
# Using ASN.1 to Describe 3GPP Layer 3 Message Formats

*Objective Systems, Inc., July 2012*

## Overview

The 3rd Generation Partnership Project (3GPP) publishes many standards documents that describe message formats for signaling and other applications within 3rd generation and 4th generation (LTE) wireless networks. Many of these use Abstract Syntax Notation 1 (ASN.1) to describe the message formats including Radio Resource Control (RRC), Node B Application Part (NBAP), etc.

Other formats such as those within the Non-Access Stratum (NAS) layer are described using a tabular format. These are known as Layer 3 messages and consist of a set of Information Elements (IE's) assembled under a common header structure. These information elements range from very simple to very complex in structure.

Users of our ASN.1 software products (most notably, the ASN1C code generation tool) have expressed an interest in being able to process messages of this type within their existing ASN1C-based framework. They have also expressed interest in looking for ways to further automate the production of encoders and decoders for these message types due to the fact that working with tabular data directly is tedious and error-prone and the fact that these specifications change often making forward-going maintenance imperative.

We have therefore researched ways of expressing these message types in ASN.1 format for use with our tools. We found that the majority of structures can be directly represented using existing ASN.1 types and encoding rules very similar to the Packed Encoding Rules (PER) used to encode and decode the components. There are some corner cases however, in which it would be very difficult if not impossible to automate the production of encoders and decoders. For these cases, we have added additional flexibility to the code generation process that allows substitution of custom code for either complete encode/decode functions or, at a finer level, for individual elements within functions.

This paper describes the changes we have made to ASN1C to allow 3GPP L3 messages to be processed in the same way as any other ASN.1-based format. By adding the special logic to produce encoders and decoders for these formats, users can benefit from the generation of existing utility code for other common operations including printing, copying, comparing, and test code generation.

## Why ASN.1?

Other modeling schemes have been proposed for the automation of 3GPP message parsing and formatting. The following are arguments for using ASN.1 for this task:

- It is a well understood, standardized syntax used in other 3GPP message format specifications.
- Most constructs for Information Elements and Message can be directly expressed using the standard syntax.
- Allows automation of the task of producing encoders and decoders.
- Code generation tools can also generate supporting code for print, copy, compare, and test code generation.

## Modeling 3GPP Information Elements

The first items to be taken into consideration when designing this capability are Information Elements (IE's). We will use examples from 3GPP TS 24.008, Section10 which defines Information Elements used within the 3G NAS. It should be possible to carry over these concepts to other specifications that

define messages in similar ways.

Information Element structures are defined in 2 ways:

- Through bit field diagrams with accompanying descriptive text
- Through the use of "Concrete Syntax Notation 1" (CSN.1), a syntax for defining complex bit patterns

How each of these are defined using ASN.1 and the configuration items that have been added to deal with special cases follows.

## Configuration Items

Before proceeding to the details on how Information Elements are represented in ASN.1 for use with ASN1C, we first examine a change to existing configuration directive processing. The current custom configuration capability in ASN1C v6.5 and below uses an external XML file that targets specific items within a specification for special processing. For example, the following external configuration file targets element octet5a within the BackupBearerCapOctetGroup5 production within the TS24008IES module for special processing:

```
<asn1config>
  <module name="TS24008IES">
    <production name="BackupBearerCapOctetGroup5">
      <element name="octet5a">
        <end3GExtElem name="octet5.ext"/>
      </element>
    </production>
  </module>
</asn1config>
```

The production that is targeted in this case is as follows:

```
BackupBearerCapOctetGroup5 ::= SEQUENCE {
   octet5 BearerCapOctet5,
   octet5a BearerCapOctet5a
}
```

