.2 Addition of an IE to a structured group

If the extension is the addition of an information element inside a CHOICE, SEQUENCE OF, etc. (meaning that the information element can be absent or present more than once, depending on some condition), the structure of the original message should be duplicated in MessageA-v440ext-IEs using only the elements relevant to the extension (usually the CHOICEs, SEQUENCE OFs, etc.), and a comment should be included to indicate that the two structures should be used consistently (e.g. when a CHOICE is duplicated, the same branch should be followed in both places, when a SEQUENCE OF is duplicated, the number of occurrences should be the same etc.).

This is illustrated in Example5, where a new element, "element1a-3", has to be included inside the "choice1b" branch of the "choice1" CHOICE. Here "choice1" is included again in MessageA-v440ext-IEs, and "element1a-3" is included there in the appropriate branch.

```
MessageA-r3-IEs ::=                 SEQUENCE {
-- For the "choice1b" branch of "choice1", an additional information element is
-- defined in MessageA-v440ext-IEs ("element1a-3").
    choice1                             CHOICE {
        choice1a                            SEQUENCE {
            element1a-1                         Element1a-1
        },
        choice1b                            SEQUENCE {
            element1a-2                         Element1a-2
        }
    }
}

MessageA-v440ext-IEs ::=                SEQUENCE {
-- In the following CHOICE the same branch shall be used as in choice1 in MessageA-r3-IEs.
    choice1                             CHOICE {
        choice1a                            NULL,
        choice1b                            SEQUENCE {
            element1a-3                         Element1a-3-r4
        }
    }
}
```

Example 5

10.4.3.4.3 Addition of a new CHOICE group

If the extension consists of moving some existing information elements inside a newly created CHOICE, the new branches of the created CHOICE should be included in MessageA-v440ext-IEs, and the CHOICE marked OPTIONAL, where absence means that the old elements are used. If the CHOICE is present, the old elements should be set to some default values, in order for older equipment to be understood, and new equipment should ignore the information therein.

This is illustrated in Example 6, where "element1" is to be moved inside the branch "choice1a" of a new CHOICE ("choice1").

```
MessageA-r3-IEs ::=                 SEQUENCE {
-- The contents of "element1" shall be ignored, if in "MessageA-v440ext-IEs" the branch
-- "choice1b" of the CHOICE "choice1" is used.
    element1                            Element1,
    element2                            Element2
}

MessageA-v440ext-IEs ::=                SEQUENCE {
    choice1                             CHOICE {
        choice1a                            SEQUENCE {},
        choice1b                            SEQUENCE {
            element3                            Element3-r4
        }
    }
}
```

Example 6

10.4.3.4.4 Extension of value range

If the value range of an element is to be extended, an element including the new values should be defined in MessageA-v440ext-IEs. If one of the new values is to be used, the already existing element from Release '99 should be set to some

defined value (or be absent if it was OPTIONAL), in order for older equipment to work properly, and the new value should be signalled in the new information element.

In Example 7, "element1" is extended to have a range (0..15).

```
MessageA-r3-IEs ::=                 SEQUENCE {
-- "element1" shall be ignored if "element1" in MessageA-v440ext-IEs is present, and the
-- value of that element used instead.
    element1                            INTEGER (0..7)
    element2                            Element2
}

MessageA-v440ext-IEs ::=                SEQUENCE {
    element1                            INTEGER (0..15)         OPTIONAL
}
```

Example 7

10.4.3.4.5 Replacement of a spare value with a new element

If a new value is to be included in an IE of type ENUMERATED, for which spare values were defined in the previous version, those spare values can be replaced with the new values.

If more new values are needed, than spare values included in the previous version, one spare value can be replaced by a special extension value (called e-new in example 8). If that value is used, a new element in the nonCriticalExtension part (element1-new) will define the new values, as shown in Example 8.

```
-- In the previous version, MessageA-r3-IEs was defined:
MessageA-r3-IEs ::=                 SEQUENCE {
    element1                            ENUMERATED { e1, e2, spare1, spare2 }
}

-- Now three new values are needed for element1: e3, e4 and e5. MessageA-r3-IEs is redefined:
MessageA-r3-IEs ::=                 SEQUENCE {
-- If the following has the value e-new, the actual value of element1 is defined in
-- element1-new included in MessageA-r4-ext-IEs
    element1                            ENUMERATED { e1, e2, e3, e-new }
}

MessageA-r4-ext-IEs ::=             SEQUENCE {
-- the following shall be present, if element1 in MessageA-r3-IEs has the value e-new.
    element1-new                        ENUMERATED { e4, e5, spare1, spare2 }   OPTIONAL
}
```

Example 8

If a spare value is included in a CHOICE, and that has to be replaced with a new information element and an appropriate type in the new version, the name of the element replaces the spare name in the CHOICE, but the type cannot be replaced, because that would lead to incompatibilities. Instead, the new type is included in the nonCriticalExtension part of the message, as shown in Example 9.

```
-- In the previous version, MessageA-r3-IEs was defined:
MessageA-r3-IEs ::=                 SEQUENCE {
    element1                            CHOICE {
        e1                                  E1,
        e2                                  E2,
        spare                               NULL
    }
}

-- Now a new option is needed for the element1 CHOICE: e3 with type E3.
-- MessageA-r3-IEs is redefined:
MessageA-r3-IEs ::=                 SEQUENCE {
-- If element1 has the value e3, the value of e3 is specified in the element e3
-- included in MessageA-r4-ext-IEs.
    element1                            CHOICE {
        e1                                  E1,
        e2                                  E2,
        e3                                  NULL
    }
```

```
}

MessageA-r4-ext-IEs ::=             SEQUENCE {
-- the following shall be present, if element1 in MessageA-r3-IEs has the value e3.
    e3                                  E3          OPTIONAL
}
```

Example 9

10.4.3.4.6 Introducing new System Information Block Types

In general new message types are introduced by replacing a spare value as described in subclause 10.4.3.4.5. That subclause also shows that in case there are insufficient spare values available, the last spare value can be replaced by a special extension value. If that value is used, an additional message type extension IE is included to distinguish between the additional message types, as shown in Example 10.

```
DL-CCCH-Message ::= SEQUENCE {
    integrityCheckInfo      IntegrityCheckInfo      OPTIONAL,
    message                 DL-CCCH-MessageType
}

DL-CCCH-MessageType ::= CHOICE {
    cellUpdateConfirm                   CellUpdateConfirm-CCCH,
    rrcConnectionReject                 RRCConnectionReject,
    rrcConnectionRelease                RRCConnectionRelease-CCCH,
    rrcConnectionSetup                  RRCConnectionSetup,
    uraUpdateConfirm                    URAUpdateConfirm-CCCH,
    ext1                                Ext1Message-CCCH,
    ext2                                Ext2Message-CCCH,
    extension                           DL-CCCH-MessageTypeExt
}

DL-CCCH-MessageTypeExt ::= CHOICE {
    Ext3                                Ext3Message-CCCH,
    spare3                              NULL,
    spare2                              NULL,
    spare1                              NULL
}
```

Example 10

For system information block types, the "SIB type" information element is also included in each of the segments. If in this case there are insufficient spare values, the last value can again be used to indicate "extension". If that value is used, an additional SIB type extension IE is included to distinguish between the additional SIB types. This additional IE is not included in the segments; it is only included in the scheduling information included in the MIB and/or the SBs.

