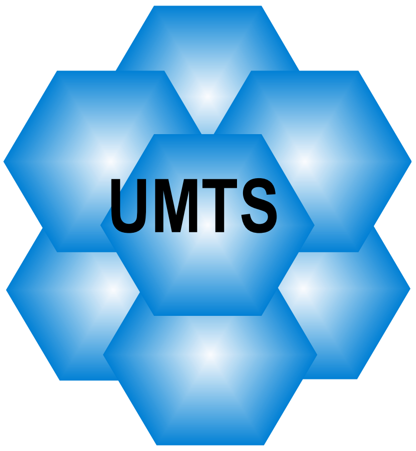
condat®



G23-UMTS Protocol Stack

Introduction to ASN.1

Author:	Condat AG Alt Moabit 90a 10559 Berlin Germany

Date:	9 October, 2001
ID:	9999.999.99.001
Status:	Planned

Condat Proprietary Information
NDA - Confidential
Do Not Copy

Table of Contents

0 Document Control

© Copyright Condat AG, 2001
All rights reserved.

Every effort has been made to ensure that the information contained in this document is accurate at the time of printing. However, the software described in this document is subject to continuous development and improvement. Condat AG reserves the right to change the specification of the software. Information in this document is subject to change without notice and does not represent a commitment on the part of Condat AG. Condat AG accepts no liability for any loss or damage arising from the use of any information contained in this document.

The software described in this document is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. It is an offence to copy the software in any way except as specifically set out in the agreement. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Condat AG.

Condat AG
Alt Moabit 90a
10559 Berlin
Germany

Telephone:	+49.30.39 49 0
Fax:	+49.30.39 49 1300

Internet: www.condat.de

0.1 Document History

ID	Author	Date	Status
9999.999.99.001	JHO	9 October, 2001	Planned

0.2 References, Abbreviations, Terms

[C_7010.801]	7010.801, References and Vocabulary, Condat AG

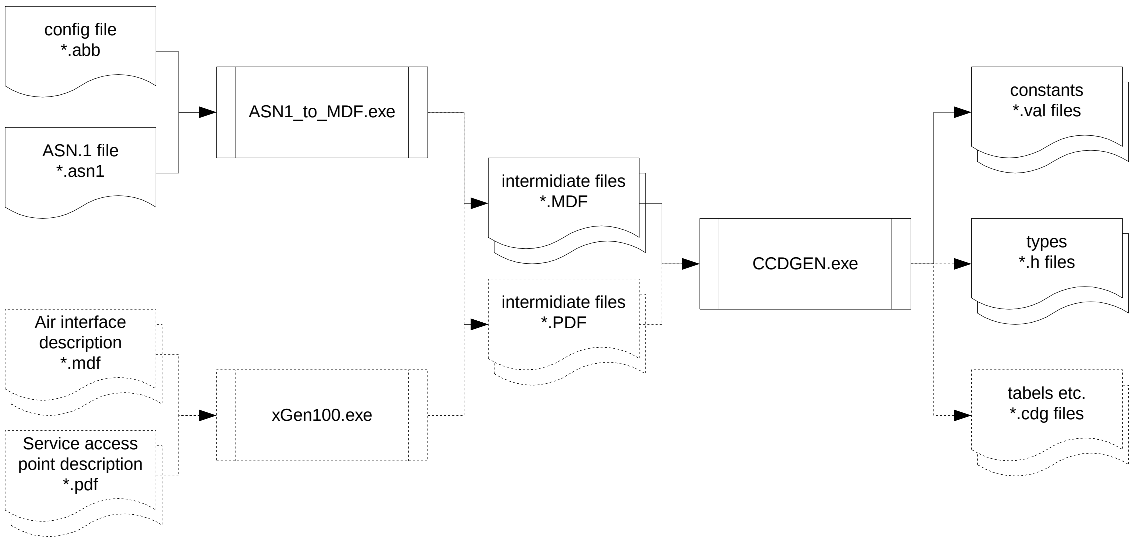
1 Introduction

The purpose of this document is to give a brief introduction to how the ASN.1 to MDF compiler and CCDGEN works together to produce an h-file

Knowledge of ASN.1 and C is necessary to understand this document
Basic knowledge of BNF syntax description and regular search expressions[1] is recommended for chapter 3 as well

However only knowledge of C, UMTS and the stuff mentioned in chapter 2 should be needed to use the include files[2] that is the final product the 2 compilers ASN1_to_MDF.exe and CCDGEN.exe.

