TSG-RAN Meeting #7
Madrid, Spain, 13 – 15 March 2000

RP-000048

Title: **Agreed CRs to TR 25.921**

Source: **TSG-RAN WG2**

Agenda item: **6.3.3**

Doc-1st-	Spec	CR	Rev	Subject	Cat	Version	Versio
R2-000254	25.921	001	2	Further clarifications on specialised	D	3.0.0	3.1.0
R2-000520	25.921	003	1	Modification of the 'presence' column	D	3.0.0	3.1.0
R2-000544	25.921	005		Editorial corrections on section 11.2	D	3.0.0	3.1.0
R2-000619	25.921	006		Improvement of integers and	D	3.0.0	3.1.0

3GPP TSG-RAN Meeting #7
Madrid, Spain, 13 - 15 March 2000

***Document* R2-000254**
e.g. for 3GPP use the format TP-00xxxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.921	CR	001r2	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to:	TSG-RAN #7	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑		for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from: ftp://ftp.3gpp.org/Information/CR-Form-v2.doc*

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 2000-01-21

Subject: Further clarifications on specialised encoding

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction	
A	Corresponds to a correction in an earlier release	
B	Addition of feature	
C	Functional modification of feature	
D	Editorial modification	X

Release:

Phase 2	
Release 96	
Release 97	
Release 98	
Release 99	X
Release 00	

Reason for change: Alignment of specialised encoding on the ECN approach. Restructuration of clause. Addition of examples.

Clauses affected: 11.2 is replaced, revision marks are relative to revision 1 of the CR

Other specs Affected:

Other 3G core specifications		→ List of CRs:
Other GSM core specifications		→ List of CRs:
MS test specifications		→ List of CRs:
BSS test specifications		→ List of CRs:
O&M specifications		→ List of CRs:

Other comments:


help.doc

<--------- double-click here for help and instructions on how to create a CR.

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 approachs 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 mentionned 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 values of a type can be represented with it, i.e. there shall be a mapping from each abstract value to an encoded value. 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 numberof components.

In the ASN.1 module :

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

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.

11.2.4.4 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> ::=
   construct
    <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.

11.2.4.5 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.

11.2.4.6 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
```

11.2.4.7 Example 7

~~WARNING : NOT ALIGNED WITH COMPLETE MODULES : DISCUSSION NEEDED~~

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>>
    <Extension : 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>>
    <Extension : 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 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 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, SpecialisedExtensionV2,
        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>>
            <Extension : 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>>
            <Extension : 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
```

3GPP TSG-RAN Meeting #7
Madrid, Spain, 13 - 15 March 2000

1?

***Document* R2-000520**
e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.921	CR	003r1	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to:	TSG-RAN #7	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑		for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from:* ftp://ftp.3gpp.org/Information/CR-Form-v2.doc

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 28th Feb. 2000

Subject: Modification of the 'presence' column specification in tabular format, and other editorial modifications

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction		**Release:**	Phase 2	
A	Corresponds to a correction in an earlier release			Release 96	
B	Addition of feature			Release 97	
C	Functional modification of feature			Release 98	
D	Editorial modification	X		Release 99	X
				Release 00	

Reason for change: Distinction of more cases, for better accuracy. Alignment on usage.

Clauses affected: 9

Other specs affected:

Other 3G core specifications		→ List of CRs:
Other GSM core specifications		→ List of CRs:
MS test specifications		→ List of CRs:
BSS test specifications		→ List of CRs:
O&M specifications		→ List of CRs:

Other comments:

help.doc

<--------- double-click here for help and instructions on how to create a CR.

9 Usage of tabular format

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

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

9.1 Tabular description of messages and IEs

9.1.1 Message description

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

A message is described with, in this order:

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

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	~~Presence~~Need	Multi	~~IE~~ Type and reference	Semantics description

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

Indentations are explicitely written with the character ">", one per level of indentation. Indentations of lines can be found in IE/Group Name and ~~IE Type~~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 ~~Presence~~ Need and ~~Mult~~Multi columns are meaningless and should be left empty.

The "~~IE type~~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 ~~Presence~~ Need and multiplicity (~~Mult~~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 mentionned (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	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~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 ~~this~~ MP indicates that one and only one copy of 'Name IE' is necessary in the message.