NOTE: One could include this additional IE in the segments e.g. by changing the SIB-type into a choice as shown in example 11. This option should not be used since it involves additional overhead (more scarce BCH bits are needed to indicate the SIB type) and complicates the scheduling (more different SIB data sizes are to be considered).

```
FirstSegment ::=                    SEQUENCE {
    -- Other information elements
        sib-Type                        SIB-Type,
        seg-Count                       SegCount,
        sib-Data-fixed                  SIB-Data-fixed
}

SIB-Type ::=                            CHOICE {
        MasterInformationBlock              NULL,
        systemInformationBlockType1         NULL,
        systemInformationBlockType2         NULL,
        systemInformationBlockType3         NULL,
        systemInformationBlockType4         NULL,
        systemInformationBlockType5         NULL,
        systemInformationBlockType6         NULL,
        systemInformationBlockType7         NULL,
        systemInformationBlockType8         NULL,
        systemInformationBlockType9         NULL,
        systemInformationBlockType10        NULL,
```

```
        systemInformationBlockType11        NULL,
        systemInformationBlockType12        NULL,
        systemInformationBlockType13        NULL,
        systemInformationBlockType13-1      NULL,
        systemInformationBlockType13-2      NULL,
        systemInformationBlockType13-3      NULL,
        systemInformationBlockType13-4      NULL,
        systemInformationBlockType14        NULL,
        systemInformationBlockType15        NULL,
        systemInformationBlockType15-1      NULL,
        systemInformationBlockType15-2      NULL,
        systemInformationBlockType15-3      NULL,
        systemInformationBlockType16        NULL,
        systemInformationBlockType17        NULL,
        systemInformationBlockType15-4      NULL,
        systemInformationBlockType18        NULL,
        schedulingBlock1                    NULL,
        schedulingBlock2                    NULL,
        systemInformationBlockType15-5      NULL,
        ext1                                NULL,
        extension                           SIB-TypeExt
}

SIB-TypeExt ::=                     CHOICE {
        ext2                            NULL,
        spare7                          NULL,
        spare6                          NULL,
        spare5                          NULL,
        spare4                          NULL,
        spare3                          NULL,
        spare2                          NULL,
        spare1                          NULL
}
```

Example 11 – Not recommended

The addition of new SIB types to the scheduling information is illustrated by example 12. The example shows the extension of the choice. The example also shows that the information applicable for the extended choice values is appended at the end of the SIB (in this case the MIB), as a non critical extension.

NOTE: In this example only the number of SIB types is increased; the number of SIBs that can be scheduled (as reflected in the size of the list in the scheduling information) is not extended.

```
MasterInformationBlock ::=          SEQUENCE {
        mib-ValueTag                    MIB-ValueTag,
        -- TABULAR: The PLMN identity and ANSI-41 core network information
        -- are included in PLMN-Type.
        plmn-Type                       PLMN-Type,
        sibSb-ReferenceList             SIBSb-ReferenceList,
        vxy0NonCriticalExtensions           SEQUENCE {
            masterInformationBlock-vxy0ext      MasterInformationBlock-vxy0ext-IEs,
            nonCriticalExtensions               SEQUENCE {}                     OPTIONAL
        } OPTIONAL
}

SIBSb-ReferenceList ::=             SEQUENCE (SIZE (1..maxSIB)) OF
                                        SchedulingInformationSIBSb

SchedulingInformationSIBSb ::=          SEQUENCE {
    sibSb-Type                          SIBSb-TypeAndTag,
    scheduling                          SchedulingInformation
}

SIBSb-TypeAndTag ::=                    CHOICE {
    sysInfoType1                        PLMN-ValueTag,
    sysInfoType2                        CellValueTag,
    sysInfoType3                        CellValueTag,
    sysInfoType4                        CellValueTag,
    sysInfoType5                        CellValueTag,
    sysInfoType6                        CellValueTag,
    sysInfoType7                        NULL,
    sysInfoType8                        CellValueTag,
    sysInfoType9                        NULL,
    sysInfoType10                       NULL,
    sysInfoType11                       CellValueTag,
```

```
    sysInfoType12                       CellValueTag,
    sysInfoType13                       CellValueTag,
    sysInfoType13-1                     CellValueTag,
    sysInfoType13-2                     CellValueTag,
    sysInfoType13-3                     CellValueTag,
    sysInfoType13-4                     CellValueTag,
    sysInfoType14                       NULL,
    sysInfoType15                       CellValueTag,
    sysInfoType16                       PredefinedConfigIdentityAndValueTag,
    sysInfoType17                       NULL,
    sysInfoTypeSB1                      CellValueTag,
    sysInfoTypeSB2                      CellValueTag,
    sysInfoType15-1                     CellValueTag,
    sysInfoType15-2                     SIBOccurrenceIdentityAndValueTag,
    sysInfoType15-3                     SIBOccurrenceIdentityAndValueTag,
    sysInfoType15-4                     CellValueTag,
    sysInfoType18                       CellValueTag,
    sysInfoType15-5                     CellValueTag,
    ext1                                NULL,
    ext2                                NULL,
    extension                           NULL
}

SIBSb-TypeAndTagExt ::=                 CHOICE {
    ext3                                NULL,
    spare7                              NULL,
    spare6                              NULL,
    spare5                              NULL,
    spare4                              NULL,
    spare3                              NULL,
    spare2                              NULL,
    spare1                              NULL
}

MasterInformationBlock-vxy0ext-IEs ::= SEQUENCE {
    extSIBTypeInfoSchedulingInfo-List       ExtSIBTypeInfoSchedulingInfo-List   OPTIONAL
}

-- For each extended SIB type the value tag information is added at the end
ExtSIBTypeInfoSchedulingInfo-List::=    SEQUENCE (SIZE (1..maxSIB)) OF
                                            ExtSIBTypeInfoSchedulingInfo

ExtSIBTypeInfoSchedulingInfo-List::= SEQUENCE {
    schedulingInfoListIndex             INTEGER (1..maxSIB),
    valueTagInfo                        ValueTagInfo
}

ValueTagInfo ::= CHOICE {
    None                                NULL.
    sysInfoType2                        CellValueTag,
    sysInfoType1                        PLMN-ValueTag,
    sysInfoType15-3                     SIBOccurrenceIdentityAndValueTag
}
```

Example 12 – Recommended method

10.4.3.5 Additional guidelines on the use of variable length extension containers

"Variable length extension containers" (i.e. non critical extension containers that have their abstract syntax defined using the ASN.1 type "BIT STRING") have been defined to support the introduction of extensions to a release after the subsequent release is frozen (and UEs based on that subsequent release may appear).

Extension containers should be introduced in each message unless the size of the message is critical and the likelihood of late corrections is low. For downlink messages for which different versions have been defined, an extensions container should be introduced for each message version (branch).

In case a variable length extension container (VLEC) includes an extension, the PER encoder will include an additional length determinant. In case a separate container is introduced for each release, this would result in a significant signalling overhead. In order to avoid this signalling overhead, the extensions container should not be dedicated to late corrections of one specific release. If the extensions container is required to support the introduction of late corrections in an order of release, one or more release specific extensions container(s) may be nested within the original extension container.

The above guidelines are illustrated by means of an example. Suppose a message includes a single VLEC and one late R99 correction has been defined using this extension container. When the need for a late Release 5 correction arises, this correction may be added to the extensions container in two different ways:

1. Just by adding the extension after the late Release '99 correction

2. By introducing a Release 5 extensions container, that is nested within the existing extensions container

In case the first option is used, the addition of another late Release '99 correction (after the Release 5 correction) would require the Release '99 receiver to comprehend the transfer syntax of the Release 5 extension. This would require the introduction of the late Release 5 correction (or a type with an equivalent encoding) in the Release '99 transfer syntax. The second option avoids this problem, although this comes at the cost of additional signalling overhead.

Which option to use should be decided when introducing an extension for a release later than Release '99. This can be decided on a case by case basis eg. depending on whether for the concerned message further late Release '99 corrections need to be accommodated.

10.4.3.6 Use of non critical extensions for release independent features

The objective of release independent features is that they do not require the UE to implement a specific release of the specification. Some release independent features may require the introduction of a non critical extension in one or more messages. In this case the recommendation is to introduce the extensions in the latest release of the specification for which the ASN.1 is frozen. This applies both to the tabular and the ASN.1.