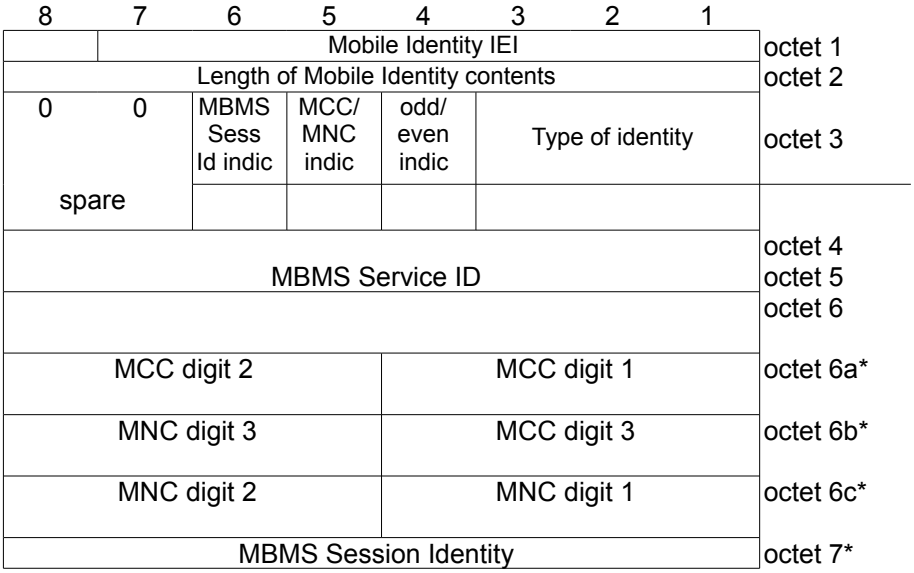
It is now possible to embed the configuration directive for the item being targeted directly as an ASN.1 comment preceding it:

```
BackupBearerCapOctetGroup5 ::= SEQUENCE {
   octet5 BearerCapOctet5,
   --<end3GExtElem name="octet5.ext"/>
   octet5a BearerCapOctet5a
}
```

This is a more practical and compact notation that can be used if users are free to edit the ASN.1 specifications they are dealing with. An external configuration is good if users are working with standardized ASN.1 that they are not free to edit. The remainder of this paper will show directives in the embedded notation, although either form may be used to apply the directives.

## Bit Field Diagram Described Items

An example of a 3GPP IE bit field diagram is as follows:




### *Fixed-size Bit Fields*

The simplest items to model within these descriptions are single-bit and fixed-size bit fields. A single bit is modeled using an ASN.1 BOOLEAN type. Fixed-size fields are modeled using constrained INTEGER types. For example, a two-bit field would be of type INTEGER (0..3), a three-bit field would be INTEGER (0..7), etc. The encoding in this case would be identical to that of PER – the value would be packed into the minimal width field represented by the range. It is also possible to model these fields using fixed-size BIT STRING types, but the standard use of these items would appear to be as numeric variables, making INTEGER the better choice.

The specification of bit fields typically also includes the specification of spare bits either for the purpose of padding to octet boundaries or for future extensions. These can be handled in ASN.1 by either creating a separate element for them (typically a constrained INTEGER with a DEFAULT value of zero), or by combining the width with the adjacent bit field to form a larger constrained INTEGER value. The advantage of the former is better constraint checking to ensure the value is within the defined range whereas the latter can provide more compact code as processing for the fields is combined.

### *Octet Fields*

Message fields are sometimes described as a fixed-size or variable-sized array of octets (8-bit bytes). An example is the MBMS Service ID field in the bit field diagram above.

These are modeled using the ASN.1 OCTET STRING type with a size constraint specification. If the array is of fixed-size, the ASN1C -strict-size command-line option can be used to insert an array of the given size without a length qualifier.

### *TBCD String*

The Telephony Binary Coded Decimal (TBCD) String type is a common type used in 3GPP L3 messages as well as other telephony standard applications. Unfortunately, there is no built-in support for this type in ASN.1. A special configuration directive - <isTBCDString/> - has been added that allows a type to be designated to be a TBCD string. The ASN1C compiler generates a standard character string type for these items and handles the conversion to and from packed format (one digit per 4 bits) on encoding and decoding.

