### 6.1.3 General rules

In the ASN.1 of this specification, the first bit of a bit string refers to the leftmost bit, unless stated otherwise.

Upon reception of a list not using ToAddModList and ToReleaseList structure, the UE shall delete all entries of the list currently in the UE configuration before applying the received list and shall consider each entry as newly created. This applies also to lists whose size is extended (i.e. with a second list structure in the ASN.1 comprising additional entries). This implies that Need M should not be used for fields in the entries of these lists; if used, UE will handle such fields equivalent to a Need R.

### […]

### 6.3.2 Radio resource control information elements

#### […]

#### *– BeamFailureRecoveryConfig*

The IE *BeamFailureRecoveryConfig* is used to configure the UE with RACH resources and candidate beams for beam failure recovery in case of beam failure detection. See also TS 38.321 [3], clause 5.1.1.

*BeamFailureRecoveryConfig* information element

-- ASN1START

-- TAG-BEAMFAILURERECOVERYCONFIG-START

BeamFailureRecoveryConfig ::= SEQUENCE {

 rootSequenceIndex-BFR INTEGER (0..137) OPTIONAL, -- Need M

 rach-ConfigBFR RACH-ConfigGeneric OPTIONAL, -- Need M

 rsrp-ThresholdSSB RSRP-Range OPTIONAL, -- Need M

 candidateBeamRSList SEQUENCE (SIZE(1..maxNrofCandidateBeams)) OF PRACH-ResourceDedicatedBFR OPTIONAL, -- Need M

 ssb-perRACH-Occasion ENUMERATED {oneEighth, oneFourth, oneHalf, one, two,

 four, eight, sixteen} OPTIONAL, -- Need M

 ra-ssb-OccasionMaskIndex INTEGER (0..15) OPTIONAL, -- Need M

 recoverySearchSpaceId SearchSpaceId OPTIONAL, -- Need R

 ra-Prioritization RA-Prioritization OPTIONAL, -- Need R

 beamFailureRecoveryTimer ENUMERATED {ms10, ms20, ms40, ms60, ms80, ms100, ms150, ms200} OPTIONAL, -- Need M

 ...,

 [[

 msg1-SubcarrierSpacing SubcarrierSpacing OPTIONAL -- Need M

 ]],

 [[

 ra-PrioritizationTwoStep-r16 RA-Prioritization OPTIONAL, -- Need R

 candidateBeamRSListExt-v1610 SetupRelease{ CandidateBeamRSListExt-r16 } OPTIONAL -- Need M

 ]],

 [[

 spCell-BFR-CBRA-r16 ENUMERATED {true} OPTIONAL -- Need R

 ]]

}

PRACH-ResourceDedicatedBFR ::= CHOICE {

 ssb BFR-SSB-Resource,

 csi-RS BFR-CSIRS-Resource

}

BFR-SSB-Resource ::= SEQUENCE {

 ssb SSB-Index,

 ra-PreambleIndex INTEGER (0..63),

 ...

}

BFR-CSIRS-Resource ::= SEQUENCE {

 csi-RS NZP-CSI-RS-ResourceId,

 ra-OccasionList SEQUENCE (SIZE(1..maxRA-OccasionsPerCSIRS)) OF INTEGER (0..maxRA-Occasions-1) OPTIONAL, -- Need R

 ra-PreambleIndex INTEGER (0..63) OPTIONAL, -- Need R

 ...