In case, for some reasons eg. alignment with other specifications/ groups, the release independent feature is introduced in release X, which is later than the release recommended according to the above, the following recommendations apply:

1. Within the tabular notation, the non critical extension is introduced in release X.

2. Within the ASN.1, the information element is introduced as a non critical extension in release Y, which is the latest release of the specification for which the ASN.1 is frozen.

3. Within the ASN.1 of releases z, with $Y \leq z < X$, the non critical extension is provisioned for by introducing a dummy information element that is encoding compatible with the actual information element introduced in release X

The reason for the provisioning of the extension in the ASN.1 of specifications earlier than release X is that for releases z, with $Y \leq z < X$, it is still possible to modify the ASN.1. As a result, if no provisions would be created in such releases, it would not be possible to use the release independent feature until release X is frozen.

An alternative approach would be to include the non critical extension within the variable length extension container (VLEC). However, considering the overhead associated with the introduction of such a container this approach is not recommended.

Note For release Y it is allowed to introduce non critical extensions to release Y. Furthermore, backwards incompatible changes are allowed for later releases. If the release independent feature is introduced in release X, which is later than Y, it will be affected by such changes.

10.5 Extensions for future releases in RANAP, RNSAP, NBAP; and SABP

The following clauses contain rules for extension mechanisms of ASN.1 for RANAP, SABP, RNSAP and NBAP. The purpose of these rules is to guarantee backward compatibility for ASN.1.

10.5.1 Allowed Extension

The allowed extension for ASN.1 description in RANAP, SABP, RNSAP and NBAP are:

1) adding New IEs or IE groups which should be achieved by using the protocol extension container (extension by using of ellipsis notation (...) should be avoided) for:

- adding at the top level of message; and
- adding in the SEQUENCE type,

2) extending the range of already define IEs which has ellipsis notation(...);

3) changing the assigned criticality information of already defined IEs; and

4) adding new IEs of IE groups after ellipsis notation (...) in the CHOICE type if the ellipsis notation (...) is present.

10.5.2 Not Allowed Extension

The not allowed extension for ASN.1 description in RANAP, SABP, RNSAP and NBAP are:

1) deleting the already defined IEs or IE groups when no individual criticality information is defined;

2) adding or deleting the criticality information of existing IEs;

3) deleting the already defined values in the ASN.1 type. Instead, a semantic description is added in order to clarify the behaviour; and

4) changing the presence of already defined IEs with no assigned criticality.

This is because above changes do not guarantee the backward compatibility.

10.5.3 Recommendations for extensions for further releases

10.5.3.1 General

This subclause gives recommendations for future extensions in versions of the RANAP, RNSAP, NBAP, and SABP where non-backward compatible changes are not acceptable.

10.5.3.2 Usage of Presence and Assigned Criticality in Future Releases

10.5.3.2.1 New Procedures

For procedures introduced when the backward compatibility mechanisms are taken into use the following recommendation applies to the Assigned Criticality of the procedure (in the tabular description of messages visible as the Assigned Criticality of the IE Message Type).

Assigned Criticality	Recommendation	Typical usage
Ignore	Should be used if: - the sender does not care whether or not the procedure is supported; or - if the sender "already knows" that the procedure is supported.	Typically used for procedures where: - the sender do not care whether or not the procedure is supported; or - where the usage is dependent on previously exchanged information.
Ignore and Notify	Should be used if: - the sender does not care whether or not the procedure is supported; or - if the sender "already knows" that the procedure is supported; but need to know whether or not the procedure was understood.	Typically not used.
Reject	Should be used if: - the procedure shall be rejected when not supported.	Typically used for new procedures where the sender has no prior knowledge on whether or not the procedure will be understood.

10.5.3.2.2 New IEs

For new IEs introduced when the backward compatibility mechanisms are taken into use the following recommendation applies to the Assigned Criticality of the IE.

Presence	Assigned Criticality	Recommendation	Typical usage
Optional	Ignore	Should be used if the sender does not care whether or not the function related to the IE is supported.	Typically used for "non core" features (specification text; "… shall, if supported, …".
	Ignore and Notify	Should be used if: - the sender does not care whether or not the function related to the IE is supported. but need to know whether or not the IE was understood.	Typically used for "non core" features (specification text; "… shall, if supported, …").
	Reject	Should be used if: - the alternative to executing the feature related to the IE is rejecting the procedure.	Typically used for "core" features (specification text; "… shall …").
Mandatory / Conditional	Ignore	Should be used for "core" features where: - it is essential that all implementations of future releases support the feature related to the IE; - it is possible to inter-work with nodes implementing older releases (not understanding the IE related to the feature and consequently not supporting the feature).	Typically not used. Note 1.
	Ignore and Notify	Should be used for "core" features where: - it is essential that all implementations of future releases support the feature related to the IE; - it is possible to inter-work with nodes implementing older releases (not understanding the IE related to the feature and consequently not supporting the feature); but the sending node need to know whether or not the IE was understood.	Typically not used. Note 1.
	Reject	Should be used for "core" features where: - it is essential that all implementations of future releases support the feature related to the IE; - it is not possible to inter-work with nodes implementing older releases (not supporting the feature).	Typically not used. Note 2.

NOTE 1: This combination (presence + assigned criticality) could be used as an intermediate state, i.e. when the Assigned Criticality is expected/planned to be changed to "Reject" in the future.

NOTE 2: This combination (presence + assigned criticality) should be avoided since it prevents inter-working with older version of a specification.

10.5.3.2.3 Changing the Presence of an IE

The Presence can always be changed in future version of a specification.

NOTE: Mandatory and Conditional IEs with Assigned Criticality "Reject" will still cause rejection when missing in a node based on a previous version of the specification (even though changed to Optional).

Recommendation:

The Presence of Mandatory IEs with Assigned Criticality "Reject" should not be changed in future versions of a specification.

The Presence of Conditional IEs with Assigned Criticality "Reject" should not be changed in future versions of a specification, unless it is also ensured that the condition will not result in a requirement to include the IE.

10.5.3.2.4 Changing the Assigned Criticality of an IE

The Assigned Criticality can always be changed in future version of a specification.

NOTE: The behaviour for missing IEs will remain unchanged when inter-working with a node based on a previous version of the specification.

Recommendation:

When changing the Assigned Criticality of Mandatory and Conditional IEs with Assigned Criticality "Reject" in future versions of a specification special attention should be paid to inter-working between different versions of the specification.

10.5.3.2.5 Removing IEs

Any IE (with Assigned Criticality) can be removed in future version of a specification.

NOTE: Mandatory and Conditional IEs with Assigned Criticality "Reject" will still cause rejection when missing in a node based on a previous version of the specification (even though changed to Optional).

Recommendation:

Mandatory IEs with Assigned Criticality "Reject" should not be removed in future versions of a specification.

Conditional IEs with Assigned Criticality "Reject" should not be removed in future versions of a specification, unless it is also ensured that the condition, if evaluated for the message where the IE is removed by a node based on a previous version of the specification, will not result in a requirement to include the IE.

10.5.4 Use of extensions

The following rules apply to the use of extensions in RANAP, SABP, RNSAP and NBAP:

1) IEs added in the extension containers shall be added before the ellipsis notation.

2) IEs added in the extension containers shall be placed in the extension containers in the chronological order of their introduction e.g. a late correction of Release '99 may appear after Release 4 corrections in a Release 4 version of the specification.

3) When new values are added into an ENUMERATED type after the ellipsis notation in the later release (e.g. Release 4), the same amount of new values need to be added as "dummy" in the same IE in the correspondent older release of the specification (e.g. Release '99).

