NON-CRITICAL 3GPP RRC MESSAGE EXTENSIONS

1. Background

[1] and [2] contain examples and proposals on how ASN.1 definitions for the RRC messages specified in [3] could be modified when new non-critical information elements are added to RRC messages. The main point in [1] and [2] is that for maintenance and clarity reasons new IEs should be added in a place where they logically belong, even if this means addition of new IEs in a middle of a message. It should then be encoding specification's responsibility to order encoding of IEs such that encoding of non-critical extensions is moved in the end of a message. [4] contains a proposal on how these requirements can be fulfilled with use of ASN.1 extensibility mechanism and version brackets.

2. Wanted functionality

2.1. ASN.1 definitions with normal ASN.1 extensibility

The following simple example illustrates the wanted situation. The example is a simplified version of the example presented in [2], but the basic structure and ideas are the same.

The module "Example" contains an ASN.1 type "IE". The first version of the type is specified as follows:

```
-- v.1
Example DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

IE ::= SEQUENCE {
    alternatives    CHOICE {
          a1              SEQUENCE {
               list1           List1      OPTIONAL
          },
          a2              SEQUENCE {
               list2           List2      OPTIONAL
          }
    }
}

List1 ::= SEQUENCE (SIZE (1..8)) OF Elem1

List2 ::= SEQUENCE (SIZE (1..8)) OF Elem2

Elem1 ::= SEQUENCE {
    a     INTEGER (0..1)
}

Elem2 ::= SEQUENCE {
    b     INTEGER (0..1)
}

END
```

When the protocol evolves there is a need to extend the IE structure. The locations for non-critical extensions are deep in the structure. The following ASN.1 module contains the extensions. The location of the extensions is were they logically belong to. Extension versions are shown in comments.

```
--v.N
Example DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

IE ::= SEQUENCE {
    alternatives    CHOICE {
         a1              SEQUENCE {
              list1           List1      OPTIONAL,
              ... ,
              [[   -- v.2 extensions
              ext-list1       Ext-List1  OPTIONAL,
              ext-comp        BOOLEAN    OPTIONAL
              ]]
         },
         a2              SEQUENCE {
              list2           List2      OPTIONAL,
              ... ,
              [[   -- v.2 extensions
              ext-list2       Ext-List2  OPTIONAL
              ]]
         }
    }
}

List1 ::= SEQUENCE (SIZE (1..8)) OF Elem1

List2 ::= SEQUENCE (SIZE (1..8)) OF Elem2

Ext-List1 ::= SEQUENCE (SIZE (1..8)) OF Ext-Elem1

Ext-List2 ::= SEQUENCE (SIZE (1..8)) OF Ext-Elem2

Elem1 ::= SEQUENCE {
    a    INTEGER (0..1),
    ... ,
    [[   -- v.4 extensions
    x    INTEGER (0..3) OPTIONAL
    ]]
}

Elem2 ::= SEQUENCE {
    b    INTEGER (0..1)
}

Ext-Elem1 ::= SEQUENCE {
    a    INTEGER (0..3),
    ... ,
    [[   -- v.3 extensions
    groupOfExtensions    SEQUENCE {
         y    INTEGER (0..3),
         z    INTEGER (0..3)
    } OPTIONAL
    ]]
}

Ext-Elem2 ::= SEQUENCE {
    b    INTEGER (0..3)
}

END
```

2.2. Encoding for non-critical extensions

The requirement for encoding of extensions are:

- They shall be smaller that encoding produces by pure PER.
- They shall be backwards compatible.

The requirements can be fulfilled with the following modifications:

- Extensions are put at the end of the encoding. The order of extensions shall be based first on the version in which the extensions have been introduced and second on the order in which extensions appear in the value to be encoded (i.e., v.2 non-critical

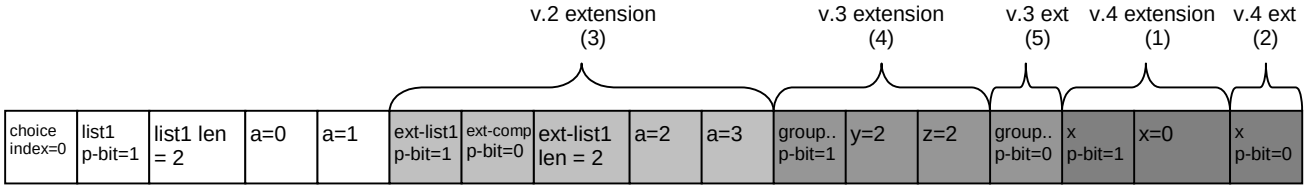
extensions before v.3 non-critical extensions, which in turn appear before v.4 non-critical extensions).

- PER generated auxiliary information like length fields and extension bits are omitted. This means that unknown trailing non-critical extensions in a message are silently ignored.

Given the following ASN.1 value:

```
exampleValue IE ::= {
    alternatives a1 : {
          list1 {
                {
                      a 0,
                      x 0                    -- v.4 extension present (1)
                },
                {
                      a 1
                                             -- v.4 extension absent  (2)
                }
          },
          ext-list1 {                        -- v.2 extension present (3)
                {
                      a 2,
                      groupOfExtensions {    -- v.3 extension present (4)
                            y 2,
                            z 2
                      }
                },
                {
                      a 3
                                             -- v.3 extension absent  (5)
                }
          }
    }
}
```

the following encoding fulfils the requirements:



In the figure, older extensions are shown in light gray and new extensions in darker gray.

