Technical Document

GPF TOOLS

CCDGEN

TECHNICAL DOCUMENTATION

Document Number:	06-03-22-HLL-0001
Version:	0.3
Status:	Draft
Approval Authority:	
Creation Date:	2001-May-15
Last changed:	2015-Mar-08 by SIJ
File Name:	CcdgenTech.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

No part of this document may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying and recording, for any purpose without the express written permission of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
2001-May-15	SKA		0.1		1
2001-Nov-29	SKA		0.2		2
2003-May-20	XINTEGRA		0.3	Draft	

Notes:

1. Draft
2. proceed

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000]	International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1 Introduction

This document describes technical details of ccdgen.

Ccdgen is a LL(1) parser which means, that the parsing starts at the most left token of a line and there is a lookahead of only one token to make a decision how to proceed. Under normal conditions there are no backtrackings to earlier recognized token to get a context free parsed line of tokens. But there are few backtrackings for the values to a variable and for ASN1 counters to ASN1 BITSTRINGS and ASN1 SEQUENCES.

One lack of ccdgen and the syntax of mdf and pdf is that there is no support for name spacing. All names of identifiers are global. Therefore a variable with a given name must always have the same type.

For an easy navigation through the code of ccdgen there is a ccdgen.chm file. The viewer for *.hcm files is "C:\WINNT\hh.exe" where the file suffix hcm is registered to hh.exe by default.

1.1 Abbreviations in *.cdg tables

Red colored entries are new for UMTS and to mark signed types.

Abbr.	meaning	where
“B”	unsigned char	m/pvar.cdg
“C”	signed char	m/pvar.cdg
“L”	unsigned long	m/pvar.cdg
“M”	signed long	m/pvar.cdg
“S”	unsigned short	m/pvar.cdg
“T”	signed short	m/pvar.cdg
“X”	Buffer	m/pvar.cdg
"c"	SDU	m/pelem.cdg
"d"	Pointer (code tranparent) to SDU	m/pelem.cdg
"p"	Pointer to SDU	m/pelem.cdg
“C”	Composition	m/pelem.cdg
“S”	Spare	m/pelem.cdg
“U”	Union	m/pelem.cdg
“V”	Variable	m/pelem.cdg
"P"	Pointer to COMP	m/pelem.cdg
"Q"	Pointer to UNION	m/pelem.cdg
"R"	Pointer to VAR	m/pelem.cdg
"D"	Pointer (code tranparent) to COMP	m/pelem.cdg
"E"	Pointer (code tranparent) to UNION	m/pelem.cdg
"F"	Pointer (code tranparent) to VAR	m/pelem.cdg
	linked elements from pelem.cdg to mvar/mcomp.cdg	
“Z”	Composition in mcomp.cdg	pelem.cdg

“Y”	Union in mcomp.cdg	pelem.cdg
“W”	Variable in mvar.cdg	pelem.cdg
"K"	Pointer to COMP in mcomp.cdg	pelem.cdg
"L"	Pointer to UNION in mcomp.cdg	pelem.cdg
"M"	Pointer to VAR in mvar.cdg	pelem.cdg
"G"	Pointer (code tranparent) to COMP in mcomp.cdg	pelem.cdg
"H"	Pointer (code tranparent) to UNION in mcomp.cdg	pelem.cdg
"I"	Pointer (code tranparent) to VAR in mcomp.cdg	pelem.cdg
	repetition types	
‘b’	bit field (GSM/GPRS)	m/pelem.cdg
‘c’	conditional repition (GSM/GPRS)	m/pelem.cdg
‘i’	interval (GSM/GPRS)	m/pelem.cdg
‘v’	variable repition (GSM/GPRS)	m/pelem.cdg
'c'	variable repetition of ASN1 BISTRING	melem.cdg
'C'	fix repetition of ASN1 BISTRING	melem.cdg
'j'	variable repetition of ASN1 INTEGER,OCTET,SEQUENCE	melem.cdg
'J'	Pointer to variable repetition of ASN1 BISTRING	melem.cdg

2 Usage of ccdgen

```
  ccdgen - CCD-Generator Version: 3.0.28 (C) Copyright Condat AG, Germany, 1996-2000
      Build at Mon Nov 12 17:30:09 2001
USAGE:
  ccdgen [Options] file1 file2 file3 ...

    Options: -p    - Process primitive-description files (*.pdf)
                     Default: Process message-description files (*.mdf)

             -aX   - Alignment in structures for word and long adressing
                     where X may be 0 - align at byte boundaries (no alignment)
                                    1 - align at word boundaries (default)

                                    2 - align words at word boundaries
                                        and longs at long boundaries
             -bs (or -ms)  - Declares the station type: either mobile or base
                     Default: mobile station (ms)

             -d    - Generate additional sizing information
                     into the C-Headerfiles
```

```
           -Fp  - Defines the path (p) to the input directory
                  there are the Files to process
                  Default: the current directory

           -h   - Generate code for multiple include
                  of the C-Headerfiles

           -Ip  - Defines the path (p) to the include files (*.con,*.sub ..)

           -l   - Generate SDL declarations

           -mX  - allocate X KByte of memory for symbol processing
                  Default: 10 KByte, max: 512KByte

           -Op  - Defines the path (p) to the output directory
                  Default: the current directory

           -rF  - Read imput files from filelist F, F must contain a space separated
                  list of files of the form path\name.[PDF|MDF]
                  -p and -f are ignored for files specified with -rF

           -s   - Generate string table for long names and symbolic values
                  Default: empty string table

           -t   - Generate Symbols in CCD-Tables
                  Default: no symbols

           -veb  - Best fit for enums - refer to your compiler manual !!!

           -vei2 - Treat enums as 16 bit integer

           -vei4 - Treat enums as 32 bit integer (Default)

           -x    - extended structure name definition for CAD-UL debugger

           -zzz  - zzz_alignN for DevStudion 6.0

  Files: mdf or pdf without extension!

NEW WAY to process all mdf and pdf files at once

ccdgen [general options] -Fpath_to_msg MSG1 .. MSGn -Fpath_to_sap -P SAP1 .. SAPn
                                                              >>> -P <<<
```

