**3GPP TSG-RAN WG2 Meeting #112-e R2-20xxxxx**

**Online, xx-yy November 2020**

**Agenda Item: x.y**

**Source: MediaTek Inc.**

**Title: TP from email discussion [Post111-e][901] Extension scenarios for ToAddMod lists (MediaTek)**

**Document for: Discussion, decision**

5 Text proposal

## A.4.2 Critical extension of messages and fields

The mechanisms to critically extend a message are defined in A.3.3. There are both "outer branch" and "inner branch" mechanisms available. The "outer branch" consists of a CHOICE having the name *criticalExtensions*, with two values, *c1* and *criticalExtensionsFuture*. The *criticalExtensionsFuture* branch consists of an empty SEQUENCE, while the c1 branch contains the "inner branch" mechanism.

The "inner branch" structure is a CHOICE with values of the form "*MessageName-rX-IEs*" (e.g., "*RRCConnectionReconfiguration-r8-IEs*") or "*spareX*", with the spare values having type NULL. The "-rX-IEs" structures contain the *complete* structure of the message IEs for the appropriate release; i.e., the critical extension branch for the Rel-10 version of a message includes all Rel-8 and Rel-9 fields (that are not obviated in the later version), rather than containing only the additional Rel-10 fields.

The following guidelines may be used when deciding which mechanism to introduce for a particular message, i.e. only an 'outer branch', or an 'outer branch' in combination with an 'inner branch' including a certain number of spares:

- For certain messages, e.g. initial uplink messages, messages transmitted on a broadcast channel, critical extension may not be applicable.

- An outer branch may be sufficient for messages not including any fields.

- The number of spares within inner branch should reflect the likelihood that the message will be critically extended in future releases (since each release with a critical extension for the message consumes one of the spare values). The estimation of the critical extension likelihood may be based on the number, size and changeability of the fields included in the message.

- In messages where an inner branch extension mechanism is available, all spare values of the inner branch should be used before any critical extensions are added using the outer branch.

The following example illustrates the use of the critical extension mechanism by showing the ASN.1 of the original and of a later release

-- /example/ ASN1START -- Original release

RRCMessage ::= SEQUENCE {

 rrc-TransactionIdentifier RRC-TransactionIdentifier,

 criticalExtensions CHOICE {

 c1 CHOICE{

 rrcMessage-r8 RRCMessage-r8-IEs,

 spare3 NULL, spare2 NULL, spare1 NULL

 },

 criticalExtensionsFuture SEQUENCE {}

 }

}

-- ASN1STOP

-- /example/ ASN1START -- Later release

RRCMessage ::= SEQUENCE {

 rrc-TransactionIdentifier RRC-TransactionIdentifier,

 criticalExtensions CHOICE {

 c1 CHOICE{

 rrcMessage-r8 RRCMessage-r8-IEs,

 rrcMessage-r10 RRCMessage-r10-IEs,

 rrcMessage-r11 RRCMessage-r11-IEs,

 rrcMessage-r14 RRCMessage-r14-IEs

 },

 later CHOICE {

 c2 CHOICE{

 rrcMessage-r16 RRCMessage-r16-IEs,

 spare7 NULL, spare6 NULL, spare5 NULL, spare4 NULL,

 spare3 NULL, spare2 NULL, spare1 NULL

 },

 criticalExtensionsFuture SEQUENCE {}

 }

 }

}

-- ASN1STOP

It is important to note that critical extensions may also be used at the level of individual fields i.e. a field may be replaced by a critically extended version. When sending the extended version, the original version may also be included (e.g. original field is mandatory, E-UTRAN is unaware if UE supports the extended version). In such cases, a UE supporting both versions may be required to ignore the original field. The following example illustrates the use of the critical extension mechanism by showing the ASN.1 of the original and of a later release.

-- /example/ ASN1START -- Original release

RRCMessage ::= SEQUENCE {

 rrc-TransactionIdentifier RRC-TransactionIdentifier,

 criticalExtensions CHOICE {

 c1 CHOICE{

 rrcMessage-r8 RRCMessage-r8-IEs,

 spare3 NULL, spare2 NULL, spare1 NULL

 },

 criticalExtensionsFuture SEQUENCE {}

 }

}

RRCMessage-rN-IEs ::= SEQUENCE {

 field1-rN ENUMERATED {

 value1, value2, value3, value4} OPTIONAL, -- Need N

 field2-rN InformationElement2-rN OPTIONAL, -- Need N

 nonCriticalExtension RRCConnectionReconfiguration-vMxy-IEs OPTIONAL

}

RRCConnectionReconfiguration-vMxy-IEs ::= SEQUENCE {

 field2-rM InformationElement2-rM OPTIONAL, -- Cond NoField2rN

 nonCriticalExtension SEQUENCE {} OPTIONAL

}

-- ASN1STOP

| Conditional presence | Explanation |
| --- | --- |
| *NoField2rN* | The field is optionally present, need N, if field2-rN is absent. Otherwise the field is absent |

Finally, it is noted that a critical extension may be introduced in the same release as the one in which the original field was introduced e.g. to correct an essential ASN.1 error. In such cases a UE capability may be introduced, to assist the network in deciding whether or not to use the critical extension.

In the case of list fields (SEQUENCE OF types in ASN.1) using the ToAddMod/ToRelease construction, the use of critical extensions to increase the size of a list should be avoided; that is, extensions done according to the following example should be avoided:

-- /example/ ASN1START -- Discouraged example

ContainingStructure ::= SEQUENCE {

 originalToAddModList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementType OPTIONAL, -- Need N

 ...