For a group not belonging to another group, ~~this~~ 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, ~~this~~ 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 oninformation obtained in the past.

9.1.1.2.1.2 Optional

IE/Group Name	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~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	Presence	~~Mult~~Multi	~~IE Type~~Type and reference	Semantics description
	CV *cond*			
	CH			

The multiplicity column is empty.

~~This~~ 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, ~~T~~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	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~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 ~~Presence~~ Need and ~~Mult~~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	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~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 presence column can be empty CVor 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 ~~presence~~ '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 ~~IE type~~Type and reference column

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

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

a) A reference to a sub-clause in the Information Element Description clause in the same document; Typically the sub-clause number and titles are given, and if possible this should be a hypertext link;

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

c) A reference to a sub-clause of this document, 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 presence 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	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~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 sub-clauses needed to elaborate conditions and symbolic names (e.g., range bounds). There must be one explanatory clause for each named condition, and for each symbolic name. The text must give the information sufficient to decide whether the IE/group is to be included or not, or the value of the symbolic name.

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

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

IE/Group Name	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~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	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~Type and reference	Semantics description
			Enumerated (*c1*, *c2*, *c3*)	
			Enumerated (1..*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, *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	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~Type and reference	Semantics description
			Boolean	

This is shorthand for:

			Enumerated (*False, True*)	

9.2.3 Integer

The different lines below indicate different alternatives.

IE/Group Name	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~Type and reference	Semantics description
			Integer	
			Integer (*nn..pp*)	
			Integer (*nn*..indefinite)	
			Integer (*sym1..pp*)	
			Integer (*nn..sym2*)	
			Integer (*sym1..sym2*)	

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

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

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

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.

9.2.4 Bit string

IE/Group Name	~~Presence~~Need	~~Mult~~Multi	~~IE Type~~Type and reference	Semantics description
			Bit string (*nn*)	

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

3GPP TSG-RAN Meeting #7
Madrid, Spain, 13 - 15 March 2000

Document ***R2-000544***

e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.921	CR	005	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to: TSG-RAN #7 for approval [X] strategic
list expected approval meeting # here ↑ for information [ ] non-strategic [ ] *(for SMG use only)*

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from:* ftp://ftp.3gpp.org/Information/CR-Form-v2.doc

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 28th Feb. 2000

Subject: Editorial corrections on section 11.2

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction	[ ]	**Release:** Phase 2	[ ]
A	Corresponds to a correction in an earlier release	[ ]	Release 96	[ ]
B	Addition of feature	[ ]	Release 97	[ ]
C	Functional modification of feature	[ ]	Release 98	[ ]
D	Editorial modification	[X]	Release 99	[X]
			Release 00	[ ]

Reason for change:

Clauses affected: 11.2

Other specs Affected:

Other 3G core specifications	[ ]	→ List of CRs:
Other GSM core specifications	[ ]	→ List of CRs:
MS test specifications	[ ]	→ List of CRs:
BSS test specifications	[ ]	→ List of CRs:
O&M specifications	[ ]	→ List of CRs:

Other comments:

help.doc

<--------- double-click here for help and instructions on how to create a CR.

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 approachs 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 numberof 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.

11.2.45.4 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> ::=
    construct
    <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.

11.2.45.5 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.

11.2.45.6 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 modulo   INTEGER(1..2048),
    Phasephase  INTEGER(0..2047)}
```

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

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

-- 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
```