With the introduction of links from SAP documents to MSG documents it has been necessary to call ccdgen with all *.mdf and all *.pdf at once. A calling of ccdgen in that way lokks like this:

```
ccdgen.exe -s -t -h -l -m512 -a0 -oPathToCdginc -fpath_to_msg MSG1 .. MSGn -fpath_to_sap -P
SAP1 .. SAPn
```

Due to the fact that one have to give a lot of file names on command line, for simplicity one can write a file which contains all MSG and SAP files. The format of this filelist file is:

```
w:\GSM\Condat\ms\DFILE\m_umts_as_asn1_inc.mdf
w:\GSM\Condat\ms\DFILE\7010_150_GSI_INC.pdf
w:\GSM\Condat\ms\DFILE\MSG\CC.MDF
w:\GSM\Condat\ms\DFILE\MSG\SM.MDF
…
w:\GSM\Condat\ms\DFILE\MSG\m_umts_as_asn1_msg.mdf
w:\GSM\Condat\ms\DFILE\PRIM\DL.PDF
…
w:\GSM\Condat\ms\DFILE\PRIM\7010_115_RCM_SAP.PDF
w:\GSM\Condat\ms\DFILE\PRIM\7010_116_CIPH_SAP.PDF
```

In difference to the command line where the file names are given without extension, in the filelist file there is to give the path and the file name with extension.

Calling of ccdgen : `ccdgen -t -h -m512 -c -a0 -s -l -opath_to_cdginc -Rpath\filelist.rsp`

3 Grammar of ccdgen

The grammar of ccdgen consists of two dialects. One is for the message syntax and one is for the primitve syntax. The grammar for the primitive syntax is a subset of the message syntax. The main difference is that the message syntax relates to bit level and the primitive syntax relates to C level types. The grammar notated in EBNF have been written years later after the first running versions of ccdgen. To confirm that the grammar is as close as possible to the real implemetation of the ccdgen parser, this grammar have been tested with ANTLR. ANTLR is a similar tool as YACC and generates parsers to a given grammar notated in EBNF. There are several kinds of EBNF notations and ANTLR uses its own dialect to get it machine readable. Detailed information about ANTLR: http://www.polhode.com/pccts.html

3.1 EBNF Descriptions in ANTLR notation

Name	Form	Example
plain subrule	`(...)`	`(ID | INT)`
zero-or-more	`(...)*`	`ID ( “,” ID )*`
one-or-more	`(...)+`	`( declaration )+`
optional	`{...}`	`{ “else” statement }`