}

CandidateBeamRSListExt-r16::= SEQUENCE (SIZE(1.. maxNrofCandidateBeamsExt-r16)) OF PRACH-ResourceDedicatedBFR

-- TAG-BEAMFAILURERECOVERYCONFIG-STOP

-- ASN1STOP

|  |
| --- |
| *BeamFailureRecoveryConfig* field descriptions |
| ***beamFailureRecoveryTimer***Timer for beam failure recovery timer. Upon expiration of the timer the UE does not use CFRA for BFR. Value in ms. Value *ms10* corresponds to 10 ms, value *ms20* corresponds to 20 ms, and so on. |
| ***candidateBeamRSList, candidateBeamRSListExt-v1610***The list of reference signals (CSI-RS and/or SSB) identifying the candidate beams for recovery and the associated RA parameters. The UE shall consider this list to include all elements of *candidateBeamRSList* (without suffix) and all elements of *candidateBeamRSListExt-v1610*, and the entire list is replaced when a new list is signalled in *candidateBeamRSList* (without suffix) and/or *candidateBeamRSListExt-v1610*. The *release* branch of *candidateBeamRSListExt-v1610* is not used. The network configures these reference signals to be within the linked DL BWP (i.e., within the DL BWP with the same *bwp-Id*) of the UL BWP in which the *BeamFailureRecoveryConfig* is provided.  |
| ***msg1-SubcarrierSpacing***Subcarrier spacing for contention free beam failure recovery. Only the values 15 kHz or 30 kHz (FR1), and 60 kHz or 120 kHz (FR2) are applicable. See TS 38.211 [16], clause 5.3.2. |
| ***rsrp-ThresholdSSB***L1-RSRP threshold used for determining whether a candidate beam may be used by the UE to attempt contention free random access to recover from beam failure (see TS 38.213 [13], clause 6). |
| ***ra-prioritization***Parameters which apply for prioritized random access procedure for BFR (see TS 38.321 [3], clause 5.1.1). |
| ***ra-PrioritizationTwoStep***Parameters which apply for prioritized 2-step random access procedure for BFR (see TS 38.321 [3], clause 5.1.1). |
| ***ra-ssb-OccasionMaskIndex***Explicitly signalled PRACH Mask Index for RA Resource selection in TS 38.321 [3]. The mask is valid for all SSB resources. |
| ***rach-ConfigBFR***Configuration of contention free random access occasions for BFR. |
| ***recoverySearchSpaceId***Search space to use for BFR RAR. The network configures this search space to be within the linked DL BWP (i.e., within the DL BWP with the same *bwp-Id*) of the UL BWP in which the *BeamFailureRecoveryConfig* is provided. The CORESET associated with the recovery search space cannot be associated with another search space. Network always configures the UE with a value for this field when contention free random access resources for BFR are configured. |
| ***rootSequenceIndex-BFR***PRACH root sequence index (see TS 38.211 [16], clause 6.3.3.1) for beam failure recovery. |
| ***spCell-BFR-CBRA***Indicates that UE is configured to send BFR MAC CE for SpCell BFR as specified in TS38.321 [3]. |
| ***ssb-perRACH-Occasion***Number of SSBs per RACH occasion for CF-BFR, see TS 38.213 [13], clause 8.1. |

|  |
| --- |
| *BFR-CSIRS-Resource* field descriptions |
| ***csi-RS***The ID of a *NZP-CSI-RS-Resource* configured in the *CSI-MeasConfig* of this serving cell. This reference signal determines a candidate beam for beam failure recovery (BFR). |
| ***ra-OccasionList***RA occasions that the UE shall use when performing BFR upon selecting the candidate beam identified by this CSI-RS. The network ensures that the RA occasion indexes provided herein are also configured by *prach-ConfigurationIndex* and *msg1-FDM*. Each RACH occasion is sequentially numbered, first, in increasing order of frequency resource indexes for frequency multiplexed PRACH occasions; second, in increasing order of time resource indexes for time multiplexed PRACH occasions within a PRACH slot and Third, in increasing order of indexes for PRACH slots.If the field is absent the UE uses the RA occasion associated with the SSB that is QCLed with this CSI-RS. |
| ***ra-PreambleIndex***The RA preamble index to use in the RA occasions associated with this CSI-RS. If the field is absent, the UE uses the preamble index associated with the SSB that is QCLed with this CSI-RS. |

|  |
| --- |
| *BFR-SSB-Resource* field descriptions |
| ***ra-PreambleIndex***The preamble index that the UE shall use when performing BFR upon selecting the candidate beams identified by this SSB. |
| ***ssb***The ID of an SSB transmitted by this serving cell. It determines a candidate beam for beam failure recovery (BFR). |