### *MCC/MNC Pair Type*

Related to TBCD String is the MCC/MNC pair type. This type consists of a pair of TBCD numbers stored back-to-back. Due to the way the inner 2 digits are packed (one from MCC and one from MNC), ASN1C was unable to generate a proper encoder/decoder by default. For this reason, a custom built-in encode/decode function was created to handle this special case.

An example of the MCC/MNC pair declaration is shown in the bit field diagram above in octets 6a through 6c.

### *Enumerated Types*

Type definitions within IE structures sometimes define a set of alternatives. For these cases, the ASN.1 ENUMERATED type can be used. If the number of alternatives is large enough to cover the full range of the defined bit field, the enumerated type definition itself without modification is sufficient for generation of an encoder and decoder for the range. For example, it the maximum defined enumerated value is 11 and the bit field size is 4, the code generator will create the proper sized field because ranges between 8 and 15 inclusive result in a 4 bit field.

If, on the other hand, there are not enough alternatives to cover the full range, an adjustment to the type definition is necessary. This can be done in one of two ways:

1. Define a dummy maximum enumerated identifier that is large enough to fully define the range, or
2. Add a value range subtype constraint to the type definition that defines the bit field range.

For example, if the maximum enumerated value is 6 and the bit field width is 4, the following can be done to ensure the bit field is the correct width:

```
-- option 1: add dummy identifier

MyEnumType ::= ENUMERATED {
   myEnum1(1), myEnum(2), -- other defined items
   myEnumType-MAX(15) -- dummy identifier to pad field
}
```

or

```
-- option 2: add value range constraint

MyEnumType ::= ENUMERATED {
   myEnum1(1), myEnum2(2) -- other defined items
} (0..15) -- value range constraint to define field size
```

### *SEQUENCE*

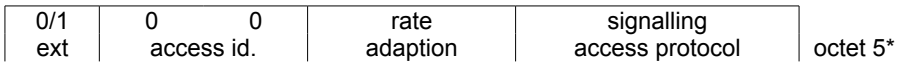
SEQUENCE is the standard collection type used to define the full set of bit fields in a standard bit field

diagram. The fields are simply declared in the order they appear in the diagram from left-to-right, top-to-bottom.

All fields within a bit field diagram are always required to be present at the specified location. The ASN.1 concept of optional fields may be defined using Concrete Syntax Notation. This is discussed later in the paper.

### *Extension Elements*

Sometimes an element will be designated to be an *extension element*. This is a single-bit element which when set to zero signals the presence of a group of elements following it. The general form of this type of element is as follows:




In order to delimit the group of elements to which an extension element applies, the "<end3GExtElem>" configuration directive was added:

```
<end3GExtElem name="name"/>
```

where name would be replaced with the name of the element which is used to delimit the group.

### *SEQUENCE OF*

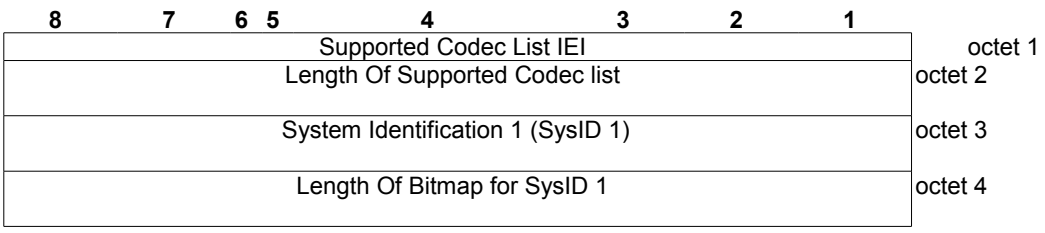
A shortcoming of the definition of repeating items in L3 IE's is that a standard, well-defined, consistent pattern is not established as it is in ASN.1 encoding rules. Depending on whether bit-field diagrams or Concrete Syntax Notation is used, at least two different patterns for representing repeating items have been identified:

1. Use of a preceding length field that defines the number of items in the list, or
2. Use of an extension bit based approach in which a bit either within or external to the structure is used to indicate either the presence of another item or end-of-list.