3.2 Grammar of mdf in ANTLR notation

```
#token "[\ \t]+"      <<skip();>>
#token "[\n\r]"       << newline(); skip(); >>
#token "; ~[\n]* \n"  << newline(); skip(); >>
```

```
#token ASN1_INTEGER  "ASN1_INTEGER"
#token ASN1_SEQUENCE "ASN1_SEQUENCE"
#token ASN1_CHOICE   "ASN1_CHOICE"
#token ASN1_OCTET    "ASN1_OCTET"
#token BITSTRING     "BITSTRING"
#token BCDODD        "BCDODD"
#token BCDEVEN       "BCDEVEN"
#token BCD_NOFILL    "BCD_NOFILL"
#token BCD_MNC       "BCD_MNC"
#token CSN1_S1       "CSN1_S1"
#token CSN1_SHL      "CSN1_SHL"
#token GSM1_V        "GSM1_V"
#token GSM1_TV       "GSM1_TV"
#token GSM2_T        "GSM2_T"
#token GSM3_V        "GSM3_V"
#token GSM3_TV       "GSM3_TV"
#token GSM4_LV       "GSM4_LV"
#token GSM4_TLV      "GSM4_TLV"
#token GSM5_V        "GSM5_V"
#token GSM5_TLV      "GSM5_TLV"
#token GSM6_TLV      "GSM6_TLV"
#token GSM7_LV       "GSM7_LV"
#token GSM1_ASN      "GSM1_ASN"
#token S_PADDING     "S_PADDING"
#token T30_IDENT     "T30_IDENT"

#token AND           "AND"
#token AS            "AS"
#token BITAND          "\&"
#token BITOR         "\|"
#token BOTH          "both"
#token CL_REP        "\]"
#token COMP          "COMP"
#token COMPDEF       "COMPATIBILITY_DEFINES"
#token CONST         "CONST"
#token DEFAULT_L     "def"
#token DEFAULT_U     "DEF"
#token DOWNLINK      "downlink"
#token DIV           "/"
#token DUPL          ":"
#token DYN           "DYN"
#token ENUM          "ENUM"
#token EQUAL         "="
#token EXTERN        "EXTERN"
#token GETPOS        "GETPOS"
```

```
#token GREATERTHEN    ">"
#token IFNOTPRESENT "IFNOTPRESENT"
#token LESSTHEN              "<"
#token MINUS        "\-"
#token MSG          "MSG"
#token MULT         "\*"
#token NEQ          "#"
#token NO           "NO"
#token OP_REP       "\["
#token OPTIONAL     "optional"
#token OR           "OR"
#token PLUS         "\+"
#token PTR          "PTR"
#token PRAGMA       "PRAGMA"
#token PREFIX       "PREFIX"
#token RANGE        "RANGE"
#token SETPOS       "SETPOS"
#token TYPE         "TYPE"
#token TYPEDEF      "TYPEDEF"
#token UNION        "UNION"
#token UPLINK       "uplink"
#token VAR          "VAR"
#token VAL          "VAL"
#token XOR                "XOR"
#token YES          "YES"

#token Num        "[0-9]+"
#token HexNum     "0x[a-fA-F0-9]+"
#token BinNum     "0b[01]+"
#token SpareBit   ".0"
#token SpareBits  ".[01]+"
#token BitNum     ".[0-9]"
#token Range      ".."
#token EmptyString "\"\""
#token ID         "[a-zA-Z0-9_]+"
#token TAG_ID     "[A-Z0-9_]+"
#token BitID      ".[a-zA-Z0-9_]+"
#token String     "\" [a-zA-Z0-9_/.:;,='\(\)\-\+\ \t]+ \""
#token Eof           "@"



#lexclass START


class MdfParser
{
  mdf : (codingDef)+ Eof;
```

```
codingDef
  : pragmaDef
  | constantDef
  | externConstantDef
  | codingTypeDef
  | varDef
  | typeDef
  | enumDef
  | compDef
  | unionDef
  | msgDef
  ;

pragmaDef
  : PRAGMA (prefixDef | compatibleDef)
  ;

prefixDef
  : PREFIX ID
  ;

compatibleDef
  : COMPDEF (YES | NO)
  ;

constantDef
  : CONST ID (Num | HexNum)
  ;

externConstantDef
  : EXTERN CONST "\@"ID MINUS ID"\@" ID
  ;

codingTypeDef
  : TYPE codingType  Num {OPTIONAL}
  ;

codingType
  : ASN1_INTEGER
  | ASN1_SEQUENCE
  | ASN1_CHOICE
  | ASN1_OCTET
  | BITSTRING
  | BCDODD
  | BCDEVEN
  | BCD_NOFILL
  | BCD_MNC
  | CSN1_S1
  | CSN1_SHL
  | GSM1_V
  | GSM1_TV
  | GSM2_T
  | GSM3_V
  | GSM3_TV
  | GSM4_LV
  | GSM4_TLV
  | GSM5_V
  | GSM5_TLV
  | GSM6_TLV
  | GSM7_LV
  | GSM1_ASN
  | S_PADDING
  | T30_IDENT
  ;

varDef
  : VAR ID {AS ID} String
    Num  {HexNum}
    {RANGE {MINUS|PLUS} Num Range {MINUS|PLUS} Num}
    {IFNOTPRESENT (Num|ID)}
    (VAL numDefault {ID} {String | EmptyString})*
  ;

numDefault
  : Num {MINUS Num}
  | DEFAULT_L
  | DEFAULT_U
  ;

typeDef
  : TYPEDEF ID ID String
  ;

enumDef
  : ENUM ID String
    (VAL (Num | HexNum) ID {String})*
  ;

unionDef
  : UNION ID String elemDef
  ;
  compDef
    : COMP ID String {HexNum} elemDef
    ;

  msgDef
    : MSG ID (DOWNLINK  | UPLINK | BOTH) (HexNum | BinNum) elemDef
    ;

  elemDef
    : "\{" (optionalDef | mandatoryDef)* "\}"
    ;

  optionalDef
    : "<" {prologDef}(spareDef | (elemDefVCU ID {AS ID} {DYN | PTR} {arrayDef})) {epilogDef}
">"
    ;

  mandatoryDef
    : {bitGroupDef} (spareDef | (elemDefVCU ID {AS ID} {DYN | PTR} {arrayDef})) {epilogDef}
    ;

  bitGroupDef
    : PLUS
    | MINUS
    | MULT
    ;

  spareDef
    : SpareBit
    | SpareBits
    ;

  elemDefVCU
    : TAG_ID EQUAL Num {codingType} {EXTERN (VARTYPE | COMP ) "\@"ID MINUS ID"\@"}
    | {codingType} {EXTERN (VARTYPE | COMP | UNION) "\@"ID MINUS ID"\@"}
    ;

  prologDef
        : "\(" ( ID (arithmOpDef ID)* compareOpDef (Num | ID) (logicOpDef ID (arithmOpDef ID)*
compareOpDef (Num | ID))* )* "\)"
    ;

  arithmOpDef
        : BITAND
        | BITOR
```

```
       | PLUS
       | MINUS
       | MULT
       | DIV
       ;

  compareOpDef
       : EQUAL
       | NEQ
       | GREATERTHEN
       | LESSTHEN
       ;

  logicOpDef
       : AND
       | OR
       | XOR
       ;

  epilogDef
       : "\(" ((GETPOS | SETPOS | Num | PLUS | DUPL)",")* "\)"
    ;

  arrayDef
    : OP_REP (bitArrayDef | nonBitArrayDef)  {variableArrayDef} CL_REP
    ;

  bitArrayDef
    : ((BitID {PLUS Num}) | BitNum)
    ;

  nonBitArrayDef
    : (ID {PLUS Num}) | Num
    ;

  variableArrayDef
   : Range (ID {PLUS Num} |Num)
   ;
}
```

