ETSI TR 125 921 V3.1.0 (2000-03)
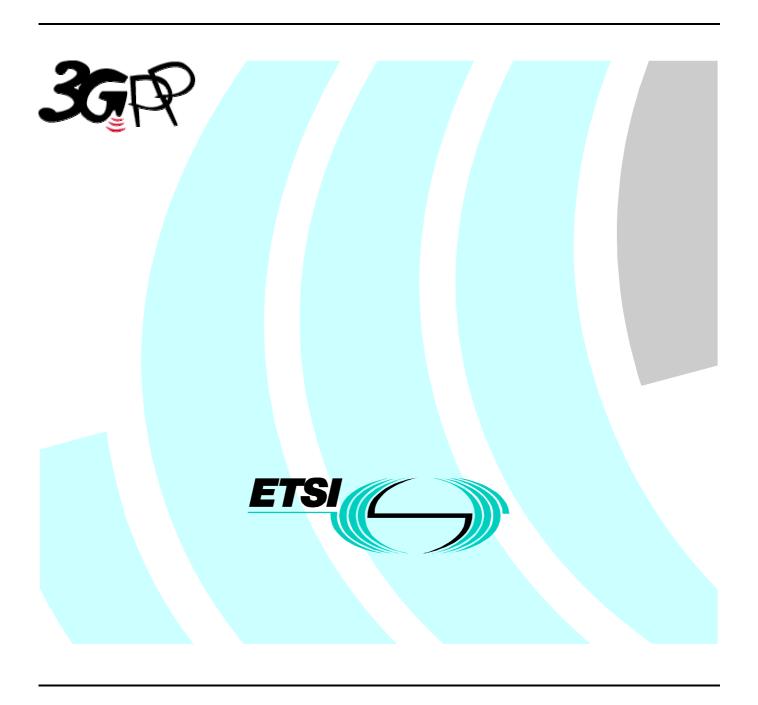
**Universal Mobile Telecommunications System (UMTS);
Guidelines and Principles for protocol description
and error handling
(3G TR 25.921 version 3.1.0 Release 1999)**

Reference
RTR/TSGR-0225921UR1

Keywords
UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00   Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at http://www.etsi.org/tb/status/

If you find errors in the present document, send your comment to:
editor@etsi.fr

Copyright Notification

ETSI

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Report (TR) has been produced by the ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under www.etsi.org/key .

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

2 References

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

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies.

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

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

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

[4]   ITU-T Recommendation X.690: "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: "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 (SDL/ASN.1)".

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

[10]   ISO/IEC 9646-3: "The Tree and Tabular Combined Notation".

[11]   3G TS 21.200: "3GPP Drafting rules".

3 Void

4 Principles to ensure compatibility

4.1 Introduction

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

4.2 Level 1 of principles: Protocol level

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

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

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

4.3 Level 2 of principles: Message level

4.3.1 New messages

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

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

Such a mechanism is not required inside the network part.

4.3.2 Partial decoding

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

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

4.4 Level 3 of principles: Information element level

4.4.1 New IE

New elements shall generally be discarded when not understood.

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

4.4.2 Optional IE

Optional IE should be located after mandatory ones.

4.4.3 Adding mandatory IE

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

4.4.4 Missing optional IE

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

See also missing values in Values level.

4.4.5 Comprehension required

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

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

4.4.6 Partial Decoding

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

4.5 Level 4 of principles: Values level

4.5.1 Reserved values and spare fields

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

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

4.5.2 Unspecified values

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

4.5.3 Missing optional value

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

4.5.4 Extension of value set

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

1) Infinite extension of a value set. Example: The first version of a data field may contain only values 0-3. In the future the field may contain any positive integer value.

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

4.6 Decision of TSG RAN WG2

TSG RAN WG2 decided to use version number for MAC and RLC protocol layers.

TSG RAN WG2 is not able to decide yet what is the best to ensure compatibility when extending RRC in future releases.

5 Message Sequence Charts

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

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

6 Specification and Description Language

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

The groups themselves should decide how SDL is used.

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

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

In TSG RAN WG2, release 99 of the specifications shall not use SDL for the normative part of the specifications. This may be revisited in future releases.