   The "dummy" has to be added into the older release (e.g. Release '99) of the specifications when needed, i.e. if the later release (e.g. Release 4) of specification is considered as not stable, the "dummy" shall not be added in the same IE in the correspondent older release (e.g. Release '99) of the specification. When the older release (e.g. Release '99) adds new values for later version (e.g. Release '99 June version), the same values shall be added in the same position of the correspondent later release (e.g. Release 4) in order to have backward compatibility when in the future the later release (e.g. Release 4) is considered as stable. Further explanation with an example is shown below:

   - Release '99 June version: *Example* ::= ENUMERATED {a, b,…};

   - Release 4 June version: *Example* ::= ENUMERATED {a, b,…, c, d};

NOTE 1: In the June version the Release 4 is considered as not stable, therefore no "dummy" was added in Release '99).

   - Release '99 September version, *Example* ::= ENUMERATED {a, b,…, e};

   - Release 4 September version, *Example* ::= ENUMERATED {a, b,…, e, c, d};

NOTE 2: In the September version the value "e" was added in Release '99, the value "e" shall be added at the same position in Release 4 since the Release 4 is yet considered as not stable.

- Release '99 December version, *Example* ::= ENUMERATED {a, b,…, e, dummy1, dummy2, f};
- Release 4 December version, *Example* ::= ENUMERATED {a, b,…, e, c, d, f}.

NOTE 3: In the December version, when a new value "f" has to be added in Release '99 and the Release 4 is considered as stable, "dummy1" and "dummy2" are added.

4) When new choice tags are added into a CHOICE type after the ellipsis notation, new tags shall need to be added in the protocol container.

- Any addition of a choice tag in Release x for a CHOICE not existing in Release y (y<x) shall be made by introducing a normal choice tag (not included in the protocol container) after the ellipses if no release z (z >= x) has included a protocol container for this CHOICE yet. Further explanation with an example is shown below:
  - Release '99 June version, *Example* ::= CHOICE {a, b,…};
  - Release 4 June version, *Example* ::= CHOICE {a, b,…};
  - Release '99 September version, *Example* ::= CHOICE {a, b,…, c};
  - Release 4 September version, *Example* ::= CHOICE {a, b,…, c}.
- Any addition of a choice tag in Release x for a CHOICE already existing in Release y (y<x) shall be made inside a protocol container after the ellipses. Further explanation with an example is shown below:
  - Release '99 June version: *Example* ::= CHOICE {a, b,…};
  - Release 4 June version: *Example* ::= CHOICE {a, b,…};
  - Release '99 September version: *Example* ::= CHOICE {a, b,…};
  - Release 4 September version: *Example* ::= CHOICE {a, b,…, protocol container {d}}.
- Any addition of a choice tag in Release x for a CHOICE already containing a protocol container in Release y (y>x) shall be made inside a protocol container after the ellipses. Further explanation with an example is shown below:
  - Release '99 June version: *Example* ::= CHOICE {a, b,…};
  - Release 4 June version: *Example* ::= CHOICE {a, b,…, protocol container {d}};
  - Release '99 September version: *Example* ::= CHOICE {a, b,…, protocol container {e}};
  - Release 4 September version: *Example* ::= CHOICE {a, b,…, protocol container {d, e}}.
- If any Release has already included a protocol container in this CHOICE, then all future changes to this CHOICE shall be made by introducing IE's in the protocol container. Further explanation with an example is shown below:
  - Release 4 June version: *Example* ::= CHOICE {a, b,…, protocol container {d, e}};
  - Release 4 September version: *Example* ::= CHOICE {a, b,…, protocol container {d, e, f}}.

10.6 Comments

Comments in ASN.1 are a mechanism for providing essential or useful information about the definitions to which they apply. However, their use must be consistent or they can cause confusion and misunderstandings.

The placement of comments is important as it aides understanding of to which elements they refer to. In the following example the comments are used as section headings, grouping together the IEs that follow until the next section heading. In this case the comment is placed at one level less (one TAB less) than the IEs to which it refers. This style of comment is normally used to reflect the groupings of IEs in messages as shown in the tabular format.

```
ExampleMsg ::= SEQUENCE {
    -- First group of IEs
        exampleA                    ExampleTypeA,
        exampleB                    ExampleTypeB    OPTIONAL,
    -- Next group of IEs
        exampleC                    ExampleTypeC,
    -- Extension mechanism
        nonCriticalExtensions       SEQUENCE {} OPTIONAL
}
```

The IEs that are referred to or grouped, are implied from the relative levels and so it is not required to include their names in the text of the comment itself.

In the next example the comment is used to refer to a specific IE. In this case the comment should always be placed directly above the IE to which it refers and at the same level of indentation. The name of the IE should be contained in the text of the comment so as to make it completely unambiguous to which IE it refers.

```
ExampleTypeA ::= SEQUENCE {
    subTypeA1       SubTypeA1,
    -- Actual value subTypeA2 = IE value * 4
    subTypeA2       INTEGER (0..16)
    sunTypeA3       SubTypeA3   OPTIONAL
}
```

If a comment of this type needs to refer to more than one IE or value then each must be specifically named in the text of the comment.

The next example shows the case where a comment is added to the ASN.1 to make its use clear when compared to the tabular format. In this case, as the text is purely to reflect information contained elsewhere, the comment is prefixed with "TABULAR:".

```
ExampleTypeB ::= SEQUENCE {
    subTypeB1       BOOLEAN,
    subTypeB2       SubTypeB2
    -- TABULAR: subTypeB3, presence is conditional on the value of subTypeB1
    sunTypeB3       SubTypeB3   OPTIONAL
}
```

11 Message transfer syntax specification

11.1 Selection of transfer syntax specification method

For RRC Basic Packed Encoding Rules (BASIC-PER) PER Unaligned Variant and possible use of specialised encoding is chosen.

For RANAP, RNSAP, NBAP, and SABP Basic Packed Encoding Rules (BASIC-PER) Aligned Variant is chosen.

11.2 Specialised encoding (only RRC)

11.2.1 General

Specialised encoding is an escape mechanism that allows the specification of exceptional encodings for parts of messages. Specialised encoding acts as an exception mechanism to the normally applied encoding rules (e.g. Unaligned PER).

The detailed encoding rules for specialised encodings are defined within an ECN module. A link module is used to associate an ECN module with an ASN.1 module. For example:

```
Example-ASN1-Module DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
    John ::= SEQUENCE {
        a BOOLEAN,
        b INTEGER
    }
END

Example-ECN-Module ENCODING-DEFINITIONS ::=
BEGIN
    IMPORTS John FROM Example-ASN1-Module;

    MyProc ::=
        USER-FUNCTION-BEGIN
            -- Description of special encoding goes here
        USER-FUNCTION-END

    John.b ENCODED BY MyProc
END

Example-Link-Module LINK-DEFINITIONS ::=
BEGIN
    Example-ASN1-Module ENCODED BY perUnaligned WITH Example-ECN-Module
END
```

In the above example the link module **Example-Link-Module** specifies that the ASN.1 module **Example-ASN1-Module** has the PER unaligned encoding rules as a default with extra specialised encoding defined in the ECN module **Example-ECN-Module**.

11.2.2 Notation in ASN.1

The ASN.1 modules shall contain only the abstract definition of the messages.

11.2.3 Notation in ECN

If specialised encodings are to be used, all such encodings shall be specified in an ECN module.

Several approaches are possible for specialised encoding. One approach is to use the ECN notation, which allows direct specification of encoding rules (see example 9). The other approaches are to specify using CSN.1 or to reference an encoding defined informally in an existing specification. These last two methods are explained in the following clauses.

11.2.3.1 Use of CSN.1

In this case, user functions are defined starting by "`--<ECN.Encoding CSN1>--`", and containing each one or several CSN.1 types. Specialised encoding of an ASN.1 type is indicated by "ENCODED BY" clauses referring to a CSN.1 user function and followed by the identifier of the CSN.1 type to apply for the encoding.