The order of extensions is determined:

a) by the order in which the extensions have been introduced in the specification (e.g., extension "x" is introduced in a later version than extension "y" and thus encoding of "x" comes after encoding of "y"); and

b) by the order in which extension values appear in the value to be encoded (e.g., the presence bit and value for "groupOfExtensions {y 2, z 2}" in the first "ext-list1" element appears before the presence bit for the absent "groupOfExtensions" in the second "ext-list1" element).

The p-bits indicate the presence of optional components as in PER.

3. How the wanted encoding can be specified?

3.1. ECN definitions

The first version of ECN specification will not contain support for extensibility. Support for extensibility will be added in the first amendment. The extensibility support can be emulated with user-defined encoding objects.

- All sequences are marked as implicitly extensible. This is achieved by marking the encoding structures as #SEQUENCE-WITH-IMPLIED-EXTENSIBILITY.
- Each group of extensions for a given version is encapsulated within extension brackets ("[[", "]]").
- Each extension group is identified with version information. This version information is specified in comments. (But the ITU-T ASN.1 group is currently working on a more formal way to handle versioning: see section 3.3 below.)
- Known extensions (both present and absent) are not encoded in the location where PER would put them. Instead they are moved into an extension list.
- Known extensions in the extension list are sorted according to their extension version.
- The extension list is encoded after all the non-extended fields.

The following encoding definition module contains an encoding object that provides the needed properties for **all** sequence types.

```
RRC-Encodings ENCODING-DEFINITIONS ::=

BEGIN

-- ************************************************************
-- Mark those IE types that need support for non-critical extensions
-- ************************************************************
RENAMES
    #SEQUENCE AS #SEQUENCE-WITH-IMPLIED-EXTENSIBILITY
    IN ALL – if necessary, some encoding structures can be EXCEPTed
FROM RRC-IE-definitions
    #SEQUENCE AS #SEQUENCE-WITH-IMPLIED-EXTENSIBILITY
    IN ALL -- if necessary, some encoding structures can be EXCEPTed
FROM RRC-PDU-definitions;

-- ************************************************************
-- This encoding object set contains all the encodings for RRC
-- messages.
-- ************************************************************
RRC-Encodings #ENCODINGS ::= {
    sequence-with-implied-extensibility-encoding
}

#SEQUENCE-WITH-IMPLIED-EXTENSIBILITY ::= #SEQUENCE

-- ************************************************************
-- This encoding object changes the encoding for all sequences.
-- For non-extended sequences encoding is as in PER.
-- For extended sequences encoding of the extension root is as
-- in PER without the preceding extension bit. Extension additions
-- are encoded in the end of container.
-- ************************************************************
```

```
sequence-with-implied-extensibility-encoding
    #SEQUENCE-WITH-IMPLIED-EXTENSIBILITY ::=
USER-DEFINED-BEGIN
-- ENCODE WITH PER-BASIC-UNALIGNED
-- EXCEPT
-- * Do not encode extension bit as specified in X.691 clause 18.1.
-- * Do not encode "n" bits for extension addition as specified in X.691
--   clause 18.7.
-- * Move extension groups in the end of container. Order extensions as follows:
--   a) Extensions with lower version number shall appear before extensions with
--      a higher version number.
--   b) The relative order of extensions with the same version number shall be
--      preserved (i.e. the order is the same in which they would appear in PER
--      encoding).
-- * When decoding an extension group is present if end-of-container has
--   not been reached.
USER-DEFINED-END

END
```

Basically encoding for extensions is the same as in PER except that there is no extension bit nor the count for extension additions and extension groups are moved in the end of message. An extension group is encoded as specified in X.691, subclause 18.9, i.e., as a sequence.

3.2. Implications

- Placeholders for non-critical disappear in the ASN.1 definition for RRC messages.
- Non-critical extensions can be introduced in the middle of a message using the normal ASN.1 extensibility mechanism ⇒ ASN.1 definitions can be kept simpler.
- All sequence types can be extended to have non-critical extensions, and such extensions will be backwards compatible.
- When extensions are introduced then **management** of extension versions becomes important. It must be specified which extension groups belong to a given extension version.
- ECN definition for non-critical extensions is fairly simple. The presented version uses user-defined encoding objects to specify the extensibility part.
- The cost for non-critical extensions is zero bits in all cases.

3.3. ECN support for extensibility

The first ECN amendment [1] will have means to specify the same encoding as specified above without need to resort to user-defined encoding objects. Support for extensibility will mature in late 2001.

4. References

[1] TSG-R2#18(01)0152 Inclusion of Release-4 Information Elements in ASN.1

[2] Examples for inclusion of release-4 IEs in ASN.1

[3] 3GPP TS 25.331 v3.5.0 RRC Protocol Specification

[4] Non-critical extensibility in 3GPP, email by prof. John Larmouth on ecn@oss.com mailing list

[5] X.692 Amd 1: Extensibility