7 Protocol procedure specification rules

7.1 General

- The procedure specification shall be made using text and verbal forms.

- Words "shall", "should" and "may" are used in conformance with [10] Annex E.

- All normal cases shall be covered. Normal cases are straightforward cases, branches of procedures and combinaisons of procedures.

- The way to describe procedures is the following:

  Protocol errors (global to the protocol layer).

  Error handling (global to the protocol layer).

  …

  1. Procedure <Procedure Name>;

     list of normal cases;

     Protocol errors (specific to the procedure);

     Error handling (specific to the procedure).

- Redundancy/duplication shall be avoided, in order to avoid problems with later CR, even if this makes the specification initially less readable.

- Mutual crossreferencing shall be introduced: section X that is referred to in section Y should also *say* that it is referred to in section Y.

- 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.

- The chapter "Default actions upon reception of an IE" is used to avoid duplication of text, this chapter is put at the beginning of the "Message contents to use" text.

- When optional (or conditional) IEs are possible in a given message, the meaning of the presence (ie: which «function» are activated with the given IE) shall be specified in the procedure.

- The formal values of the IE, e.g., "TRUE" or "FALSE" rather than the coded value, e.g. "1" or "0" shall be used.

- Requirements on the content of a message at the sending entity is put before analysis of the message at the receiving entity.

  - References to IEs that are parts of another IE is allowed. The notation shall be changed to the so-called "dot-notation" for references to IEs that are parts of another IE.

  - Names of IEs shall be put between "<IE Name>" quotes, where <IE Name> is the exact name from the tabular format.

  - Square brackets [ ] shall be used for addressing one element of a list.

  - When referring to a message, "<MESSAGE NAME>" *message* shall be used. Message names are always in upper cases and the word message follows the message name.

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.

- "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.

- UE performance requirements are considered to be TSG RAN WG2 work. These must be specified only if they are testable.

7.3 Handling of DS-41

- Modeling 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 (ie: 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;

  Routing info is FFS.

- In Paging messages, a tag to indicate CN type is used.

- Extensions like handover message to Multicarrier is handled the same way as GSM.

- Ciphering is FFS.

8 Message specification

8.1 Summary of what has been agreed

1) use subset of ASN.1 (compatible with Z.105) for definition of message contents description of protocol messages;

2) there is a need for a default encoding, which can be applied in most cases;

3) there is a need for a special encoding e.g. by means of CSN.1;

4) how to link the message contents description to the different encoding rules needs to be specified;

5) ASN.1 definitions can be used within SDL and TTCN parts of the specifications.

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; and

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

8.3 Logical description

The logical description of messages shall be done using tabular format specified in clause 8 of this 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 which transform a sequence of received bits into a sequence of IEs which can be utilised by the protocol machine. CSN.1 may be used. A link between message contents description and transfer syntax needs to be specified.

8.6 Efficiency/Compactness

The transfer syntax should allow to minimise 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.

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

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.

9.1　　Tabular description of messages and IEs

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 5 columns, labelled and presented as shown below.

IE/Group Name	Need	Multi	Type and reference	Semantics description

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

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

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.

The "Type and reference" column is not filled in the case of a group line and must be filled for "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 followed by the key word IE, the whole enclosed between quotes [or in italics] to refer to the IE or the group of IEs in the procedural description.

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

9.1.1.2.1 Need and multiplicity (Multi) columns

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

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.

A value for that information is needed (presence needed) or unacceptable (absence needed) when some conditions are met that can be evaluated on the sole basis of the content of the message.

If conditions for presence needed are specified, the transfer syntax must allow for the presence of the information. If the transfer syntax allows absence, absence when the conditions for presence are met leads to an error diagnosis.

If conditions for absence needed are specified, the transfer syntax must allow to encode the absence. If the information is present and the conditions for absence are met, an error is diagnosed.

When neither conditions for presence or absence are met, the information is treated as optional, as described for 'OP'.

CH   Conditional on history.

A value for that information is needed (presence needed) or unacceptable (absence needed) when some conditions are met that must be evaluated on the basis of information obtained in the past (e.g., from messages received in the past from the other party).