A user-function based on CSN.1 is limited to a list of descriptions, each description respecting the syntax of CSN.1 V2.0, preceded by the starting text mentioned above and optionally by an IMPORTS clause. The header part of modules as defined in CSN.1 V2.0 is not used. The IMPORTS clause respects the ASN.1 syntax.

NOTE: It is expected to move to CSN.1 V2.2 as soon as available.

The specialised encoding shall be such that all the relevant values of a type can be represented with it, i.e. there shall be a mapping from each meaningful abstract value to an encoded value, taking into account any applicable informally stated constraints. Reciprocally, decoding of any received string shall be mapped either to an abstract value or to an error indication.

In the case of a composite ASN.1 type (e.g.: choice or sequence), labels are used in the CSN.1 construction for the association with the corresponding parts in the abstract description (see examples 5 and 6). Case is significant. The

order of alternatives in a choice construction, or of fields in a sequence, may differ between the abstract and the representation descriptions (see example 7). On the other hand, incompleteness is a specification inconsistency.

In the CSN.1 module <ASN1.*Identifier*> is a reference to a construction defined in an ASN.1 module, as given by an IMPORTS clause at the beginning of the CSN.1 user function. This describes a construction as derived from the ASN.1 description (note that this might contain specialised encoding). This notation aims at distinguishing constructions defined in the CSN.1 module from those defined in an ASN.1 module. Such a reference could be replaced by a complete description in the CSN.1 module, however this would be redundant and cumbersome in the case of complex constructions. See example 3.

In some cases, an elementary ASN.1 type is replaced in the CSN.1 description by a sequence. In such a case, the field name 'V' is used as a label in the sequence to indicate the field that does encode the elementary type. See example 4.

11.2.3.2 Reference to informally specified encodings in other specifications

In this case, user functions are defined starting by "`--<ECN.Reference>--`", and containing a textual description of the reference. See example 8.

11.2.4 Notation in Link Module

If specialised encodings are to be used, a link module shall be used to associate the ASN.1 module(s) with the corresponding ECN module(s).

NOTE: All the specialised encodings for a given ASN.1 module shall be contained within a single ECN module. See example in subclause 11.2.6.3.

11.2.5 Detailed and Commented Examples

The different examples below illustrate different possibilities, and provide some explanations. Examples of complete modules can be found in subclause 11.2.6.

11.2.5.1 Example 1

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

In the ASN.1 module:

```
SparseEvenlyDistributedValueSet ::= INTEGER (0|2|4|6|8|10|12|14)
```

In the CSN.1 user function of the ECN module:

```
<SparseEvenlyDistributedValueSet> ::=
    bit(3);

    -- Representation: This represents the integer equal to half the
    -- binary encoding of the field
    -- e.g., 010 encodes integer 4
```

11.2.5.2 Example 2

An integer value set is not continuous and evenly distributed.

In the ASN.1 module:

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

In the CSN.1 user function of the ECN module:

```
<SparseValueSet> ::= bit(3) exclude {110 | 111};

-- Representation :
-- 0 => 000
-- 3 => 001
-- 5 => 010
-- 6 => 011
-- 8 => 100
-- 11 => 110
```

Explanations:

The exclusion part implies that the reception of 110 or 111 triggers an exception.

11.2.5.3 Example 3

A list type is encoded using the 'more' bit technique.

This allows optimising the cases where there are few components relatively to the maximum number of components.

In the ASN.1 module:

```
VariableLengthList ::= SEQUENCE (SIZE (0..10)) of Status
```

In the CSN.1 user function of the ECN module:

```
<VariableLengthList> ::=
    <Length : 1** 0>
    <V : <ASN1.Status>*(len(Length)-1);
```

Explanations:

<ASN1.Status> is a reference to a construction defined in the ASN.1 module.

The traditional 'more' bit technique looks like:

```
<Not recommended VariableLengthList> ::=
    { 1 <ASN1.Status> }(*)
    0;
```

It can be checked that the recommended construction is exactly the same except for the bit order (all the tags are grouped on the start). The recommended construction is highly preferable since it makes it clear that the 'more' bits are just a variable length encoding of a length field. The more traditional technique may have some application when alignment is a concern.

11.2.5.4 Example 4

A variable length integer using the 'more' bit technique.

This can be used to obtain an encoding of integers where efficiency is sought for small values, but bigger values are still allowed.

In the ASN.1 module:

```
VariableLengthList ::= INTEGER
```

In the CSN.1 user function of the ECN module:

```
<VariableLengthInteger> ::=
    <Length : 1** 0>
    <V : bit*3*(len(Length)-1)>;

-- This represents the integer encoded in binary by the V field
```

Explanations:

This makes use of the same basic technique than in the previous example.

The traditional 'more' bit technique looks like:

```
<Not recommended VariableLengthInteger> ::=
    { 1 bit(3) }(*)
    0;
```

It can be checked that the recommended construction is exactly the same except for the bit order (all the tags are grouped on the start). The recommended construction is highly preferable since it makes it clear that the 'more' bits are just a variable length encoding of a length field. In addition, it allows specifying the encoding/decoding of the integer as a continuous string.

11.2.5.5 Example 5

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.

In the ASN.1 module:

```
VariantRecord ::= CHOICE {
    flag    Flag,       -- The two first alternatives are mostly used
    counter Counter,
    extEnum ExtendedEnum,
    status  Status,
    list    VariableLengthList
}
```

In the CSN.1 user function of the ECN module:

```
<VariantRecord> ::=
    { 00  <flag    : <ASN1.Flag>>
    | 01  <counter : <ASN1.Counter>>
    | 100 <extEnum : <ASN1.ExtendedEnum>>
    | 101 <status> : <ASN1.Status>>
    | 110 <List    : <ASN1.VariableLengthList>>
    };
```

Explanations:

The tag list can be adapted precisely to the expected statistics. Any tag list such that no member is the start of another member is acceptable.

11.2.5.6 Example 6

The size of a component (e.g., integer, bit string, character string, sequence-of) depends on the value of one or several other components. The example here is that of an integer whose range depends on the value of another integer.

In the ASN.1 module:

```
ConditionalSized ::= SEQUENCE
    {
    modulo  INTEGER(1..2048),
    phase   INTEGER(0..2047)}
```

In the CSN.1 user function of the ECN module:

```
<ConditionalSized> ::=
    <modulo : bit(12)>
    <phase : bit*logval(modulo)>;

-- where logval is the function to the smaller integer higher or equal
-- to the logarithm in base 2 of 1 plus the integer encoded in binary in the
--  argument
-- e.g., logval(0101) = 3
--       logval(00) = 0
--       logval(10) = 2
-- this can be also described as the position of the last '1' in the argument,
-- starting from the end
```

11.2.5.7 Example 7

A specialised extension mechanism optimised for very short extensions.

In the ASN.1 module:

```
SpecialisedExtensionV1 ::= SEQUENCE  {
    c1  C1,
    c2  C2,
    extension SEQUENCE{} OPTIONAL
}
```

In the CSN.1 user function of the ECN module:

```
<Empty Extension> ::=
    <Length : <Extension Length>>
    bit* lval(Length) &
    {<SpuriousExtension : bit(*) = null>;

<Extension Length> ::=
    <L:0> |                 -- lval = 0
    1   <L : bit(3) - 111> |        -- lval = val(L) + 1
    1111 <L : bit(4)>;          -- lval = 8*val(L)+8
```

In the ECN module:

```
SpecialisedExtensionExampleV1.extension ENCODED BY CSN1Proc."Empty Extension"
```

Explanations:

The use of the intersection (&) is not needed in the empty extension placeholder. It is introduced here to prepare the description of the eventual extension, see further on.

The specialisation is on the encoding of the length field.

The '= null' forbids that a sender compliant with this version sends anything else than an empty 'extension', while the 'bit(*)' allows a receiver to accept any string (the end is constrained by the length field).