The following drawing shows the tool and filles related to ASN.1 compilation



The command line syntax of ASN1_to_MDF is simple it takes one parameter, the name of the ASN.1 file the config file must have the same base name as the ASN.1 file

[1] Known from perl, awk, sed, phyton, and many editors
[2] h-files and val-files

2 Facilities

2.1 Name transformation

Names are transformed from the ASN.1 form to the C form through several steps

1) Create proper C-identifiers (done by

   a. Abbreviate words and sequences of words according to abbreviating list

   b. Replace dash (-) with underscore (_)

   c. Insert underscore on all word boundaries where not already present

   d. Make all character lowercase

ASN.1 example	**C example**
UPPERMix	upper_mix
UPPER-lowerMix2	upper_lower_mix_2

2) If this is an unnamed embedded type created a name by concatenating container name and member name separated by **2** underscores (__)

ASN.1 example	**C example**

outer SEQUENCE {
  inner SEQUENCE { -- outer__inner, 2 underscores!
    i INTEGER (0..255)
  }
}
	typedef struct {
	  U8 i;
	} T_PREFIX_outer__inner;
	typedef struct {
	  T_PREFIX_outer__inner inner;
	} T_PREFIX_outer;

other SEQUENCE {
  inner SEQUENCE { -- other__inner, 2 underscores!
    i INTEGER (0..255)
  }
}
	typedef struct {
	  U8 i;
	} T_PREFIX_other__inner;
	typedef struct {
	  T_PREFIX_other__inner inner;
	} T_PREFIX_other;

3) If this is an implicit counter (c_), valid-flag (v_), or union-controller (ctrl_) add the corresponding prefix

   see example in:

   v_: 2.3.5.1 SEQUENCE element

   c_: 2.3.6 BIT STRING, 2.3.7 OCTET STRING, 2.3.8 SEQUENCE OF

   ctrl_: 2.3.9 UNION

4) Add module prefix (UNTS_AS_ASN1_)

5) Add T_ prefix for types

2.1.1 Compiler responsible for transforming

ASN1_to_MDC does step 1, 2 and c_ of 3.

CCDGen does step v_ and ctrl_ of 3 and 4 and 5.

2.2 Configuration file syntax (*.abb for abbreviation)

The abbrivation file contains additional information needed by CCD/CCDGen but not present in the ASN.1 syntax:

These are abbreviations, top level types, where to put pointers

The syntax of the file is line oriented and "#" is used for commenting out the reset of the line, empty lines are ignored.

2.2.1 Top level types

Top level types lines list top level type, direction and message id.

These identifies types for which CCD encoder / decoder function can be called and information to be given to CCD to identify the types when actual calling CCD.

Syntax:

```
TopLevelTypeSpec        :    TypeIdentifier "=" (  "downlink" | "uplink" ) CCDMessageTypeValue
                        ;
TypeIdentifier          :    /[A-Z][A-Za-z0-9]*(-[A-Za-z0-9]+)*/
                        ;
CCDMessageTypeValue     :    /[0-9]+/
                        ;
```

Semantic constraint:

CCDMessageTypeValue must be in range 0..255

2.2.2 Abbreviations

Abbreviation was introduced due to the following facts:

Names have been constructed for anonymous inner types for the sake of c-compiler and other tools and to help a little on readability for such names it was found necessary to abbrivate common words in such a way that no type name was longer than approximately 100 characters.

For some common word it was kind of random if they was abbrivated i.e. in some context they was and in some context they was not.

The abbreviation list is also used to split words that is not detected as concatenated words due to odd usage of upper/lower cases.

Syntax:

```
AbbrivationSpec         :    AbbrivationExpression
                        |    AbbrivationExpression PartialCIdentifier
                        ;
AbbrivationExpression   :    /[-:/!^]?[A-Za-z0-9]+([-:/!][A-Za-z0-9]+) [-:/!$]?/
                        ;
PartialCIdentifier      :    /[A-Za-z0-9_]+/
                        ;
```

Semantic behaviour for AbbrivationSpec:

use '^' to test for start of identifier

use '$' to test for end of identifier

use ':' to test for class scope seperator

use '!' to test for tag member separator (between enumerated class name and enumerator name at inner most level)

use '-' to test for '-'

if a word start with upper case and rest is lower case then case are ignored for first letter

use '/' to test for word bounbary (this is optional)

Word boundary is always detected in the following cases:

Before and after a number	e.g. ASN2MDF == ASN/2/MDF
Just before a single upper case char	e.g. firstCharUpperCaseWords == first/char/upper/case/words
At special chars	e.g. dash-separted-word == dash/separted/word
Just before several uppercase chars	e.g. myEG == my/EG

Word boundary is detected based on word content

After or before last of several uppercase chars if the word on both sides is known and there is only one way to split	e.g.assuming known words: ASN, MDF ,words ASNwords-to-MDF Words == ASN/words/to/MDF/Words
	e.g. assuming known words: PC, PCP, reconf, preconf can't split PCPreconf is it PCP/reconf or PC/Preconf? e.g. assuming unknow word: ab can't split PCab (to much assumption)

Word boudary is NOT detecte

When all chars is lowercase	e.g. lowercase -> lowercase; NOT lower/case

If no PartialCIdentifier is present the word(s) is protected against replacement (but word splitting will still occur).

2.2.3 Pointer configurations

If flat types was generated from the ASN.1 types the size of the biggest of theses would be more than 1 mega byte, however this is a rather academic figure since it comes from e.g. multiple levels of variable sized arrays, and theses are constraint by non ASN.1 visible constraints to not all be of maximal size.

Hence a method for making some struct elements pointers have been introduced:

Syntax:

```
PointerMemberSpec          :    QualifiedMemberIdentifier “=” “dynamic”
                           ;
QualifiedMemberIdentifier  :    TypeIdentifier NestedMemberIdentifier
                           ;
NestedMemberIdentifier     :    MemberIdentifier
                           |    NestedMemberIdentifier “:” MemberIdentifier
                           ;
MemberIdentifier           :    /[a-z][A-Za-z0-9]*(- [A-Za-z0-9]+)*/
```

;

2.2.4 Static analyse configuration

This facility is for TI internal use only

Syntax:

```
StaticAnalyseSpec          :    MemberIdentifier "=" "capability" NonASN1VisibleConstarint
                           |    "align_size" "=" PositiveNumericValue
                           |    "dyn_size" "=" PositiveNumericValue
                           |    TypeIdentifier "=" "skip"
NonASN1VisibleConstraint   :    /[0-9]+/
```

2.3 Type mapping

2.3.1 BOOL

BOOL are mapped to U8

2.3.2 NULL

NULL are mapped to U8

2.3.3 INTEGER

INTEGER are mapped to one of U8, S8, U16, S16, U32, S32 according to their range

2.3.4 ENUMERATED

An enum is generated each ENUMERATED type (in *.val files)

But this enum is not used in the struct and union types, this are due to the fact that enum best fit is not supported by the target compiler

2.3.5 SEQUENCE

SEQUENCE are mapped to a struct unless they are empty in which case they are mapped to U8

ASN.1 example	**C example**

S1 ::= SEQUENCE {
  something INTEGER (0..65535),
  extension SEQUENCE {}
}
	typedef struct {
	  U16 something;
	  U8 extension;
	} T_…_s_1;

2.3.5.1 SEQUENCE element

SEQUENCE elements are coded in 2 different ways depending on their type.

INTEGER, ENUMERATED, BOOLEAN, NULL, and empty SEQUENCE are coded as is, see respective types for examples.

SEQUENCE, BIT STRING, OCTET STRING, and SEQUENCE OF is coded as array of [1]

ASN.1 example	**C example**
S2 ::= SEQUENCE {	typedef struct {

a S1,
b INTEGER
}
	T_…_s_1 a**[1]**;
	U8 extension;
	} T_…_s_2;

The array of 1 (**[1]**) means that you should always use pointers (->) to dereference struct members originating from ASN.1 no matter if it is dynamic or not

2.3.5.2 OPTIONAL SEQUENCE element

OPTIONAL elements can be coded in 2 different ways depending on whether the element is an ordinary element or a dynamic element (pointer)

Ordinary OPTIONAL elements are coded as 2 struct members, where the member value is prefixed with a valid flag named v_*membername*

ASN.1 example	**C example**

i INTEGER (0..255) OPTIONAL	U8 v_i;
	U8 i;

Dynamic OPTIONAL elements are coded as only 1 struct element, which is a point to the OPTIONAL element, the pointer acts a s its own valid flag and if the element is absent the pointer is 0

INTEGER, ENUMERATED, BOOLEAN, NULL, and empty SEQUENCE can not be dynamic.