If conditions for presence needed are specified, the transfer syntax must allow for the presence of the information. If the transfer syntax allows absence, absence when the conditions for presence are met leads to an error diagnosis.

If conditions for absence needed are specified, the transfer syntax must allow to encode the absence. If the information is present and the conditions for absence are met, an error is diagnosed.

When neither conditions for presence or absence are met, the information is treated as optional, as described for 'OP'.

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
Name	MP			
Name	MD			(default value is indicated)

The multiplicity column must be left empty.

For an IE not belonging to a group MP indicates that one and only one copy of 'Name IE' is necessary in the message.

For a group not belonging to another group, MP means that one and only one copy 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 one and only one copy of the 'Name group' or 'Name IE' is necessary in the embedding group.

For an IE not belonging to a group MD indicates that one and only one value for information 'Name IE' is necessary in the message, and that a special value (the default value) exists and is mentioned in the 'Semantics description' column.

For a group not belonging to another group, MD means that one and only one value for information structure 'Name group' is necessary in the message, and that a special value (the default value) exists 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 one and only one value 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.

9.1.1.2.1.2   Optional

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

The multiplicity column is empty.

This indicates that 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
	CV *cond*			
	CH			

The multiplicity column is empty.

CV indicates that the requirement for presence of absence of the IE or group of IE depends on the value of some other IE or IEs, and/or on the message flow (e.g., channel, SAP). 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.

When condition is met may means that IE is:

- Mandatorily present.

- Mandatorily absent.

- Optional.

- Absent, but optional (this is meaningful only for extension).

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
Choice *name*				
>*Name1*				
>*Name2*				

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

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

If additional conditions (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 copy 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
Name		nn..pp		
Name		nn..indefinite		
Name		nn..sym2		
Name		sym1..pp		
Name		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 copies 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 copy; numbering starts by 1.

The *nn..pp* case indicates that the number of copies is between nn and pp, inclusively. This means that *nn* copies are necessary in the message, that additional *pp-nn* copies are optional and meaningful, and that copies 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 a MP indication used instead. Similarly, the 0..1 case should be avoided and replaced by an OP indication.

The *nn..indefinite* case indicates that the number of copies is *nn* or greater. This means that *nn* copies are necessary in the message, and that additional copies 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 copies.

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

The 'Need' column is set to CV or CH followed by a condition name to indicate that the number of necessary or optional copies is conditional to the value of some other IE or IEs, or on the flow (CV case) or to information obtained in the past (CH case). An explanatory clause describes the condition. Otherwise, the column is left empty.

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;

c) A reference to a subclause of this document, with possibly additional information as described.

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 strucured 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
Choice systemtype				
>FDD				
>>element1	MP			
>>element2	OP			
>TDD				
>>element3	OP			
>>element4	MP			

9.1.1.3   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/group is to be included or not, or the value of the symbolic name.

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.

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

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

9.2   Basic types

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

9.2.1 Enumerated

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

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.

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
			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
			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 arithmetics. 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
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
			Bit string (*nn*)	

Where *nn* is a positive non null number indicating the number of bits in the string.

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
			Octet string (*nn*)	

Where *nn* is a positive non null number indicating the number of octets in the string.

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
			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 arithmetics. 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 $b1+k*st \leq b2$. 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 features that ASN.1 provides include specification of:

- Extensibility (both structural and extension of value set).

- Optional IEs and values (see the clauses 10.2.2 and 10.3.10).

- Default values (see the clauses 10.2.2 and 10.3.10).

- Comprehension required (see the clause 10.2.4).

- Inter/Intra IE dependency (see the clause 10.3.10).

- Specification of partial decoding (see the clause 10.2.5).

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 (eg: SDL, C, ASN.1, JAVA, C++, …) should be avoided.

10.1 Message level

10.1.1 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 10.1.1.1.

Figure 10.1.1.1: 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.

10.1.2   Message definition

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

1. A structure for the table .

2. The table itself.

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

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

```
XYZ-MESSAGE ::= CLASS {
    &id      MessageId 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
    }
}
```

10.1.3 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
```

10.1.4 Messages and SDL

The identifiers *messageA-id*, *MessageA*, *messageB-id*, etc. can be used in descriptive SDL when protocol behaviour is specified. Note that classes and objects can not 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.