3.3 Grammar of pdf in ANTLR notation

```
#token "[\ \t]+"      <<skip();>>
#token "[\n\r]"       << newline(); skip(); >>
#token "; ~[\n]* \n"  << newline(); skip(); >>

#token AS            "AS"
#token COMP          "COMP"
#token COMPDEF       "COMPATIBILITY_DEFINES"
#token DYN           "DYN"
#token CONST         "CONST"
#token ENUM          "ENUM"
#token EXTERN        "EXTERN"
#token IFNOTPRESENT  "IFNOTPRESENT"
#token MINUS         "\-"
#token NO            "NO"
#token OPTIONAL      "optional"
#token PLUS          "\+"
#token PTR           "PTR"
#token PRAGMA        "PRAGMA"
#token PREFIX        "PREFIX"
#token PRIM          "PRIM"
#token RANGE         "RANGE"
#token Range         ".."
#token TYPE          "TYPE"
#token TYPEDEF       "TYPEDEF"
#token UNION         "UNION"
#token VAR           "VAR"
#token VAL           "VAL"
#token YES           "YES"

#token VARTYPE  "(B|C|L|M|S|T)"
#token Num      "[0-9]*"
#token HexNum   "0x[a-fA-F0-9]*"
#token STRING   "\" [a-zA-Z0-9_/:.=,'#\(\)\-\ \t]+ \""
#token ID       "[a-zA-Z0-9_]+"
#token TAG_ID   "[a-zA-Z0-9_]+"
#token Eof         "@"



#lexclass START


class PdfParser
{
  pdf : (codingDef)+ Eof;


  codingDef
    : pragmaDef
    | constantDef
    | externConstantDef
    | varDef
    | typeDef
    | enumDef
    | compDef
    | unionDef
    | primDef
    ;

  pragmaDef
    : PRAGMA (prefixDef | compatibleDef)
    ;

  prefixDef
    : PREFIX ID
    ;

  compatibleDef
    : COMPDEF (YES | NO)
    ;

  constantDef
    : CONST ID (Num | HexNum)
    ;

  externConstantDef
    : EXTERN CONST "\@"ID MINUS ID"\@" ID
    ;

  varDef
    : VAR ID {AS ID} STRING VARTYPE
      (VAL (Num | HexNum) {MINUS (Num | HexNum)} {ID} {STRING})*
    ;

  typeDef
    : TYPEDEF ID ID STRING VARTYPE
    ;

  enumDef
    : ENUM ID STRING
      (VAL (Num | HexNum) ID {STRING})*
    ;

  unionDef
    : UNION ID STRING
```

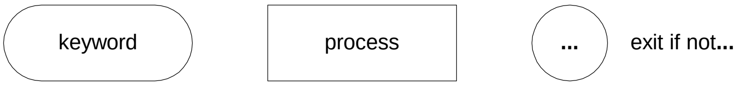
```
      "\{" elemDefU "\}"
    ;

  elemDefU
    : (  {"<"}{"\("}{"\)"} TAG_ID ID {AS ID} {arrayDef} {">"} )+
    ;

  compDef
    : COMP ID STRING
      "\{" elemDef "\}"
    ;

  primDef
    : PRIM ID HexNum
      "\{" elemDef "\}"
    ;

  elemDef
    : ( {"<"}{"\("}{"\)"} {EXTERN (VARTYPE | COMP | UNION) "\@"ID MINUS ID"\@"} ID {AS ID}
{DYN | PTR} {arrayDef} {">"} )*
    ;

  arrayDef
    : "\[" (ID|Num) {Range (ID|Num)} "\]"
    ;
}
```