ASN.1 example	**C example**
i SEQUENCE {…} OPTIONAL	T_…_i *ptr_i;

In general the following rule can be used to determine if there should be checked for presents

Element name	**Optional**
Have v_…	Yes, present if v_… != 0
Is ptr_bits	No, always present
Is ptr_elements	No, always present
Is ptr_…	Yes, present if ptr_… != 0
Else	No, always present[3]

2.3.5.3 DEFAULT SEQUENCE element

Only DEFAULT INTEGER and DEFAULT ENUMERATED are supported, all aspects of the DEFAULT modifier are handle by the tool chain and are fully transparent to the c-code developer, i.e. if during decoding the default flag was set (no value was in the coded data) then CCD is instructed to store the DEFAULT value from the ASN.1 in the corresponding struct member, the opposite approach is taken during encoding.

2.3.6 BIT STRING

BIT STRING are mapped to a **struct** this struct contain a byte array capable of holding the specified number of bits, and if a variable number of bits are specified the struct contains a counter as vel

Type	**ASN.1 example**	**C example**
Fixed size BIT STRING	FBS ::= BIT STRING (SIZE (255))	typedef struct

[3] But if the element have a default value CCD migh have set the default value during decode / will remove the default value during encode.

		{
		U8 bits[255];
		} T_…_fbs;

Variable size BIT STRING	VBS ::= BIT STRING (SIZE (0..255))	struct
		{
		U8 c_bits;
		U8 bits[255];
		} T_…_vbs;

Dynamic variable size BIT STRING	--ASN1
	DBS ::= BIT STRING (SIZE (0..255))
	--ABB: DBS=dynamic
		struct
		{
		U8 c_bits;
		U8 *ptr_bits;
		} T_…_dbs;

2.3.7 OCTET STRING

OCTET STRING are mapped to a **struct** this struct contain a byte array capable of holding the specified number of octets, and if a variable number of octets are specified the struct contains a counter as vel

Type	ASN.1 example	C example

Fixed size OCTET STRING	FOS ::= BIT STRING (SIZE (255))	typedef struct
		{
		U8 elements[255];
		} T_…_FOS;

Variable size OCTET STRING	VOS ::= BIT STRING (SIZE (0..255))	typedef struct
		{
		U8 c_elements;
		U8 elements [255];
		} T_…_VOS;

Dynamic variable size OCTET STRING	--ASN1
	DOS ::= BIT STRING (SIZE (0..255))
	--ABB: DOS=dynamic
		typedef struct
		{
		U8 c_elements;
		U8 *ptr_elements;
		} T_…_DOS;

2.3.8 SEQUENCE OF

SEQUENCE OF are mapped to a **struct** this struct contain an array capable of holding the specified number of elements, and if a variable number of elements are specified the struct contains a counter as vel

Type	ASN.1 example	C example
Fixed size SEQUENCE OF	FOS ::= SEQUENCE (SIZE (255)) OF INTEGER (-128..127)	struct {

		S8 elements[255];
		}

Variable size SEQUENCE OF	VOS ::= SEQUENCE (SIZE (0..255)) OF
	INTEGER (-128..127)
		struct
		{
		U8 c_elements;
		S8 elements [255];
		}

Dynamic variable size SEQUENCE OF	--ASN1
	DOS ::= SEQUENCE (SIZE (0..255)) OF
	INTEGER (-128..127)
	--ABB: DOS=dynamic
		struct
		{
		U8 c_elements;
		S8 *ptr_elements;
		}

2.3.9 UNION

UNION are mapped to a **struct** this struct contain an union capable of holding the specified types of elements, and a controller identifying which element type that is contained

Type	**ASN.1 example**	**C example**

UNION	FU ::= UNION {
	a INTEGER (0..65535),
	b SmallSomething
	}
		typedef enum
		{
		…_FU__A=0,
		…_FU__B=1
		} T_…_ctrl_fu__body;
		typedef union
		{
		U16 a;
		T_…_ small_something b;
		} T_…_fu__body;
		typedef struct
		{
		T_…_ctrl_fu ctrl_body;
		T_…_fu__body body;
		} T_…_fu;

Dynamic UNION	--ASN1
	DU ::= UNION {
	a INTEGER,
	c BigSomething
	}
	--ABB: DU:b=dynamic
		/* T_…_ctrl_du__body is similar to T_…_ctrl_fu__body */
		typedef union
		{
		U16 a;
		T_…_big_something *ptr_b;
		} T_…_fu__body;

		/* T_…_du is similar to T_…_fu */

2.4 Comments

The comments in h-file are on form /* … *asn1-name* (*asn1-type*) *number-of- elements* */

The comments in mdf-file are on form ;“*asn1-name* (*asn1-type*) *number-of- elements*” *debug-info*

The mdf file also contains comments about configuration at the start and name and type statistics at the end.