In an ulterior version this can become:

In the ASN.1 module:

```
SpecializedExtensionV2 ::= SEQUENCE  {
```

```
    c1  C1,
    c2  C2,
    extension SEQUENCE
    {c3 C3 OPTIONAL,
     c4 C4}
            }
```

In the CSN.1 USER-FUNCTION of the ECN module

```
< Extension of SpecialExtensionV2 > ::=
    <Length : <Extension Length>>
    bit* lval(Length) &
    {   <c4 : <ASN1.C4>>
        {0 | 1 <c3 : <ASN1.C3>>}
        <Spurious Extension : bit(*) = null>
    }//;
;
```

In the ECN module:

```
SpecialisedExtensionExampleV1.extension ENCODED BY CSN1Proc."Extension of SpecialExtensionV2"
```

Explanations:

The intersection (&) is used to put two constraints on 'extension', a) it must have a length as derived from the 'Length' field, b) it must respect the structure specified after the & (i.e., c4 followed by optional c3 followed by an extension placeholder).

The 'spurious extension' is required to allow further extension within the container.

The truncation (//) ensures that the receiver will accept the extension as encoded by an older sender (i.e., with length set to 0, and the extension empty).

The interversion of C3 and C4 is not strictly needed. However, it allows not to include the presence bit of C3 when set to 0 and if it ends the sequence, and avoids to allow the sender to skip C4.

11.2.5.8 Example 8

This example is importing the definition of the Mobile Station Classmark 2 IE from TS 24.008 [13].

In the ASN.1 module:

```
GSMClassMark ::= OCTET STRING
```

In the ECN module:

```
    GSMClassmarkProc ::=
        USER-FUNCTION-BEGIN
            --<ECN.Reference>--
            GSM 04.08, version 7.0.0, Figure 10.7 "GSM 04.08 Mobile Station Classmark 2 information
element", octets 2 to 5
        USER-FUNCTION-END

GSMClassMark ENCODED BY GSMClassmarkProc
```

11.2.5.9 Example 9

Example of encoding definition directly specified using ECN notation. This example defines a specialised encoding for small integer fields using the auxiliary ASN.1 type Int16Encoding.

In the ASN.1 module:

```
    SpecialInt ::= INTEGER (0..15)

Int16Encoding {Dummy} ::= SEQUENCE {
        length      INTEGER (0..MAX),
        value       Dummy}
```

In the ECN module:

```
-- Example encoding definition using native ECN
Int16Encoding.length ::= ENCODING
                {SPACE   {variable-self-delim},
                         -- Represents values 1,2,3,4 etc
                         -- 0 => 1, 10 => 2, 110 => 3, 1110 =>4
                 VALUE   {bit-count-simple-0},
                 LENGTH-DETERMINANT-FOR Int16Encoding.value }
Int16Encoding.value ::= ENCODING
                {SPACE   {variable-min UNITS bits(2)},
                 VALUE   {offset-suppress-zero}
                         -- Will encode the offset for lb
                         -- into the minimum number
                         -- of 2-bits (the number is determined
                         -- by length - see later, with zero
                         -- encoding into zero bits. -- }

-- Association of ECN native definitions with ASN.1 type

SpecialInt ENCODED BY Int16Encoding
```

The encoding of each component is described by fields. The SPACE field specifies the size of the component. The VALUE field specifies the bit pattern that is used to encode the value. The LENGTH-DETERMINANT-FOR field specifies that this component (Int16Encoding.length) is used to calculate the SPACE field of another component (Int16Encoding.value).

11.2.6 Complete Modules

The complete modules summarising the examples above, in conformance with the rules, can be found below.

11.2.6.1 ASN.1 module

```
Sample-ASN1-Module DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
    GSMClassMark ::= OCTET STRING

    B ::= BOOLEAN

    SparseEvenlyDistributedValueSet ::= INTEGER (0|2|4|6|8|10|12|14)

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

    VariableLengthList ::= SEQUENCE (SIZE (0..10)) OF Status

    VariableLengthInteger ::= INTEGER

    VariantRecord ::= CHOICE {
            flag        Flag,   -- The two first alternatives are mostly used
            counter     Counter,
            extEnum     ExtendedEnum,
            status      Status,
            list        VariableLengthList
    }

    ConditionalSized ::= SEQUENCE {
            modulo      INTEGER(1..2048),
            phase       INTEGER(0..2047)
    }

    SpecialisedExtensionV1 ::= SEQUENCE  {
            c1  C1,
            c2  C2,
            extension SEQUENCE {} OPTIONAL
    }
```

```
    SpecialisedExtensionV2 ::= SEQUENCE  {
            c1  C1,
            c2  C2,
            extension SEQUENCE {
                c3  C3 OPTIONAL,
                c4  C4
            } OPTIONAL
    }

    Counter ::= INTEGER (0..255)

    ExtendedEnum ::= ENUMERATED { a, b, c, d, spare4, spare5, spare6, spare7}

    Status ::= INTEGER { idle(0), veryBusy(3) } (0..3)

    Flag ::= BOOLEAN

    C1 ::= OCTET STRING

    C2 ::= BOOLEAN

    C3 ::= INTEGER (0..65535)

    C4 ::= SEQUENCE {
            c1  C1,
            c3  C3
    }

    SpecialInt ::= INTEGER (0..15)

    Int16Encoding {Dummy} ::= SEQUENCE {
            length      INTEGER (0..MAX),
            value       Dummy
    }
END
```