#### […]

## […]

## A.3.10 Guidelines on use of of lists (without ToAddModList and ToReleaseList)

As per subclause 6.1.3, when using lists without the ToAddModList and ToReleaseList structure, the contents of the lists are always replaced. To illustrate this, an example is provided below:

-- /example/ ASN1START

-- TAG\_EXAMPLE\_LISTS\_START

AnExampleIE ::= SEQUENCE {

 elementList SEQUENCE (SIZE (1..maxNrofElements)) OF Element OPTIONAL, -- Need M

 ...,

 [[

 elementListExt-v2030 SEQUENCE (SIZE (1..maxNrofElementsExt)) OF Element OPTIONAL, -- Need M

 ]]

}

Element ::= SEQUENCE {

 useFeatureX BOOLEAN,

 aField INTEGER (0..127) OPTIONAL, -- Need M

 anotherField INTEGER (0..127) OPTIONAL, -- Need R

 ...

}

maxNrofElements INTEGER ::= 8

maxNrofElements-1 INTEGER ::= 7

maxNrofElementsExt INTEGER ::= 8

maxNrofElementsExt-1 INTEGER ::= 7

-- TAG\_EXAMPLE\_LISTS\_STOP

-- /example/ ASN1STOP

As can be seen, the *elementList* list itself uses Need M, but each list entry *Element* contains mandatory, Need M and Need R fields. If the list is first signalled to UE with 3 entries, and subsequently again with 2 entries, UE shall retain only the latter list, i.e. the list with 2 elements will completely replace the list with 3 elements. That also means that the field *aField* will be treated as if it was newly created, i.e. network must include it if it wishes UE to utilize the field even if it was previously signalled. This also implies that the Need M field (*aField*) will be treated in the same way as the Need R field (*anotherField*), i.e. delta signalling is not applied and the network has to signal the field to ensure UE does not release the value (which is why Need M should not normally be used in the entries of these lists).

# A.4 Extension of the PDU specifications

## A.4.1 General principles to ensure compatibility

It is essential that extension of the protocol does not affect interoperability i.e. it is essential that implementations based on different versions of the RRC protocol are able to interoperate. In particular, this requirement applies for the following kind of protocol extensions:

- Introduction of new PDU types (i.e. these should not cause unexpected behaviour or damage).

- Introduction of additional fields in an extensible PDUs (i.e. it should be possible to ignore uncomprehended extensions without affecting the handling of the other parts of the message).

- Introduction of additional values of an extensible field of PDUs. If used, the behaviour upon reception of an uncomprehended value should be defined.

It should be noted that the PDU extension mechanism may depend on the logical channel used to transfer the message e.g. for some PDUs an implementation may be aware of the protocol version of the peer in which case selective ignoring of extensions may not be required.

The non-critical extension mechanism is the primary mechanism for introducing protocol extensions i.e. the critical extension mechanism is used merely when there is a need to introduce a 'clean' message version. Such a need appears when the last message version includes a large number of non-critical extensions, which results in issues like readability, overhead associated with the extension markers. The critical extension mechanism may also be considered when it is complicated to accommodate the extensions by means of non-critical extension mechanisms.

## 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, replacing the original list field by a new field also used to signal entries previously covered by the original field (i.e. extensions done according to the following example) should be avoided:

-- /example/ ASN1START -- Discouraged example

ContainingStructure ::= SEQUENCE {

 listElementToAddModList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElement OPTIONAL, -- Need N

 ...,

 [[

 listElementToAddModList-rN SEQUENCE (SIZE (1..maxNrofListElements-rN)) OF ListElement 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 described in subclause A.4.3.6.