Furthermore, on item 2, even the convention on which bit value signals end-of-list is not consistent. In some cases, a zero bit signal end-of-list, in other cases it is a one bit.

For this reason, new configuration directives have been introduced to specify the type of list being created. For case 1, the "<is3GVarLenList>" configuration variable identifies the repeating item as a variable length list. This is the primary mechanism used in bit field diagrams and will be discussed here.

An example of a variable length list in TS 24.008 is the Supported Codec List IE which is defined as follows:



| | |
|---|---|
| Codec Bitmap for SysID 1, bits 1 to 8 | octet 5 |
| Codec Bitmap for SysID 1, bits 9 to 16 | octet 6 |
| System Identification 2 (SysID 2) | octet j |
| Length Of Bitmap for (SysID 2) | octet j+1 |
| Codec Bitmap for (SysID 2), bits 1 to 8 | octet j+2 |
| Codec Bitmap for (SysID 2), bits 9 to 16 | octet j+3 |
| System Identification x (SysID x) | octet m |
| Length Of Bitmap for (SysID x) | octet m+1 |
| Codec Bitmap for (SysID x), bits 1 to 8 | octet m+2 |
| Codec Bitmap for (SysID x), bits 9 to 16 | octet m+3 |

In this case, the length element in the IE determines the total length of the IE in bytes. This element is identified in the "<is3GVarLenList>" directive using the lengthElem attribute. This item governs the overall length of the IE. On parsing, it is used to determine the number of elements in the repeating structure. For encoding, the length in the populated structure is ignored and the total length of the IE is calculated based on the number of elements in the populated structure.

The ASN.1 type used to represent the Supported Codec List is as follows:

```
SupportedCodecList ::= SEQUENCE {
   length INTEGER (4..254),
   --<is3GVarLenList lengthElem="length"/>
   codecs SEQUENCE (SIZE (1..83)) OF SupportedCodec
}
```

Case 2 is identified using the "<is3GExtList>" directive. These types of lists are defined in both bit field diagrams and using Concrete Syntax Notation. In bit field diagrams, an extension list is specified in much the same way as extension elements defined above. In this case, however, descriptive text is added to indicate the extension may repeat more than once and will terminate when a one bit is in the extension element bit field.

This is modeled in ASN.1 by using a SEQUENCE OF type to which the following configuration directive is applied:

```
<is3GExtList pre-eol="1">
```

This indicates the field will repeat until an end-of-list bit equal to 1 is encountered. The attribute is named "pre-eol" to indicate the bit precedes the last record. There are cases that can be described using CSN.1 where the bit indicator comes after the record. This is the "post-eol" case and is described in the section on Concrete Syntax Notation.

An example of this type of specification can be found in TS 24.008, Section 10.5.4.5 – Bearer Capability. The specification of the repeating octet 3 type is modeled this way. It is shown in the table as follows:

| | | | | | |
|---|---|---|---|---|---|
| 0/1 ext | 0 co-ding | CTM | 0 spare | speech version indication | octet 3a * |
| 0/1 | 0 | 0 | 0 | | |

| ext | co-ding | spare | spare | Speech version Indication | octet 3b etc* |
|---|---|---|---|---|---|

the ASN.1 used to model this is as follows:

```
BearerCapability-Octet3Ext ::= SEQUENCE {
   coding BOOLEAN,
   spare INTEGER (0..3) DEFAULT 0,
   speechVersion INTEGER (0..15)
}

--<is3GExtList pre-eol="1"/>
BearerCapability-Octet3ExtArray ::=
   SEQUENCE (SIZE(0..8)) OF BearerCapability-Octet3Ext
```

Note that the 'ext' element is not included in the element type definition. It is assumed to occur in the first position.