11.2.6.2 ECN module

```
Sample-ECN-Module ENCODING-DEFINITIONS ::=
BEGIN
    IMPORTS GSMClassMark, B, SparseEvenlyDistributedValueSet, SparseValueSet,
        VariableLengthList, VariableLengthInteger, VariantRecord,
        ConditionalSized, SpecialisedExtensionV1.extension, SpecialisedExtensionV2.extension,
        SpecialInt, Int16Encoding
            FROM Sample-ASN1-Module;

    -- Example encoding definition using GSM Mobile Station Classmark 2
    GSMClassmarkProc ::=
        USER-FUNCTION-BEGIN
            --<ECN.Reference>--
                        GSM 04.08, version 7.0.0, Figure 10.7 "GSM 04.08 Mobile Station Classmark 2
information element", octets 2 to 5
        USER-FUNCTION-END


    -- Example encoding definition using CSN.1
    CSN1Proc ::=
        USER-FUNCTION-BEGIN
            --<ECN.Encoding CSN1>--
            IMPORTS
                Flag, Counter, ExtendedNum, Status, VariableLengthList,
                C4, C3
            FROM Sample-ASN1-Module;

            <SpecialBoolean> ::= 0 | 1;

            <SparseEvenlyDistributedValueSet> ::= bit(3);
                -- Representation: This represents the integer equal to
                -- half the binary encoding of the field
                -- e.g., 010 encodes integer 4

            <SparseValueSet> ::= bit(3) exclude {110 | 111};
                -- Representation :
                -- 0 => 000
```

```
            -- 3 => 001
            -- 5 => 010
            -- 6 => 011
            -- 8 => 100
            -- 11 => 110

        <VariableLengthList> ::=
            <Length : 1** 0>
            <V : <ASN1.Status>*(len(Length)-1)>;

        <VariableLengthInteger> ::=
            <Length : 1** 0>
            <V : bit*3*(len(Length)-1)>;
        -- This represents the integer encoded in binary by the V field

        <VariantRecord> ::=
            { 00  <flag    : <ASN1.Flag>>
            | 01  <counter : <ASN1.Counter>>
            | 100 <extEnum : <ASN1.ExtendedEnum>>
            | 101 <status> : <ASN1.Status>>
            | 110 <List    : <ASN1.VariableLengthList>>
            };

        <ConditionalSized> ::=
            <modulo : bit(12)>
            <phase : bit*logval(modulo)>;
        -- where logval is the function to the smaller integer higher or
        -- equal to the logarithm in base 2 of 1 plus the integer
        -- encoded in binary in the argument
        -- e.g., logval(0101) = 3
        --       logval(00) = 0
        --       logval(10) = 2
        -- this can be also described as the position of the last '1' in
        -- the argument, starting from the end

        <Empty Extension> ::=
            <Length : <Extension Length>>
            bit* lval(Length) &
            {<SpuriousExtension : bit(*) = null>;

        <Extension Length> ::=
            <L:0> |             -- lval = 0
            1   <L : bit(3) - 111> |    -- lval = val(L) + 1
            1111 <L : bit(4)>;          -- lval = 8*val(L)+8

        < Extension of SpecialExtensionV2 > ::=
            <Length : <Extension Length>>
            bit* lval(Length) &
            {   <c4 : <ASN1.C4>>
                {0 | 1 <c3 : <ASN1.C3>>}
                <Spurious Extension : bit(*) = null>
            }//;
        ;
    USER-FUNCTION-END

-- Example encoding definition using native ECN
Int16Encoding.length ::= ENCODING
                {SPACE   {variable-self-delim},
                         -- Represents values 1,2,3,4 etc
                         -- 0 => 1, 10 => 2, 110 => 3, 1110 =>4
                 VALUE   {bit-count-simple-0},
                 LENGTH-DETERMINANT-FOR Int16Encoding.value }
Int16Encoding.value ::= ENCODING
                {SPACE   {variable-min UNITS bits(2)},
                 VALUE   {offset-suppress-zero}
                         -- Will encode the offset for lb
                         -- into the minimum number
                         -- of 2-bits (the number is determined
                         -- by length - see later, with zero
                         -- encoding into zero bits. -- }

-- Association of CSN.1 encoding definitions with ASN.1 types

GSMClassMark ENCODED BY GSMClassmarkProc

B ENCODED BY CSN1Proc."SpecialBoolean"

SparseEvenlyDistributedValueSet ENCODED BY
```

```
        CSN1Proc."SparseEvenlyDistributedValueSet"

    SparseValueSet ENCODED BY CSN1Proc."SparseValueSet"

    VariableLengthList ENCODED BY CSN1Proc."VariableLengthList"

    VariableLengthInteger ENCODED BY CSN1Proc."VariableLengthInteger"

    VariantRecord ENCODED BY CSN1Proc."VariantRecord"

    ConditionalSized ENCODED BY CSN1Proc."ConditionalSized"

    SpecialisedExtensionV1.extension ENCODED BY CSN1Proc."Empty Extension"

    SpecialisedExtensionV2.extension ENCODED BY CSN1Proc." Extension of SpecialExtensionV2"

-- Association of ECN native definitions with ASN.1 type

    SpecialInt ENCODED BY Int16Encoding
END
```

11.2.6.3 Link Module

```
Sample-Link-Module LINK-DEFINITIONS ::=
BEGIN
    Sample-ASN1-Module ENCODED BY perUnaligned WITH Sample-ECN-Module
END
```

12 Guidelines involving different specification parts

12.1 Correction of inconsistencies between tabular and ASN.1 in RRC

This subclause includes guidelines describing how to correct inconsistencies between the tabular description and the ASN.1 message specifications of a protocol. The guidelines apply when the inconsistency is not corrected in a straightforward manner.

In some cases clarification should be added to the procedure specification. If the error is procedure specific, this clarification should be added to the procedure specification. Otherwise it should be added to section specifying the general actions upon reception of the concerned IE In case the message is sent from UE to UTRAN, the procedure specification part of the solution may not be needed because RRC specification focuses on UE requirements.

In case comments should be added to the tabular description, these are to be included in the semantics description column.

12.1.1 Correcting the "need" of an IE

12.1.1.1 IE is optional in ASN.1 while it is correctly specified as mandatory in the tabular

The correction should involve the following elements:

- Add clarification to procedure specification, that if the information is absent, it shall be considered as a protocol error.
- Align the tabular description with the ASN.1 and add a comment that the IE is required but that the need is set to OP in order to align with the ASN.1.
- Add comments to ASN.1 that the IE should be mandatory in later versions of this message.

12.1.1.2 IE is mandatory in ASN.1 while it is correctly specified as optional in tabular

The correction should involve the following elements:

- Add clarification to procedure specification concerning what information should be sent when the IE is not required from a functional point of view. Furthermore, clarify what shall be done upon reception of this information.
- Align the tabular description with the ASN.1 and add a comment that the IE is not required but that the need is set to MP in order to align with the ASN.1.
- Add comments to ASN.1 that the IE should be optional in later versions of this message.

12.1.2 Removing an IE

12.1.2.1 IE is optional in ASN.1 while it should be absent

The guidelines provided in this subclause apply in case it is not expected that the IE is to be used in a later release.

The correction should involve the following elements:

- Remove IE from procedure description (if applicable).
- Remove IE from tabular description (if applicable).
- In ASN.1, replace IE by dummy with same type. Add comment to ASN.1 that IE shall not be sent and shall be ignored, if received. Also add comment that the IE should be removed in later versions of this message.

NOTE: There is no need for procedure specification additional to the general statement about ignoring of dummy IEs.

12.1.2.2 IE is optional in ASN.1 while the associated functionality should be removed from this release

The guidelines provided in this subclause apply in case the functionality is to be removed from the concerned release while it is likely that it is to be used in a later release.

The correction should involve the following elements:

- Add clarification to procedure specification that IE should not be sent and that if received, the configuration is considered either as invalid or as undefined is in this version.
- Add comment to tabular description that IE is not used in this version of the specification.
- Add comment to ASN.1 that, in this version of the specification, IE shall not be sent and shall be ignored, if received.

Annex A:
Usage of ASN.1

NOTE: The text in this annex should be seen as illustration of how ASN.1 can be used and do not necessarily reflect how ASN.1 is used in the RAN specifications.

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 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 subclauses A.2.2 and A.3.10).
- Default values (see the subclauses A.2.2 and A.3.10).
- Comprehension required (see the subclause A.2.4).
- Inter/Intra IE dependency (see the subclause A.3.10).
- Specification of partial decoding (see the subclause A.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 contents description 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**.

Identifiers that could be keywords of another language (e.g.: SDL, C, ASN.1, JAVA, C++, …) should be avoided.

A.1 Message level

A.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.

IE is a list of components.

Example: A protocol layer XYZ contains three messages: A, B and C. The structure of the messages is as presented in the figure A.1.1.1.

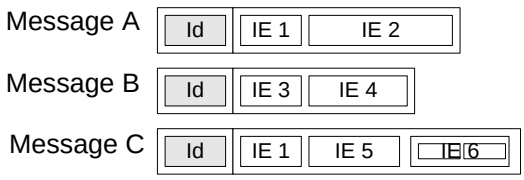


Figure A.1.1.1: Three example messages

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.

A.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 that 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 UNIQUE,
    &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 could be interpreted as the following type:

```
XYZ-Message ::= SEQUENCE {
```

```
    id      MessageId,  -- 1
    contents    SEQUENCE {
        ie1     IE1,
        ie2     IE2
    }
}
```

If *id* is 2 then the type could be interpreted as the following type:

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

A.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". Note that "AUTOMATIC TAGS" is not relevant for PER.