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.4.3.1 General principles

The mechanisms to extend a message in a non-critical manner are defined in A.3.3. W.r.t. the use of extension markers, the following additional guidelines apply:

- When further non-critical extensions are added to a message that has been critically extended, the inclusion of these non-critical extensions in earlier critical branches of the message should be avoided when possible.

- The extension marker ("...") is the primary non-critical extension mechanism that is used but empty sequences may be used if length determinant is not required. Examples of cases where a length determinant is not required:

- at the end of a message;

- at the end of a structure contained in a BIT STRING or OCTET STRING.

- When an extension marker is available, non-critical extensions are preferably placed at the location (e.g. the IE) where the concerned parameter belongs from a logical/ functional perspective (referred to as the '*default extension location*').

- It is desirable to aggregate extensions of the same release or version of the specification into a group, which should be placed at the lowest possible level.

- In specific cases it may be preferable to place extensions elsewhere (referred to as the '*actual extension location*') e.g. when it is possible to aggregate several extensions in a group. In such a case, the group should be placed at the lowest suitable level in the message. <TBD: ref to separate example>

- In case placement at the default extension location affects earlier critical branches of the message, locating the extension at a following higher level in the message should be considered.

- In case an extension is not placed at the default extension location, an IE should be defined. The IE's ASN.1 definition should be placed in the same ASN.1 section as the default extension location. In case there are intermediate levels in-between the actual and the default extension location, an IE may be defined for each level. Intermediate levels are primarily introduced for readability and overview. Hence intermediate levels need not always be introduced e.g. they may not be needed when the default and the actual extension location are within the same ASN.1 section. <TBD: ref to separate example>

### A.4.3.2 Further guidelines

Further to the general principles defined in the previous section, the following additional guidelines apply regarding the use of extension markers:

- Extension markers within SEQUENCE:

- Extension markers are primarily, but not exclusively, introduced at the higher nesting levels.

- Extension markers are introduced for a SEQUENCE comprising several fields as well as for information elements whose extension would result in complex structures without it (e.g. re-introducing another list).

- Extension markers are introduced to make it possible to maintain important information structures e.g. parameters relevant for one particular RAT.

- Extension markers are also used for size critical messages (i.e. messages on BCCH, BR-BCCH, PCCH and CCCH), although introduced somewhat more carefully.

- The extension fields introduced (or frozen) in a specific version of the specification are grouped together using double brackets.

- Extension markers within ENUMERATED:

- Spare values may be used until the number of values reaches the next power of 2, while the extension marker caters for extension beyond that limit, given that the use of spare values in a later Release is possible without any error cases.

- A suffix of the form "vXYZ" is used for the identifier of each new value, e.g. "value-vXYZ".

- Extension markers within CHOICE:

- Extension markers are introduced when extension is foreseen and when comprehension is not required by the receiver i.e. behaviour is defined for the case where the receiver cannot comprehend the extended value (e.g. ignoring an optional CHOICE field). It should be noted that defining the behaviour of a receiver upon receiving a not comprehended choice value is not required if the sender is aware whether or not the receiver supports the extended value.

- A suffix of the form "vXYZ" is used for the identifier of each new choice value, e.g. "choice-vXYZ".

Non-critical extensions at the end of a message/ of a field contained in an OCTET or BIT STRING:

- When a nonCriticalExtension is actually used, a "Need" code should not be provided for the field, which always is a group including at least one extension and a field facilitating further possible extensions. For simplicity, it is recommended not to provide a "Need" code when the field is not actually used either.

Further, more general, guidelines:

- In case a need code is not provided for a group, a "Need" code is provided for all individual extension fields within the group i.e. including for fields that are not marked as OPTIONAL. The latter is to clarify the action upon absence of the whole group.