## Items Described Using Concrete Syntax Notation

More complex IE structures are defined using "Concrete Syntax Notation". An overview of this notation is available in Annex B of TS 24.007. There is also a basic tutorial located at the following URL:

http://csn1.info

Most items in CSN.1 can be directly represented in ASN.1. There are some items that require the use of new configuration directives and some that require custom code be developed to handle. We examine certain patterns identified within TS 24.008 and the ASN.1 used to model them.

### *Fixed-size Bit Fields*

CSN.1 can be used to describe fixed-size bit fields. The syntax is simply `< name : bit >` for a single-bit field, or `< name : bit (n) >` for a multiple-bit width field. In this definition, *name* would be replaced by a CSN variable name and *n* would be replaced with a bit count value.

As was the case with bit field diagram defined values, a single-bit field is modeled with the ASN.1 BOOLEAN type and a multiple-bit field is modeled with an INTEGER type with a value range constraint that defines the full set of numeric values the bit field will hold.

### *Enumerated Types*

Enumerated bit patterns can be defined in CSN.1 using the union (|) operator. The general form is:

`< name : { pattern1 | pattern2 | … | patternN } >`

where *name* would be replaced with an identifier name and each *pattern* item would be replaced with a bit pattern. For example, the following definition exists within TS 24.008:

```
< T-GSM 400 Bands Supported : { 01 | 10 | 11 } >
```

In this case, an ASN.1 ENUMERATED type can be used to model the item. Note that there are no names for the various alternatives and the type (or element) name in this case is in a rather free form format which includes embedded whitespace. To model this with an ASN.1 type would require modification of the name and the addition of identifiers for the enumerated items. A type that could be used for this purpose is as follows:

```
T-GSM-400-Bands-Supported ::= { band1(1), band(2), band(3) }
```

The modifications would have no effect on the final encoding.

### *SEQUENCE*

As was the case for bit field diagrams, the primary container for a series of items expressed in CSN.1 is a SEQUENCE type.

A common pattern also exists in CSN.1 definitions of IE's for optional elements. This pattern is as follows:

```
{ 0 | 1 < name > }
```

In this case, *name* would be the name of a defined item or a complete CSN.1 element definition. This indicates that if a zero bit is present, the encoding of the definition is omitted; whereas, if a one bit is present, it is followed by the encoded item.

This is modeled in ASN.1 as an optional element in a SEQUENCE. The code generator by default will generate code to implement this pattern for optional elements.

Note that an optional element in an IE is treated differently than an optional element in an L3 message. In the case of a message, it signals a non-imperative IE within the message. This will be covered in more detail in the section on 3GPP Messages below.

### *SEQUENCE OF*

It is possible to express repeating items in CSN.1, the problem from an ASN.1 modeling standpoint is that it is not done in a consistent way in the 3GPP specifications. As mentioned previously in the Bit Field Diagram section, one way this is done is through the use of a variable length list in which a length element within the IE determines the number of encoded items.

In CSN.1, a common way of specifying repeating items is through an extension list. In this case, a bit value is used to signal the presence of another record in the repeating collection or the end of the list. One form of this syntax is as follows:

```
< name > ::= { 0 : < type > } ** 1;
```

This indicates that the named item is a repeating collection of items of type *type.* The presence of a zero bit indicates that a typed item follows; a one bit signals end-of-list. To express this in ASN.1, a configuration directive in the following form is applied to the SEQUENCE OF definition:

```
<is3GExtList post-eol="1">
Name ::= SEQUENCE OF Type
```

The “is3GExtList” directive indicates that the type of list is an extension list as opposed to the variable length list type defined earlier. The “post-eol” attribute indicates the list is terminated by the presence of a one bit *after* the last record. There is also a “pre-eol” attribute that indicates a list is terminated by the presence of a given bit value *before* the last record. This was defined earlier in the section on bit field diagram descriptions.