3 ASN.1 input syntax

The ASN1_to_MDF compiler only accept a subset of the ASN.1 syntax, apart from this a special comment has been introduced to ignore chapter headings from spec and the 2 first lines must contain special ASN.1 line comments with version info.

Here follows BNF for the part of ASN.1 syntax that is supported:

OpenTypes supported too, need to update this BNF accordingly.

(knowledge of regular search expressions[4] might be needed to under stand part of this BNF)

```
Comment             :     /--.*?--/               // ASN.1 comment --...-- like c /*...*/
                    |     /--.*?$/                // ASN.1 comment --...<EOL> like c++ //...<EOL>
                    |     /^11\.[0-9]+.*$/        // NOT ASN.1: introduced to ignore chapter headings from spec
                    ;

ASN1File                  :     /^--Version +(0x[0-9]+)$/          // version number as hex in ASN.1 line comment
                                /^--VersionText +(.*)$/            // version text comment in ASN.1 line comment
                                ModuleList
                          ;
TypeIdentifier            :     /[A-Z][A-Za-z0-9]+(-[A-Za-z0-9]*)*/   //according to ASN.1 specification
ModuleIdentifier          :     /[A-Z][A-Za-z0-9]+(-[A-Za-z0-9]*)*/   //according to ASN.1 specification
ConstIdentifier           :     /[a-z][A-Za-z0-9]+(-[A-Za-z0-9]*)*/   //according to ASN.1 specification
ElementIdentifier         :     /[a-z][A-Za-z0-9]+(-[A-Za-z0-9]*)*/   //according to ASN.1 specification
EnumeratorIdentifier      :     /[A-Z][A-Za-z0-9]+(-[A-Za-z0-9]*)*/   //according to ASN.1 specification


ModuleList                :     Module
                          |     ModuleList          Module
                          ;
Enumerator                :     EnumeratorIdentifier '(' Value ')'
                          |     EnumeratorIdentifier
                          ;
EnumeratorList            :     Enumerator
                          |     EnumeratorList      ','       Enumerator
                          ;
ChoiceElement             :     ElementIdentifier  Type
                          ;
ChoiceElementList         :     ChoiceElement
                          |     ChoiceElementList ','         ChoiceElement
```

[4] The regular expression syntax used here is from perl but only the sequence .*? is special to perl, it means the shortest possible list of any no newline character

```
                        ;
Value                   :   '-' NUMBER
                        |       NUMBER
                        |       Identifier
                        ;
Modifier                :       /*empty*/
                        |       'OPTIONAL'
                        |       'DEFAULT'       Value
                        ;
SequenceElement         :       ElementIdentifier  Type    Modifier
                        ;
SequenceElementList     :       SequenceElement
                        |       SequenceElementList     ','     SequenceElement
                        ;
Range                   :       Value   '..'    Value
                        |       Value
                        ;
Size                    :       '(' 'SIZE'  '('  Range   ')'     ')'
                        ;
Type                    :       'SEQUENCE' '{'  SequenceElementList     '}'
                        |       'SEQUENCE' '{'  /*empty*/               '}'
                        |       'SEQUENCE'      Size    'OF'    Type
                        |       'CHOICE' '{'    ChoiceElementList '}'
                        |       'INTEGER' '('   Range ')'
                        |       'OCTET'  'STRING' Size
                        |       'BIT' 'STRING' '{' EnumeratorList '}' Size
                        |       'BIT' 'STRING' Size
                        |       'NULL'
                        |       'BOOLEAN'
                        |       'ENUMERATED'    '{'     EnumeratorList  '}'
                        |       TypeIdentifierme
                        ;
Options                 :       /*empty*/
                        |       'AUTOMATIC'     'TAGS'
                        ;
Import                  :       IdentifierList  'FROM'  ModuleIdentifier
                        ;
ImportList              :       Import
```

```
                 |      ImportList       Import
                 ;
ImportSection    :      /*empty*/
                 |      'IMPORTS'        ImportList       ';'
                 ;
Definition       :      TypeIdentifier        '::='           Type
                 |      ConstIdentifier  'INTEGER'        '::='      Value
                 ;
DefinitionList   :      Definition
                 |      DefinitionList   Definition
                 ;
Module           :      ModuleIdentifier 'DEFINITIONS'    Options    '::='
                        'BEGIN'
                        ImportSection
                        DefinitionList
                        'END'
                 ;
```