### A.4.3.3 Typical example of evolution of IE with local extensions

The following example illustrates the use of the extension marker for a number of elementary cases (sequence, enumerated, choice). The example also illustrates how the IE may be revised in case the critical extension mechanism is used.

NOTE In case there is a need to support further extensions of release n while the ASN.1 of release (n+1) has been frozen, without requiring the release n receiver to support decoding of release (n+1) extensions, more advanced mechanisms are needed e.g. including multiple extension markers.

-- /example/ ASN1START

InformationElement1 ::= SEQUENCE {

 field1 ENUMERATED {

 value1, value2, value3, value4-v880,

 ..., value5-v960 },

 field2 CHOICE {

 field2a BOOLEAN,

 field2b InformationElement2b,

 ...,

 field2c-v960 InformationElement2c-r9

 },

 ...,

 [[

 field3-r9 InformationElement3-r9 OPTIONAL -- Need R

 ]],

 [[

 field3-v9a0 InformationElement3-v9a0 OPTIONAL, -- Need R

 field4-r9 InformationElement4 OPTIONAL -- Need R

 ]]

}

InformationElement1-r10 ::= SEQUENCE {

 field1 ENUMERATED {

 value1, value2, value3, value4-v880,

 value5-v960, value6-v1170, spare2, spare1, ... },

 field2 CHOICE {

 field2a BOOLEAN,

 field2b InformationElement2b,

 field2c-v960 InformationElement2c-r9,

 ...,

 field2d-v12b0 INTEGER (0..63)

 },

 field3-r9 InformationElement3-r10 OPTIONAL, -- Need R

 field4-r9 InformationElement4 OPTIONAL, -- Need R

 field5-r10 BOOLEAN,

 field6-r10 InformationElement6-r10 OPTIONAL, -- Need R

 ...,

 [[

 field3-v1170 InformationElement3-v1170 OPTIONAL -- Need R

 ]]

}

-- ASN1STOP

Some remarks regarding the extensions of *InformationElement1* as shown in the above example:

– The *InformationElement1* is initially extended with a number of non-critical extensions. In release 10 however, a critical extension is introduced for the message using this IE. Consequently, a new version of the IE *InformationElement1* (i.e. *InformationElement1-r10*) is defined in which the earlier non-critical extensions are incorporated by means of a revision of the original field.

– The *value4-v880* is replacing a spare value defined in the original protocol version for *field1*. Likewise *value6-v1170* replaces *spare3* that was originally defined in the r10 version of *field1.*

– Within the critically extended release 10 version of *InformationElement1*, the names of the original fields/IEs are not changed, unless there is a real need to distinguish them from other fields/IEs. E.g. the *field1* and *InformationElement4* were defined in the original protocol version (release 8) and hence not tagged. Moreover, the *field3-r9* is introduced in release 9 and not re-tagged; although, the *InformationElement3* is also critically extended and therefore tagged *InformationElement3-r10* in the release 10 version of InformationElement1.

### A.4.3.4 Typical examples of non critical extension at the end of a message

The following example illustrates the use of non-critical extensions at the end of the message or at the end of a field that is contained in a BIT or OCTET STRING i.e. when an empty sequence is used.

-- /example/ ASN1START

RRCMessage-r8-IEs ::= SEQUENCE {

 field1 InformationElement1,

 field2 InformationElement2,

 field3 InformationElement3 OPTIONAL, -- Need N

 nonCriticalExtension RRCMessage-v860-IEs OPTIONAL

}

RRCMessage-v860-IEs ::= SEQUENCE {

 field4-v860 InformationElement4 OPTIONAL, -- Need S

 field5-v860 BOOLEAN OPTIONAL, -- Cond C54

 nonCriticalExtension RRCMessage-v940-IEs OPTIONAL

}

RRCMessage-v940-IEs ::= SEQUENCE {

 field6-v940 InformationElement6-r9 OPTIONAL, -- Need R

 nonCriticalExtensions SEQUENCE {} OPTIONAL

}

-- ASN1STOP

Some remarks regarding the extensions shown in the above example:

– The *InformationElement4* is introduced in the original version of the protocol (release 8) and hence no suffix is used.