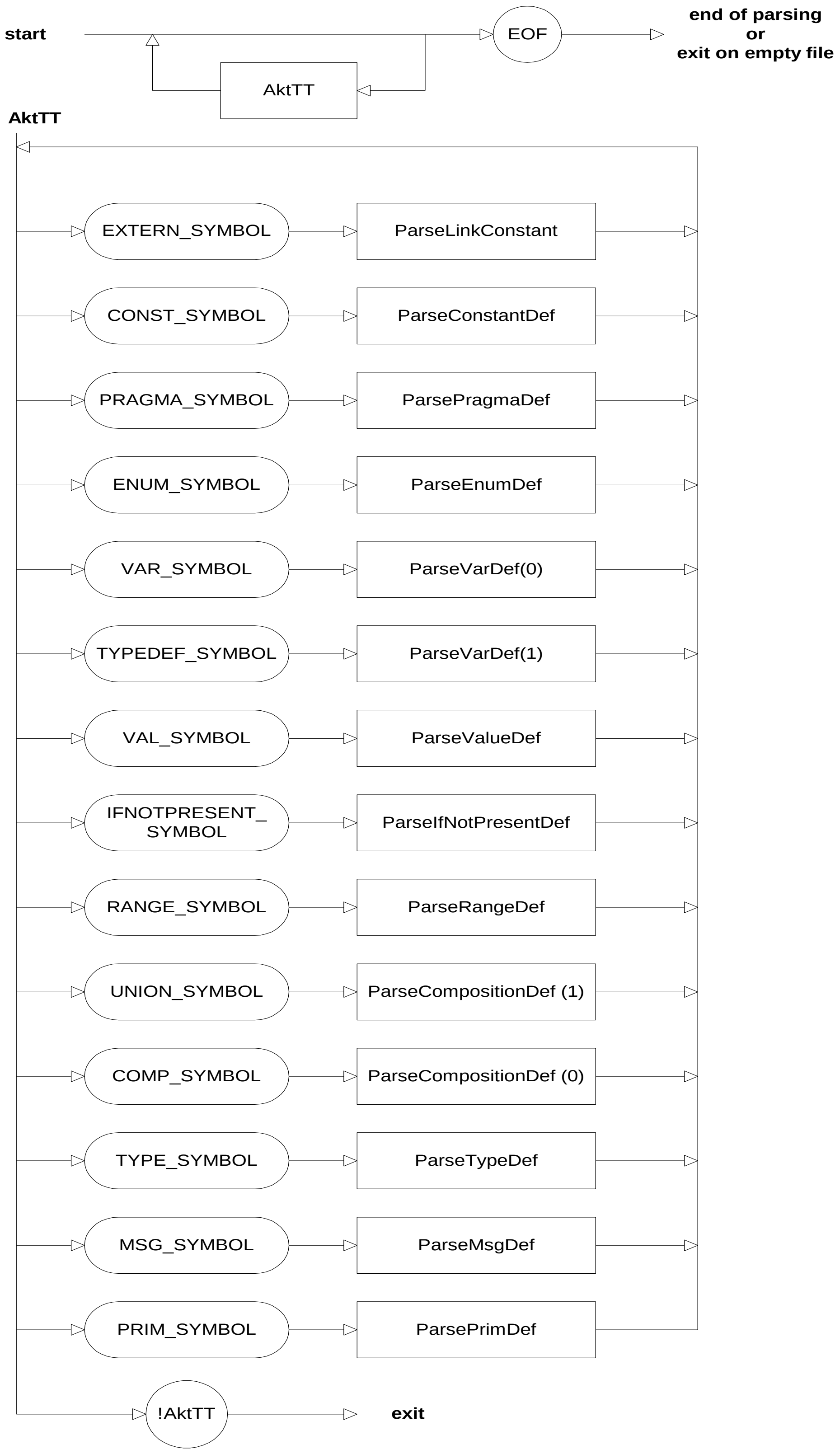
4 Syntax Diagram of the grammar

The EBNF grammar of mdf and pdf can be dark and scary, therefore the following syntax diagrams can be helpful. The syntax diagrams are not program flow charts. There are no details about the internals of ccdgen, but only shows how the parser of ccdgen acts with the token/keywords.

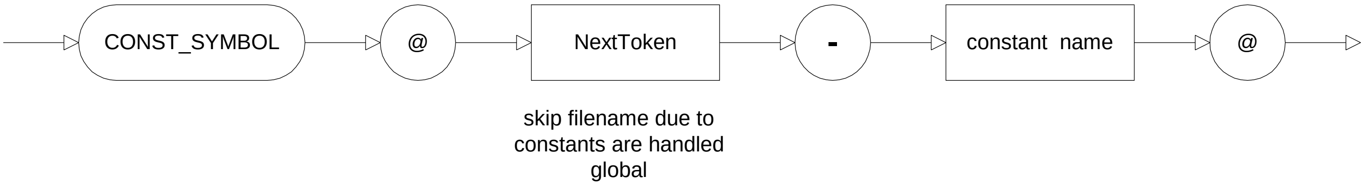
4.1 Explanation of the used symbols in the syntax diagram

The syntax diagrams consists of three symbols only. In very rare cases there is in addition the switch symbol (rhombus) to clear certain parser behaviour. The ellipse is used for the keywords recognized by ccdgen. The square is used for a function call or a pseudo name for a piece of code. The circle is used for an exit point when a certain condition is not kept. (Due to scaling from visio sheets to word the circle can become an egg shape)

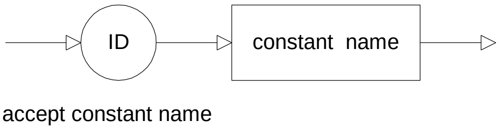



start
EOF
end of parsing
or
exit on empty file
AktTT
AktTT
EXTERN_SYMBOL
ParseLinkConstant
CONST_SYMBOL
ParseConstantDef
PRAGMA_SYMBOL
ParsePragmaDef
ENUM_SYMBOL
ParseEnumDef
VAR_SYMBOL
ParseVarDef(0)
TYPEDEF_SYMBOL
ParseVarDef(1)
VAL_SYMBOL
ParseValueDef
IFNOTPRESENT_
SYMBOL
ParseIfNotPresentDef
RANGE_SYMBOL
ParseRangeDef
UNION_SYMBOL
ParseCompositionDef (1)
COMP_SYMBOL
ParseCompositionDef (0)
TYPE_SYMBOL
ParseTypeDef
MSG_SYMBOL
ParseMsgDef
PRIM_SYMBOL
ParsePrimDef
!AktTT
exit