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
```

A.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 cannot be referenced in SDL96 but are allowed in SDL2000. Types and values however can be imported to SDL definitions. The figures below contain some examples about usage of ASN.1 in SDL specifications.

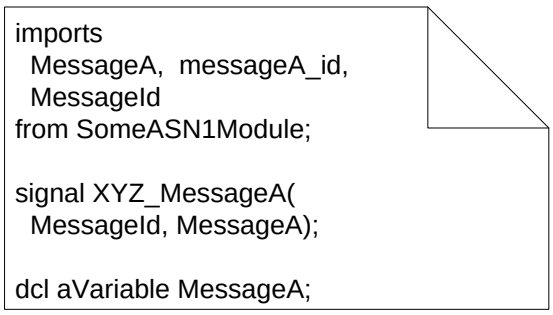


Figure A.1.4.1: Import and use of ASN.1 definitions in SDL

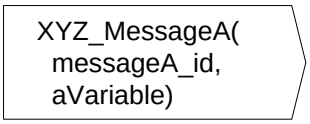


Figure A.1.4.2: Sending of a message id and contents

A.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.

-- Extensions from there
    ExtensionMarker SEQUENCE {} OPTIONAL

}

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

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

-- Extensions from there
    ExtensionMarker SEQUENCE {} OPTIONAL
}

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

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

-- Extensions from there
    ExtensionMarker SEQUENCE {
```

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

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

A.2.1 Message contents

A message contents structure is defined using a sequence type (subclause A.3.10).

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

A.2.2 Optional IEs and default values

An IE can be marked as optional.

COMPACTNESS: Optional IEs shall be after mandatory ones. When ASN.1 is used with PER, this requirement is not relevant.

When the extension "SEQUENCE {} OPTIONAL" is used, the sender shall never indicate that the field is present.

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

```
ie2 IE2 OPTIONAL
```

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
```

A.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
```

A.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)  ,
```

A.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)
```

A.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)
```

A.3 Component level

Information elements consist of components.

The following ASN.1 types shall be used at the component level:

- Boolean (subclause A.3.4);
- Integer (subclause A.3.5);
- Enumerated (subclause A.3.6);
- Bit string (subclause A.3.7);
- Octet string (subclause A.3.8);
- Null (subclause A.3.9);
- Sequence (subclause A.3.10);
- Sequence-of (subclause A.3.11);

- Choice (subclause A.3.12);
- Character string types (subclause A.3.13).

A.3.1 Extensibility

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

A.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 subclause A.2.4.

A.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 subclause A.2.5.

A.3.4 Boolean

EXAMPLE: A simple boolean type.

```
Flag ::= BOOLEAN

setFlag Flag ::= TRUE
```

A.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.

EXAMPLE: 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
```

A.3.6 Enumerated

The EnumerationItem shall not contain a named number (e.g.: foo (3)).

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 on 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
```

A.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
```

A.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    ::= '010203040506070809 0A'H

var VariableLengthOctetStr ::= 'FF'H
```

A.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
}
```

A.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 OPTIONAL
} ( WITH COMPONENT {mand(7), cond PRESENT} | WITH COMPONENT {cond ABSENT} )

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

anotherRecord DerivedRecord ::= {
    flag        TRUE,
    counter 100,
```

```
    bitFlags    '0101'B     -- bitFlags must be present
}
```

A.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 }
```

A.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
```

A.3.13 Restricted character string types

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

It should specified the permitted alphabet for compactness reasons (see examples in PER [5]).

EXAMPLE: Character string types.

```
FixedStr  ::= IA5String (SIZE (10))

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

FixedWStr ::= BMPString (SIZE (10))

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

A.3.14 IEs and ASN.1 modules

If an IE or a component field within an IE is a parameter from another protocol layer then that 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 1: The XYZ protocol message *MessageC* contains an IE, which contains an OPQ protocol layer specific field *parameter1*. Type for the field is imported from that 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 2: The OPQ protocol layer specific module exports *OPQParameter* type so that other modules can refer it.

```
OPQ-DataTypes DEFINITIONS AUTOMATIC TAGS ::=

BEGIN


OPQParameter ::= INTEGER (0..7)

END
```

Annex B:
Handling of DS-41

- Modelling of RRC services is provided by means of primitives.
- RRC CN dependent info.
- In broadcast message, neighbour cells are described the same way as for GSM neighbour cells (i.e.: in the same SystemInformationBlock but with a tag to indicate CN type or RTT).
- In dedicated messages:
  - a transparent container as NAS info is used to carry ANSI-41;
  - for PLMN Id and Identities used by the RRC, the CN Type info is used;
  - NAS binding info is used.
- In Paging messages, a tag to indicate CN type is used.
- Extensions like handover message to Multicarrier is handled the same way as GSM.

Annex C:
Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
12/1999	RP-06	RP-99659	-		Approved at TSG-RAN #6 and placed under Change Control	-	3.0.0
03/2000	RP-07	RP-000048	001	2	Further clarifications on specialised encoding	3.0.0	3.1.0
	RP-07	RP-000048	003	1	Modification of the 'presence' column specification in tabular format, and other editorial modifications	3.0.0	3.1.0
	RP-07	RP-000048	005		Editorial corrections on subclause 11.2	3.0.0	3.1.0
	RP-07	RP-000048	006		Improvement of integers and enumerated, and introduction of reals and octet strings	3.0.0	3.1.0
12/2000	RP-10	RP-000575	007		Extension rules for supporting future releases	3.1.0	3.2.0
03/2001	RP-11	RP-010033	008		Description of backward compatibility consideration rule for RANAP, SABP, RNSAP and NBAP ASN.1	3.2.0	3.3.0
	RP-11	RP-010033	009		Usage of the Version column	3.2.0	3.3.0
	RP-11	RP-010033	010	1	Clean-up	3.2.0	3.3.0
	RP-11	RP-010033	011		Recommendations on the use of the extension mechanism	3.2.0	3.3.0
	RP-11	-	-		Upgrade to Release 4 - no technical change	3.3.0	4.0.0
06/2001	RP-12	RP-010318	015		Clean up	4.0.0	4.1.0
	RP-12	RP-010318	017		Usage of spare values in future releases	4.0.0	4.1.0
	RP-12	RP-010318	019	1	Structure and naming of extensions in ASN.1	4.0.0	4.1.0
	RP-12	RP-010318	021		Addition of Recommendations for Extensions in RANAP, RNSAP, NBAP, and SABP	4.0.0	4.1.0
	RP-12	RP-010318	023		Clean-up with regard to RAN WG3 Practise of Specifying Control Plane Protocols	4.0.0	4.1.0
09/2001	RP-13	RP-010551	025		Guidelines concerning conditions, spares, defaults and correction of inconsistencies	4.1.0	4.2.0
	RP-13	RP-010551	027		Naming convention for non-critical extensions	4.1.0	4.2.0
	RP-13	RP-010551	029		Introduction of procedure specification guidelines specific to RLC	4.1.0	4.2.0
	RP-13	RP-010551	031		RAN3 usage of 1994 ASN.1 feature set	4.1.0	4.2.0
12/2001	RP-14	RP-010768	033		Modulo formula	4.2.0	4.3.0
	RP-14	RP-010768	035		Use of extensions in a backward compatible way	4.2.0	4.3.0
	RP-14	RP-010768	037		Extensions of IE value ranges in tabular	4.2.0	4.3.0
03/2002	RP-15	RP-020075	039		Additional guidelines on ASN.1 comments	4.3.0	4.4.0
	RP-15	RP-020075	041		RRC specific rules for procedure text	4.3.0	4.4.0
12/2002	RP-18	RP-020725	043		Introduction of backwards compatible correction mechanism	4.4.0	4.5.0
09/2003	RP-21	RP-030488	046		Guideline on introducing additional SIB types	4.5.0	4.6.0
03/2004	RP-23	RP-040098	050		Spare Extension in Data Frame	4.6.0	4.7.0
	RP-23	RP-040098	053	2	Guideline on release independent ASN.1 updates	4.6.0	4.7.0
	RP-23	RP-040098	056		Guideline on the use of variable length containers for late extensions	4.6.0	4.7.0
	RP-23	RP-040098	059		Guideline for the naming of extensions to the RRC ASN.1	4.6.0	4.7.0
06/2004	RP-24	RP-040204	062		Empty non-critical extensions	4.7.0	4.8.0