### A.4.3.5 Examples of non-critical extensions not placed at the default extension location

The following example illustrates the use of non-critical extensions in case an extension is not placed at the default extension location.

#### – *ParentIE-WithEM*

The IE *ParentIE-WithEM*is an example of a high level IE including the extension marker (EM). The root encoding of this IE includes two lower level IEs *ChildIE1-WithoutEM* and *ChildIE2-WithoutEM* which not include the extension marker. Consequently, non-critical extensions of the Child-IEs have to be included at the level of the Parent-IE.

The example illustrates how the two extension IEs *ChildIE1-WithoutEM-vNx0* and *ChildIE2-WithoutEM-vNx0* (both in release N) are used to connect non-critical extensions with a default extension location in the lower level IEs to the actual extension location in this IE.

*ParentIE-WithEM* information element

-- /example/ ASN1START

ParentIE-WithEM ::= SEQUENCE {

 -- Root encoding, including:

 childIE1-WithoutEM ChildIE1-WithoutEM OPTIONAL, -- Need N

 childIE2-WithoutEM ChildIE2-WithoutEM OPTIONAL, -- Need N

 ...,

 [[

 childIE1-WithoutEM-vNx0 ChildIE1-WithoutEM-vNx0 OPTIONAL, -- Need N

 childIE2-WithoutEM-vNx0 ChildIE2-WithoutEM-vNx0 OPTIONAL -- Need N

 ]]

}

-- ASN1STOP

Some remarks regarding the extensions shown in the above example:

– The fields *childIEx-WithoutEM-vNx0* may not really need to be optional (depends on what is defined at the next lower level).

– In general, especially when there are several nesting levels, fields should be marked as optional only when there is a clear reason.

#### *– ChildIE1-WithoutEM*

The IE *ChildIE1-WithoutEM* is an example of a lower level IE, used to control certain radio configurations including a configurable feature which can be setup or released using the local IE *ChIE1-ConfigurableFeature*. The example illustrates how the new field *chIE1-NewField* is added in release N to the configuration of the configurable feature. The example is based on the following assumptions:

– When initially configuring as well as when modifying the new field, the original fields of the configurable feature have to be provided also i.e. as if the extended ones were present within the setup branch of this feature.

– When the configurable feature is released, the new field should be released also.

– When omitting the original fields of the configurable feature the UE continues using the existing values (which is used to optimise the signalling for features that typically continue unchanged upon handover).

– When omitting the new field of the configurable feature the UE releases the existing values and discontinues the associated functionality (which may be used to support release of unsupported functionality upon handover to an eNB supporting an earlier protocol version).

The above assumptions, which affect the use of conditions and need codes, may not always apply. Hence, the example should not be re-used blindly.

*ChildIE1-WithoutEM* information element

-- /example/ ASN1START

ChildIE1-WithoutEM ::= SEQUENCE {

 -- Root encoding, including:

 chIE1-ConfigurableFeature ChIE1-ConfigurableFeature OPTIONAL -- Need N

}

ChildIE1-WithoutEM-vNx0 ::= SEQUENCE {

 chIE1-ConfigurableFeature-vNx0 ChIE1-ConfigurableFeature-vNx0 OPTIONAL -- Cond ConfigF

}

ChIE1-ConfigurableFeature ::= CHOICE {

 release NULL,

 setup SEQUENCE {

 -- Root encoding

 }

}

ChIE1-ConfigurableFeature-vNx0 ::= SEQUENCE {

 chIE1-NewField-rN INTEGER (0..31)

}

-- ASN1STOP

| Conditional presence | Explanation |
| --- | --- |
| *ConfigF* | The field is optional present, need R, in case of chIE1-ConfigurableFeature is included and set to "setup"; otherwise the field is absent and the UE shall delete any existing value for this field. |

#### *– ChildIE2-WithoutEM*

The IE *ChildIE2-WithoutEM* is an example of a lower level IE, typically used to control certain radio configurations. The example illustrates how the new field *chIE1-NewField* is added in release N to the configuration of the configurable feature.