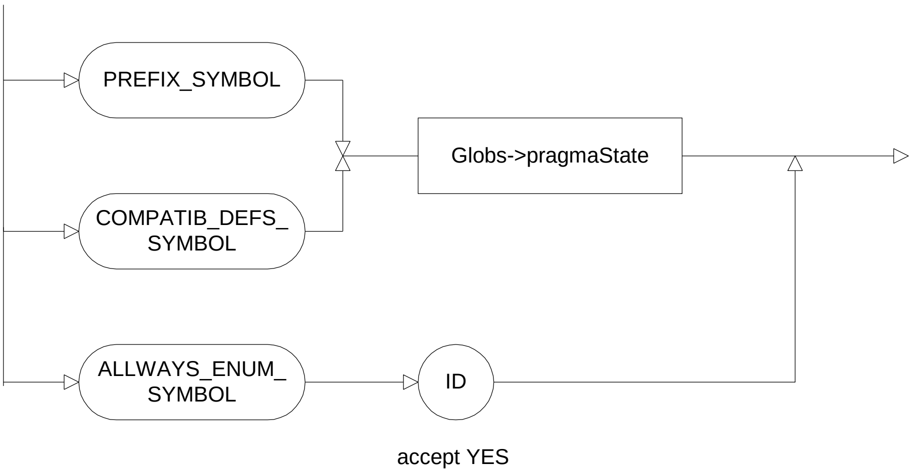
ParseLinkConstant



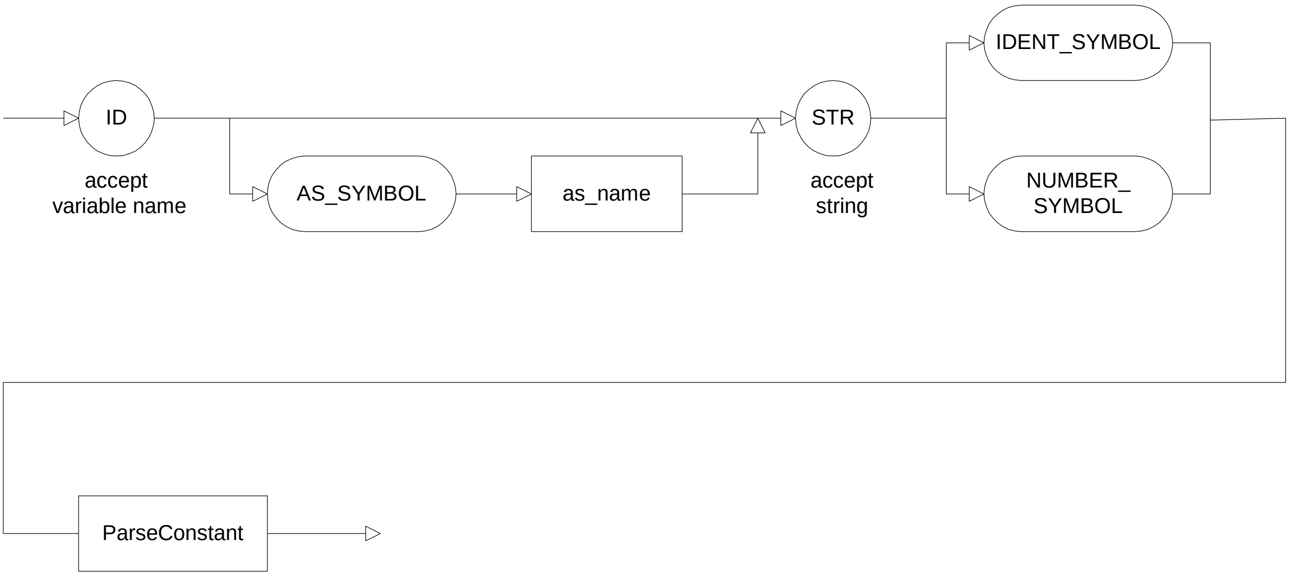
ParseConstantDef



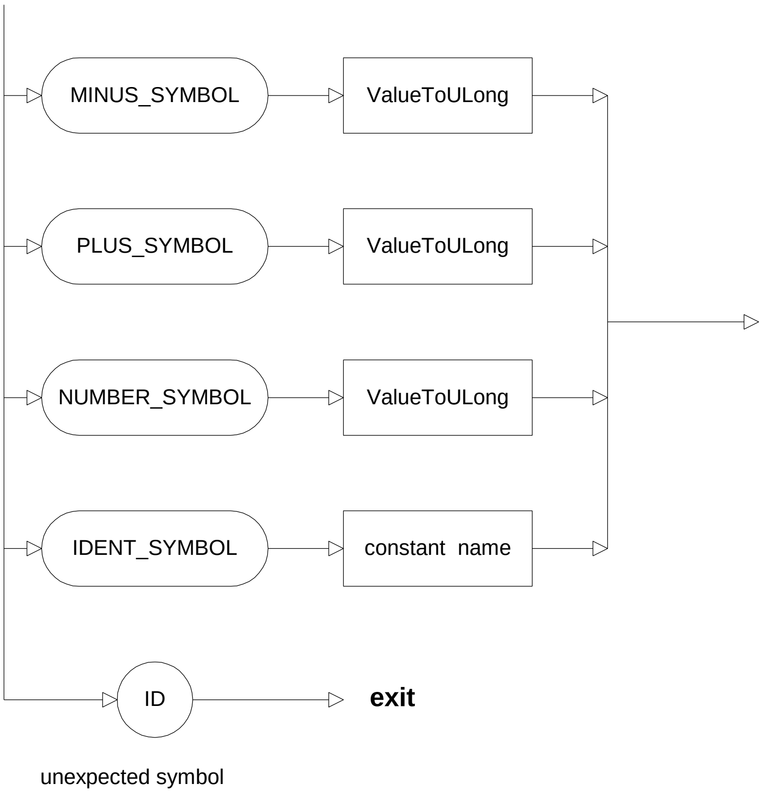
ParsePragmaDef