11.2.45.7 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>>
    <Extension : 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>>
    <Extension : 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 ~~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 ~~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 modulo : bit(12)>
                <Phase phase : bit*logval(Modulomodulo)>;
            -- 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>>
                <Extension : 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>>
                <Extension : 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
```

3GPP TSG-RAN Meeting #7
Madrid, Spain, 13 - 15 March 2000

Document 1? ***R2-000619***

e.g. for 3GPP use the format TP-99xxx or for SMG, use the format P-99-xxx

CHANGE REQUEST

Please see embedded help file at the bottom of this page for instructions on how to fill in this form correctly.

25.921	CR	006	Current Version:	3.0.0

GSM (AA.BB) or 3G (AA.BBB) specification number ↑ *↑ CR number as allocated by MCC support team*

For submission to:	TSG-RAN #7	for approval	X	strategic		(for SMG use only)
list expected approval meeting # here ↑		for information		non-strategic		

Form: CR cover sheet, version 2 for 3GPP and SMG *The latest version of this form is available from:* ftp://ftp.3gpp.org/Information/CR-Form-v2.doc

Proposed change affects: *(at least one should be marked with an X)* (U)SIM [ ] ME [X] UTRAN / Radio [X] Core Network [ ]

Source: TSG-RAN WG2 **Date:** 2nd Mar. 2000

Subject: Improvement of integers and enumerated, and introduction of reals and octet strings

Work item:

Category: *(only one category shall be marked with an X)*

F	Correction		**Release:**	Phase 2	
A	Corresponds to a correction in an earlier release			Release 96	
B	Addition of feature			Release 97	
C	Functional modification of feature			Release 98	
D	Editorial modification	X		Release 99	X
				Release 00	

Reason for change: More flexibility, and alignment on effective use

Clauses affected: 9

Other specs affected:

Other 3G core specifications	[ ]	→ List of CRs:
Other GSM core specifications	[ ]	→ List of CRs:
MS test specifications	[ ]	→ List of CRs:
BSS test specifications	[ ]	→ List of CRs:
O&M specifications	[ ]	→ List of CRs:

Other comments:

help.doc

<--------- double-click here for help and instructions on how to create a CR.

9 Usage of tabular format

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

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

9.1 Tabular description of messages and IEs

9.1.1 Message description

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

A message is described with, in this order:

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

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	Presence	Mult	IE Type and reference	Semantics description

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

Indentations are explicitely written with the character ">", one per level of indentation. Indentations of lines can be found in IE/Group Name and IE 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 Presence and Mult columns are meaningless and should be left empty.

The "IE type and reference" column is not filled in the case of a group line and must be filled for "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 Presence and multiplicity (Mult) 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.

9.1.1.2.1.1 Mandatory

IE/Group Name	Presence	Mult	IE Type and reference	Semantics description
Name	M			

The multiplicity column must be left empty.

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

For a group not belonging to another group, this 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, this 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.

9.1.1.2.1.2 Optional

IE/Group Name	Presence	Mult	IE Type and reference	Semantics description
Name	O			

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	Presence	Mult	IE Type and reference	Semantics description
	C *cond*			

The multiplicity column is empty.

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

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	Presence	Mult	IE 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 Presence and Mult 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	Presence	Mult	IE Type and reference	Semantics description
Name		nn..pp		
Name		nn..indefinite		
Name		nn..sym2		
Name		sym1..pp		
Name		sym1..sym2		
Name	C cond	nn ..pp		
Name	C cond	nn..indefinite		
Name	C cond	nn..sym2		
Name	C cond	sym1..pp		
Name	C cond	sym1..sym2		

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

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 M indication used instead. Similarly, the 0..1 case should be avoided and replaced by an O indication.

The *nn*..indefinite case indicates that the number of copies is *nn* or greater. 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 presence column is set to C 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. An explanatory clause describes the condition. Otherwise, the column is left empty.

9.1.1.2.2 IE type and reference column

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

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

a) A reference to a sub-clause in the Information Element Description clause in the same document; Typically the sub-clause number and titles are given, and if possible this should be a hypertext link;

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

c) A reference to a sub-clause of this document, 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 presence 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	Presence	Mult	IE Type and reference	Semantics description
Choice systemtype				
>FDD				
>>element1	M			
>>element2	O			
>TDD				
>>element3	O			
>>element4	M			

9.1.1.3 Explanatory clauses

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

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

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

IE/Group Name	Presence	Mult	IE 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	Presence	Mult	IE Type and reference	Semantics description
			Enumerated (*c1*, *c2*, *c3*)	
			Enumerated (*x*1..*xn*)	

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	Presence	Mult	IE 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	Presence	Mult	IE 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. ~~The presence column must be left empty.~~

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

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

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	IE 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	Presence	Mult	IE 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	Presence	Mult	IE 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	Presence	Mult	IE 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.