*ChildIE2-WithoutEM* information element

-- /example/ ASN1START

ChildIE2-WithoutEM ::= CHOICE {

 release NULL,

 setup SEQUENCE {

 -- Root encoding

 }

}

ChildIE2-WithoutEM-vNx0 ::= SEQUENCE {

 chIE2-NewField-rN INTEGER (0..31) OPTIONAL -- Cond ConfigF

}

-- ASN1STOP

| Conditional presence | Explanation |
| --- | --- |
| *ConfigF* | The field is optional present, need R, in case of chIE2-ConfigurableFeature is included and set to "setup"; otherwise the field is absent and the UE shall delete any existing value for this field. |

A.4.3.6 Non-critical extensions of lists with ToAddMod/ToRelease

When the size 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 in accordance with the following general principles:

– When only the size of the list is extended, this extension is reflected in a non-critical extension of the list, with a "SizeExt" suffix added to the end of the field name (before the -vNxy suffix). The differential size of the extended list uses the suffix "Diff". A new ToRelease list is needed, and its range should include only the increase in list size. In many cases, extending the list size will also require an extended list element ID type to account for the increased size of the list; in these cases the element type will need to be extended to include the extended element ID, resulting in a more complex extension (see example 3 for further discussion of this case). 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 {

 listElementToAddModList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElement OPTIONAL, -- Need N

 listElementToReleaseList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Non-critical extension lists

 listElementToAddModListSizeExt-vNxy SEQUENCE (SIZE (1..maxNrofListElementsDiff-rN)) OF ListElement OPTIONAL, -- Need N

 listElementToReleaseListSizeExt-vNxy SEQUENCE (SIZE (1..maxNrofListElementsDiff-rN)) OF ListElementId OPTIONAL -- Need N

 ]]

}

-- 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 and the element structure type name (before the -vNxy suffix), and a parallel ToAddMod 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 releasable without release and add of the entire list element; this can, for instance, be ensured by having the new fields be OPTIONAL Need R. If multiple extensions of the same list are needed, the version suffix should distinguish the lists (e.g. *listElementToAddModListExt-vNwz* added after *listElementToAddModListExt-vNxy*). The result is as shown in the following example:

-- /example 2/ ASN1START

ContainingStructure ::= SEQUENCE {

 listElementToAddModList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElement OPTIONAL, -- Need N

 listElementToReleaseList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Parallel list

 listElementToAddModListExt-vNxy SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementExt-vNxy OPTIONAL -- Need N

 ]],

 [[

 -- Second parallel list from a later spec version

 listElementToAddModListExt-vNwz SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementExt-vNwz OPTIONAL -- Need N

 ]]

}

ListElement ::= SEQUENCE {

 elementId ListElementId,

 field1 INTEGER (0..3),

 field2 ENUMERATED { value1, value2, value3 }

}

ListElementExt-vNxy ::= SEQUENCE {

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

}

ListElementExt-vNwz ::= SEQUENCE {

 field4-rN INTEGER (0..255) OPTIONAL -- Need R

}

-- ASN1STOP

– When the size 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 in example 1 above*.* Note that if the list element ID type changes in this case, the new ID can be added after the extension marker, and the entries of the size-extended ToRelease list should have the type of the new ID (e.g. *ListElementId-vNxy*). 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 and a parallel list with ToAddMod introduced to hold the extension structures, as in the second example above, for entries of the original list and for entries of the extension list holding new entries. 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 needed, but no additional parallel ToRelease list is needed (i.e. there is no *listElementToReleaseListExt-vNxy* in the example below), as the original and extended ToRelease lists suffice to release any element of the combined list. The extended element ID type should be captured as a non-critical extension of the original element ID type, with the field description indicating that if the extended ID is present, the original ID is ignored. The result is as shown in the following example:

-- /example 3/ ASN1START

ContainingStructure ::= SEQUENCE {

 listElementToAddModList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElement OPTIONAL, -- Need N

 listElementToReleaseList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Non-critical extension lists

 listElementToAddModListSizeExt-vNxy SEQUENCE (SIZE (1..maxNrofListElementsDiff-rN)) OF ListElement OPTIONAL, -- Need N

 listElementToReleaseListSizeExt-vNxy SEQUENCE (SIZE (1..maxNrofListElementsDiff-rN)) OF ListElementId-vNxy OPTIONAL, -- Need N

 -- Parallel list with maxNrofListElements-rN = maxNrofListElements + maxNrofListElementsDiff-rN

 listElementToAddModListExt-vNxy SEQUENCE (SIZE (1..maxNrofListElements-rN)) OF ListElementExt-vNxy OPTIONAL, -- Need N

 ]]

}

ListElement ::= SEQUENCE {

 elementId ListElementId,

 field1 INTEGER (0..3),

 field2 ENUMERATED { value1, value2, value3 }

}

ListElementExt-vNxy ::= SEQUENCE {

 -- Field description should indicate that if the elementId-vNxy is present, the elementId (without suffix) is ignored

 elementId-vNxy ListElementId-vNxy OPTIONAL, -- Need S

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

}

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

ListElementId-vNxy ::= INTEGER (maxNrofListElements..maxNrofListElements-rN-1)

-- ASN1STOP

– When different extensions are made to a list in separate releases, the extension mechanisms described above may interact. In case fields are added in Rel-M (*listElementToAddModListExt-vMxy*) and later the list size is extended in Rel-N (*listElementToAddModListSizeExt-vNwz*), the size-extended list in Rel-N should be a single list extending the combination of *listElementToAddModList* and *listElementToAddModListExt-vMxy*.This requires creating a new type (*ListElement-rN*) to contain the combined fields of *ListElement* and *ListElementExt-vMxy*. A corresponding ToRelease list is needed. The result is as shown in the following example:

-- /example 4/ ASN1START

ContainingStructure ::= SEQUENCE {

 listElementToAddModList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElement OPTIONAL, -- Need N

 listElementToReleaseList SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementId OPTIONAL, -- Need N

 ...,

 [[

 -- Parallel list (Rel-M)

 listElementToAddModListExt-vMxy SEQUENCE (SIZE (1..maxNrofListElements)) OF ListElementExt-vMxy OPTIONAL -- Need N

 ]],

 [[

 -- Size-extended list (Rel-N) with maxNrofListElements-rN = maxNrofListElements + maxNrofListElementsDiff-rN

 listElementToAddModListSizeExt-vNwz SEQUENCE (SIZE (1..maxNrofListElementsDiff-rN)) OF ListElement-rN OPTIONAL -- Need N

 listElementToReleaseListSizeExt-vNwz SEQUENCE (SIZE (1..maxNrofListElementsDiff-rN)) OF ListElementId-vNwz OPTIONAL, -- Need N

 ]]

}

ListElement ::= SEQUENCE {

 elementId ListElementId,

 field1 INTEGER (0..3),

 field2 ENUMERATED { value1, value2, value3 }

}

ListElementExt-vMxy ::= SEQUENCE {

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

}

ListElement-rN ::= SEQUENCE {

 elementId-vNwz ListElementId-vNwz,

 field1 INTEGER (0..3),

 field2 ENUMERATED { value1, value2, value3 },

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

}

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

ListElementId-vNwz ::= INTEGER (maxNrofListElements..maxNrofListElements-rN-1)

-- ASN1STOP