### *CHOICE*

It is possible to create a choice of items in CSN.1 in a similar fashion to how optional elements are declared. The union operator (|) is used followed by a bit pattern that identifies each item followed by the type of the item that is to follow:

```
< name > ::= { bit-pattern-1 : < type-1 > |
               bit-pattern-2 : < type-2 > |
               …
               bit-pattern-n : < type-n > }
```

This can be directly modeled with an ASN.1 choice type as long as the bit patterns that identify the alternatives form a sequential set of numbers (for example, 0, 1, or 00, 01, 10, etc.). This has been the case in all observed patterns of this type to date, so a configuration directive to handle the case of non-sequential numbers has yet to be developed.

## Modeling 3GPP Messages

The basic structure of 3GPP Layer 3 messages is a common header structure as defined in TS 24.007 followed by a series of Information Elements.

### Header

Components that make up the header structure can be defined in ASN.1 as follows:

```
/* 11.2.3.1 Standard L3 Message Header */
L3HdrOptions ::= CHOICE {
   skipInd INTEGER(0..15),
   --<inline/>
   transId SEQUENCE {
      flag BOOLEAN,
      value INTEGER (0..255)
   },
   epsBearerIdent EPSBearerIdentity
}

ProtoDiscr ::= ENUMERATED {
   callCtrl(3), mobMgmt(5), rrMgmt(6), epsMobMgmt(7), gprsMobMgmt(8),
   sms(9), sessMgmt(10)
} (0..15)

ProtocolIE-ID ::= SEQUENCE {
   protoDiscr   ProtoDiscr,
   msgType      INTEGER(0..255)
}
```

Some definitions have been removed for brevity.

Within the header structure are protocol discriminator and message type fields that identify the contents of the body of the message. These are contained within the “ProtocolIE-ID” type. This type is used in the following information object class definition as a key value to form a relation between this item and the body of the message:

```
NAS-PROTOCOL-CLASS ::= CLASS {
```

```
   &id ProtocolIE-ID,
   &Value
}
WITH SYNTAX {
   ID &id
   TYPE &Value
}
```

Using these items, we can come up with a common definition for a message protocol data unit (PDU):

```
PDU ::= SEQUENCE {
   l3HdrOpts    L3HdrOptions,      -- L3 header, octet 1, bits 5 thru 8
   id           NAS-PROTOCOL-CLASS.&id ({PROTOCOL-IE-OBJECTSET}),
   data         NAS-PROTOCOL-CLASS.&Value ({PROTOCOL-IE-OBJECTSET}{@id})
}
```

The PROTOCOL-IE-OBJECTSET would then contain all of the ID's and message types for all messages defined in the protocol.

### *Messages*

The body of an L3 message is described using an ASN.1 type definition. A relation is then set up between the protocol ID and this type by defining an information object for the message and then this object is added to the PROTOCOL-IE-OBJECTSET.

The type that is created is normally a SEQUENCE of the Information Elements that are identified in the table describing the message, although in simple cases the body may be empty (header only) in which case an ASN.1 NULL type is used and in others, only a reference to a single Information Element type is required.

The Information Elements that make up a message are specified in tables within the specification in which they are defined. The rows in the table define the imperative and non-imperative parts of the message.

The imperative part defines mandatory elements. These are defined first. In ASN.1, these items are defined as non-optional elements in the container SEQUENCE. No special processing is required to handle them in encoding or decoding. They are simply processed in the order they are declared.

The non-imperative part of a message begins with the first item in the table in which an IEI (Information Element Identifier) value is declared. These elements are declared as OPTIONAL in the ASN.1 definition. All elements after this first optional element will be included in the non-imperative part and must be declared to be optional as well. It is not possible to mix mandatory and optional elements.

In addition to declaring the non-imperative element to be optional, a special configuration directive is required to provide additional information. The general form of this directive is as follows:

```
   --<iei format="t|tv|tlv" [length="length"]>hex-value</iei>
```

In this definition, the format attribute is either 't' (tag), 'tv' (tag value), or 'tlv' (tag-length-value) as defined in TS 24.007. The length attribute is optional and only required for format type 'tv' to identify the length of the value. For format 'tlv', the length value will be built into the generated type. The *hex-value* would be replaced with the actual hexadecimal value as defined in the table. No conversion to

decimal format is necessary.