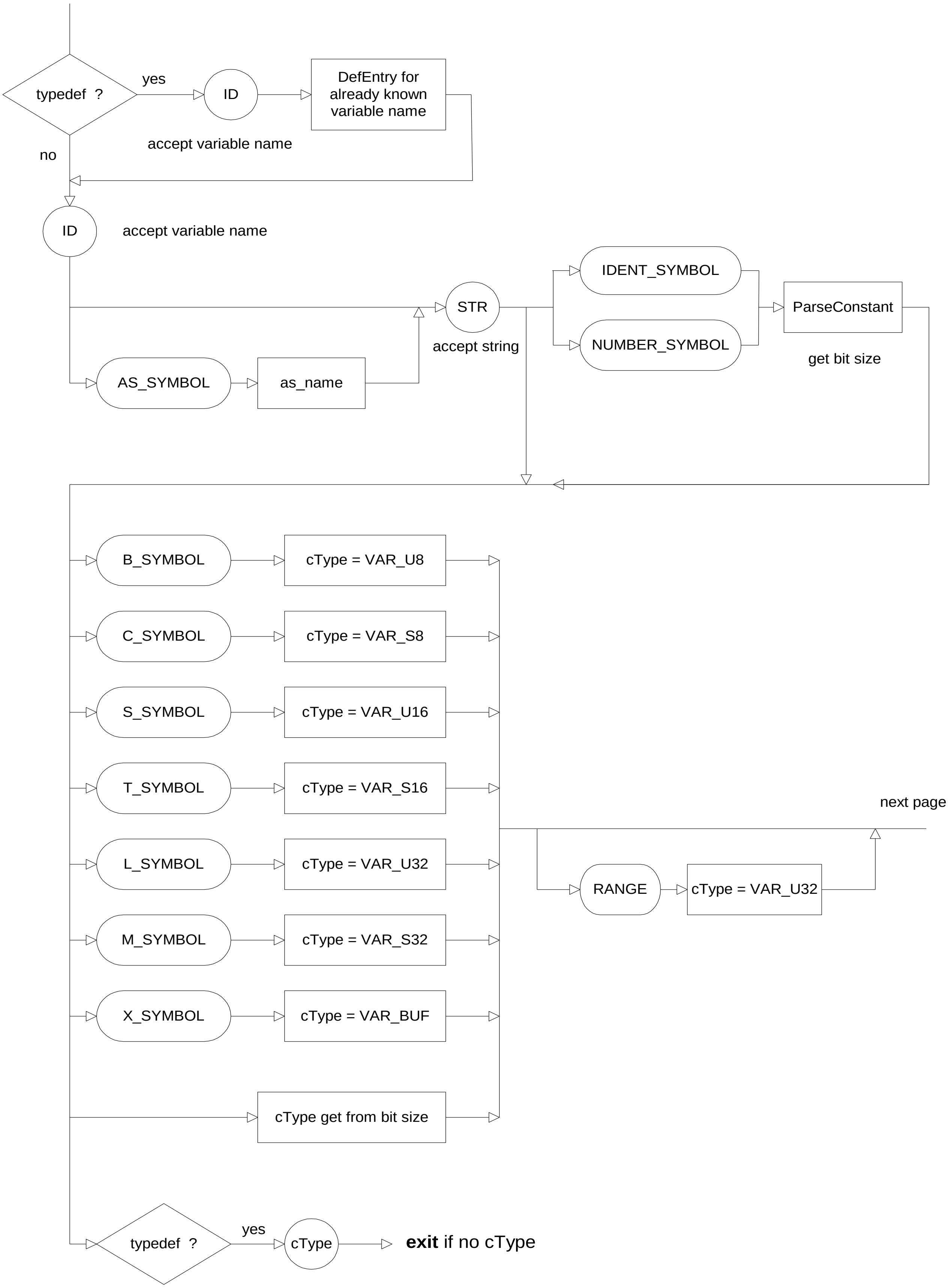
ParseEnumDef



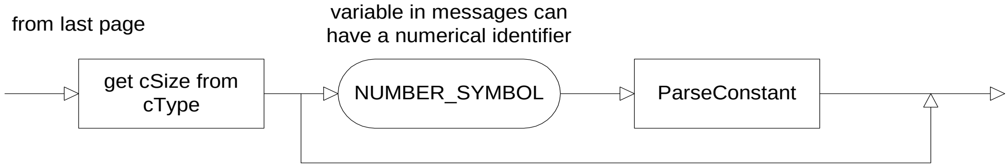
ParseConstant



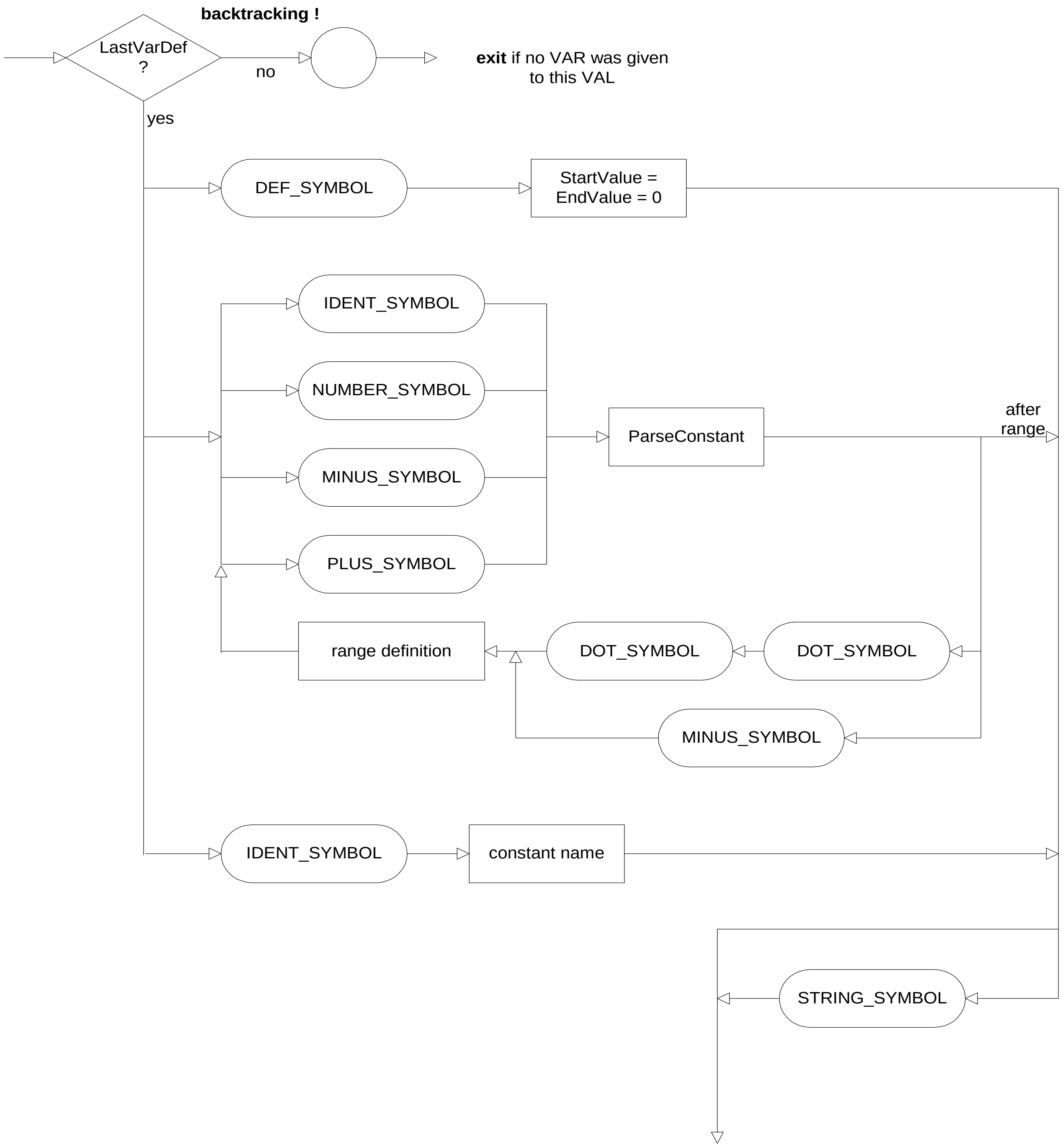
ParseVarDef



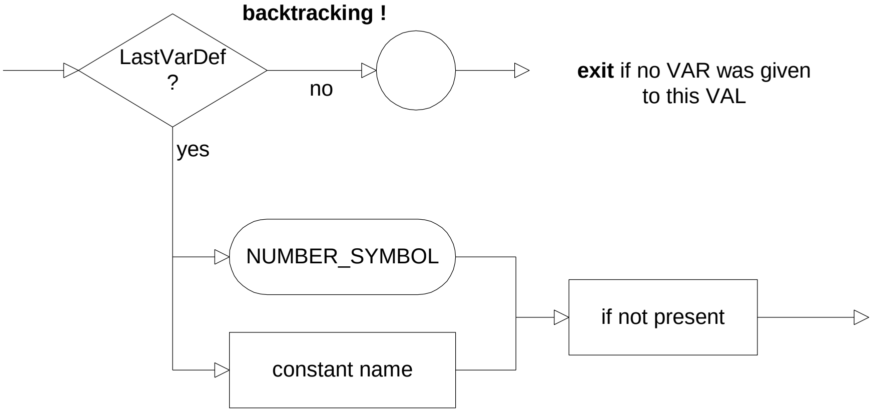