 [[

 originalToAddModList-rN SEQUENCE (SIZE (1..newMaxSize)) OF ListElementType OPTIONAL -- Need N

 ]]

}

-- ASN1STOP

Instead, a non-critical list extension mechanism should typically be used, such that the extension field only adds the new entries of the list . This approach is further discussed in section A.4.3.x.

If the critical extension mechanism for a list is used, it should be clarified in the field description that the two versions of the list are not configured together, and that the network should release the contents of the original version when configuring the replacement version.

## A.4.3 Non-critical extension of messages

### […]

### A.4.3.x Non-critical extensions of lists with ToAddMod/ToRelease

When the length of a list using the ToAddMod/ToRelease construction is extended and/or fields are added to the list element structure, the list should be non-critically extended, i.e. by adding only the new entries to the list, coupled with the use of a parallel list structure to contain any fields added to the list elements. The following general principles apply:

– When the length of the list is extended, this is reflected in a non-critical extension of the list, with a “SizeExt” suffix added to the end of the field name (before any -rN suffix). A new ToRelease list is generally needed and its range should generally include the total number of entries to allow the new ToRelease list also release the original entries. The list element ID type will typically also need to be extended to account for the increased size of the list; note that in case the list element ID is included in the list element type, an extension of the type will be needed, resulting in a more complex extension similar to example 3 below. The field description table should indicate that the UE considers the original list and the extension list as a single list; thus entries added with the original list can be modified by the extension list (or removed by the extension of the ToRelease list), or vice versa. The result is as shown in the following example:

-- /example 1/ ASN1START

ContainingStructure ::= SEQUENCE {

 originalToAddModList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementType OPTIONAL, -- Need N

 originalToReleaseList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Non-critical extension lists

 originalToAddModListSizeExt-rN SEQUENCE (SIZE (1..numAdditionalElements-rN)) OF ListElementType OPTIONAL, -- Need N

 originalToReleaseListSizeExt-rN SEQUENCE (SIZE (1..newMaxSize-rN)) OF ListElementId-rN OPTIONAL -- Need N

 ]]

}

ListElementId ::= INTEGER (0..originalMaxSize-1)

ListElementId-rN ::= INTEGER (0..newMaxSize-rN-1)

-- ASN1STOP

– When fields are added to the list element structure, an extension marker should normally be used if available. If no extension marker is available or if overhead or other considerations prevent using the extension marker, an extension structure should be created for the new fields, with the suffix “Ext” added to the end of the field name (before any -rN suffix), and a parallel list introduced to hold the new structures, also with the “Ext” suffix. The field description table should indicate that the parallel list contains the same number of entries, and in the same order, as the original list. No new ToRelease list is typically needed (unless the list element ID type changes). It should typically be ensured that the contained fields in the “Ext” elements are also releasable to avoid situations where the only way to release them is via release and add of the entire list element; this can, for instance, be ensured by having the new fields be OPTIONAL Need R. The result is as shown in the following example:

-- /example 2/ ASN1START

ContainingStructure ::= SEQUENCE {

 originalToAddModList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementType OPTIONAL, -- Need N

 originalToReleaseList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Parallel list

 originalToAddModListExt-rN SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementTypeExt-rN OPTIONAL -- Need N

 ]]

}

ListElementType ::= SEQUENCE {

 elementId ListElementId,

 field1 INTEGER (0..3),

 field2 ENUMERATED { value1, value2, value3 }

}

ListElementTypeExt-rN ::= SEQUENCE {

 field3 BIT STRING (SIZE(8)) OPTIONAL -- Need R

}

-- ASN1STOP

– When the length of a list is extended *and* fields are added to the list element structure, an extension marker should normally be used for the added fields if available, and the list extended with the non-critical mechanism as described above. If no extension marker is available or if overhead or other considerations prevent using the extension marker, the list should be non-critically extended to hold the new entries, and a second list parallel to the concatenation of the original and extended lists should be introduced to hold the new entries (similar to the second example above). Finally, an extension structure should be created for the new fields (as in the second example above). The field description table should indicate that the parallel list contains the same number of entries, and in the same order, as the concatenation of the original list and the extension list. An extended ToRelease list is generally needed; in addition, if the element ID type changes (e.g. due to the extension of the list size), a second, parallel ToRelease list would be needed. The result is as shown in the following example:

-- /example 3/ ASN1START

ContainingStructure ::= SEQUENCE {

 originalToAddModList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementType OPTIONAL, -- Need N

 originalToReleaseList SEQUENCE (SIZE (1..originalMaxSize)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Non-critical extension lists

 originalToAddModListSizeExt-rN SEQUENCE (SIZE (1..numAdditionalElements-rN)) OF ListElementType OPTIONAL, -- Need N

 originalToReleaseListSizeExt-rN SEQUENCE (SIZE (1..newMaxSize-rN)) OF ListElementId-rN OPTIONAL, -- Need N

 -- Parallel lists with newMaxSize = originalMaxSize + numAdditionalElements

 originalToAddModListExt-rN SEQUENCE (SIZE (1..newMaxSize-rN)) OF ListElementTypeExt-rN OPTIONAL, -- Need N

 originalToReleaseListExt-rN SEQUENCE (SIZE (1..newMaxSize-rN)) OF ListElementId-rN OPTIONAL -- Need N

 ]]

}

ListElementType ::= SEQUENCE {

 elementId ListElementId,

 field1 INTEGER (0..3),

 field2 ENUMERATED { value1, value2, value3 }

}

ListElementTypeExt-rN ::= SEQUENCE {

 elementId-rN ListElementId-rN,

 field3-rN BIT STRING (SIZE(8))

}

ListElementId ::= INTEGER (0..originalMaxSize-1)

ListElementId-rN ::= INTEGER (0..newMaxSize-rN-1)

-- ASN1STOP