### Example

The following is a complete example in defining a 3GPP L3 message in ASN.1 form for use with our tools. The message in this case is the Location Updating Accept message defined in 3GPP TS 24.008, Section 9.2.13:

| IEI | Information element | Type/Reference | Presence | Format | Length |
|---|---|---|---|---|---|
| | Mobility management protocol discriminator | Protocol discriminator 10.2 | M | V | 1/2 |
| | Skip Indicator | Skip Indicator 10.3.1 | M | V | 1/2 |
| | Location Updating Accept message type | Message type 10.4 | M | V | 1 |
| | Location area identification | Location area identification 10.5.1.3 | M | V | 5 |
| 17 | Mobile identity | Mobile identity 10.5.1.4 | O | TLV | 3-10 |
| A1 | Follow on proceed | Follow on proceed 10.5.3.7 | O | T | 1 |
| A2 | CTS permission | CTS permission 10.5.3.10 | O | T | 1 |
| 4A | Equivalent PLMNs | PLMN list 10.5.1.13 | O | TLV | 5-47 |
| 34 | Emergency Number List | Emergency Number List 10.5.3.13 | O | TLV | 5-50 |

The ASN.1 definition of this message would be as follows:

```
LocUpdatingAccept ::= SEQUENCE {
   locAreaIdent LocAreaIdent,
   --<iei format="tlv">17</iei>
   mobileIdentity MobileIdentity OPTIONAL,
   /* followOnProceed and ctsPermission contain no data, just IEI (tag) -
      present flags are sufficient to indicate presence or absence */
   --<iei format="t">A1</iei>
   followOnProceed NULL OPTIONAL,
   --<iei format="t">A2</iei>
   ctsPermission NULL OPTIONAL,
   --<iei format="tlv">4A</iei>
   equivPLMNs PLMNList OPTIONAL,
   --<iei format="tlv">34</iei>
   emergNumList EmergencyNumberList OPTIONAL
}
```

The “locAreaIdent” element is the lone mandatory element that makes up the imperative part of the message. The remaining elements form the non-imperative part and are therefore marked as optional. Each one is preceded by an “iei” configuration directive. Note that the “followOnProceed” and “ctsPermission” elements are of format tag-only and therefore have a type field of NULL.

We then can define a value constant for the message type and combine this with the protocol discriminator and message type to form an information object:

```
mt-LocUpdatingAccept INTEGER ::= 2


obj-LocUpdatingAccept NAS-PROTOCOL-CLASS ::= {
   ID { protoDiscr mobMgmt, msgType mt-LocUpdatingAccept }
   TYPE LocUpdatingAccept
}
```

Message types for all of the various message defined in TS 24.008 are declared as bit patterns in section 10.4. This must be converted into a decimal integer value for use in the ASN.1 definition.

The information object can then be included in the PROTOCOL-IE-OBJECTSET which defines all messages that make up the protocol.

## Inserting Custom Code

The methodology defined above can be used to automate a large part of the task of producing encoders and decoders for L3 messages. Despite this, there are still some odd and quirky parts in the specifications that cannot be easily automated. For these cases, new configuration options are being introduced which allow custom code to be inserted in place of what would normally be generated by the code generator.

The first of these directives allows complete encode or decode functions to be replaced for a given type. The variable names are “<cEncFuncFname>” and “<cDecFuncFname>” respectively. These allow the contents of the C source file that is specified as the argument to be inserted in place of the encode or decode function that would normally be generated. Note that C is currently the only supported language for this capability.

In addition, the “<cEncSrcFname>” and “<cDecSrcFname>” directives can be used to insert a code snippet for what would have been generated for a given element. These also take the name of a file containing the code to be inserted as an argument.