```
imports
  MessageA, messageA_id,
  MessageId
from SomeASN1Module;

signal XYZ_MessageA(
  MessageId, MessageA);

dcl aVariable MessageA;
```

Figure 10.1.4.1: Import and use of ASN.1 definitions in SDL

```
XYZ_MessageA(
  messageA_id,
  aVariable)
```

Figure 10.1.4.2: Sending of a message id and contents

10.2 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
```

10.2.1 Message contents

A message contents structure is defined using a sequence type (10.3.10).

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

10.2.2 Optional IEs and default values

An IE can be marked as optional.

COMPACTNESS: Optional IEs shall be after mandatory ones. When using ASN.1 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
```

10.2.3 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
```

10.2.4 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)   ,
```

10.2.5   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)
```

10.2.6   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)
```

10.3   Component level

Information elements consist of components.

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

- Boolean   (10.3.4)

- Integer   (10.3.5)

- Enumerated   (10.3.6)

- Bit string   (10.3.7)

- Octet string   (10.3.8)

- Null   (10.3.9)

- Sequence   (10.3.10)

- Sequence-of   (10.3.11)

- Choice (10.3.12)

  - Character string types (10.3.13)

10.3.1 Extensibility

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

10.3.2 Comprehension required

"Comprehension required" can be applied to components of sequence types, alternatives of choice types and elements of sequence-of types. See 10.2.4.

10.3.3 Partial decoding

"Partial decoding" can be applied to components of sequence types, alternatives of choice types and elements of sequence-of types. See 10.2.5.

10.3.4 Boolean

Example: A simple boolean type.

```
Flag ::= BOOLEAN

setFlag Flag ::= TRUE
```

10.3.5 Integer

An integer type should be constrained.

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

Named numbers can be associated with an integer type.

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

- A value set that specifies the values that can be sent in the current protocol version.

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

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

> *Send(<value set>)*

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

Examples: Integer types and values.

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

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

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

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

-- Send values 0..3 but be prepared to receive values 0..15.
```

```
 Extensible       ::= INTEGER (0..15)(CONSTRAINED BY {-- Send(0..3) --})

initialCounter Counter      ::= 0

zero          SparseValueSet ::= 0

initialStatus  Status        ::= idle
```

10.3.6   Enumerated

The EnumerationItem shall not contain a named number (eg: 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
```

10.3.7   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
```

10.3.8   Octet string

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

Example: Octet string types and values.

```
FixedLengthOctetStr    ::= OCTET STRING (SIZE (10))

VariableLengthOctetStr ::= OCTET STRING (SIZE (0..10))

UpperLayerPDUSegment   ::= OCTET STRING (SIZE (1..512))
```

```
 fix FixedLengthOctetStr    ::= '0102030405060708090A'H

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

10.3.9   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
}
```

10.3.10   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
}
```

10.3.11 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 }
```

10.3.12 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
```

10.3.13 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))
```

10.3.14 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: 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: 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
```

11 Message transfer syntax specification

11.1 Selection of transfer syntax specification method

Basic unaligned PER and possible use of specialised encoding is choosen.

11.2 Specialised encoding

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 1: 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, incompletness 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 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 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 to optimize 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 to specify 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 place-holder. 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 place-holder).

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 GSM 04.08.

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
```

Annex A:
Change history

Change history					
TSG-RAN#	Version	CR	Tdoc RAN	New Version	Subject/Comment
RAN_06	-	-	RP-99659	3.0.0	(12/99) Approved at TSG-RAN #6 and placed under Change Control
RAN_07	3.0.0	001	RP-000048	3.1.0	(03/00) Further clarifications on specialised encoding
RAN_07	3.0.0	003	RP-000048	3.1.0	Modification of the 'presence' column specification in tabular format, and other editorial modifications
RAN_07	3.0.0	005	RP-000048	3.1.0	Editorial corrections on subclause 11.2
RAN_07	3.0.0	006	RP-000048	3.1.0	Improvement of integers and enumerated, and introduction of reals and octet strings

History

Document history		
V3.1.0	March 2000	Publication