ParseVarDef



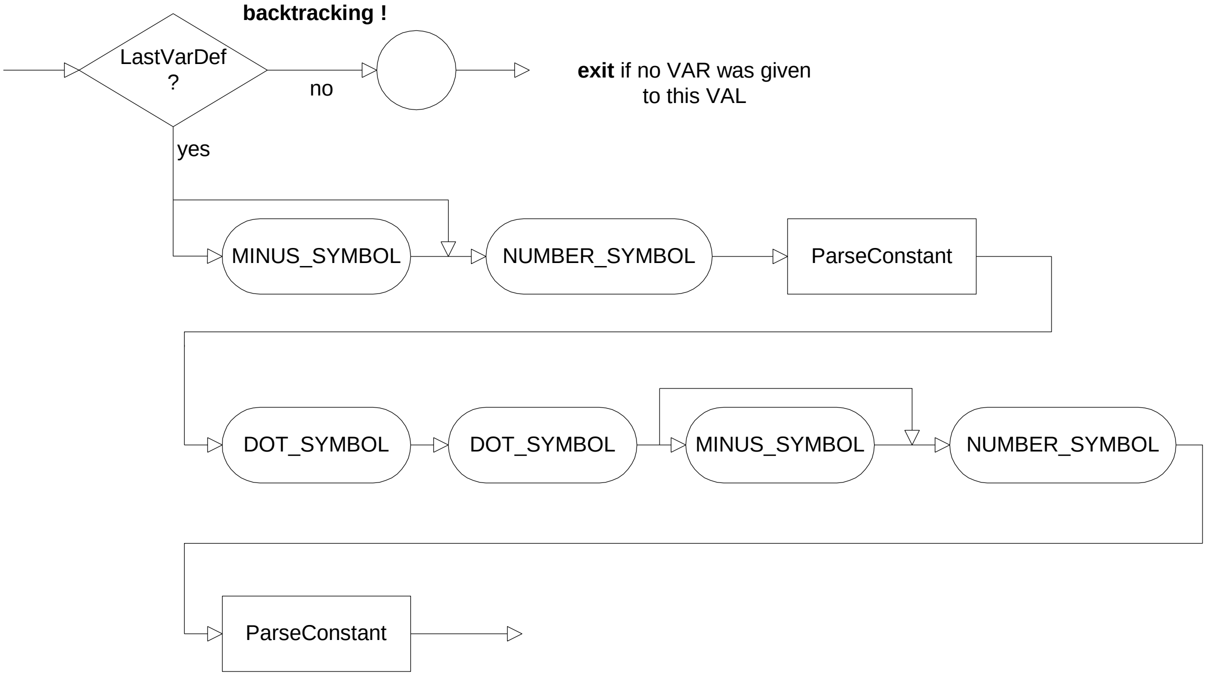
ParseValueDef



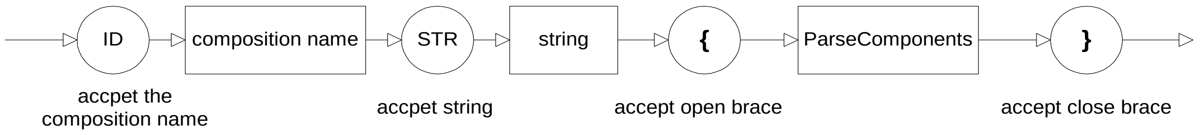
ParseIfNotPresentDef



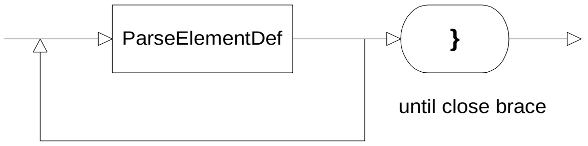
ParseRangeDef



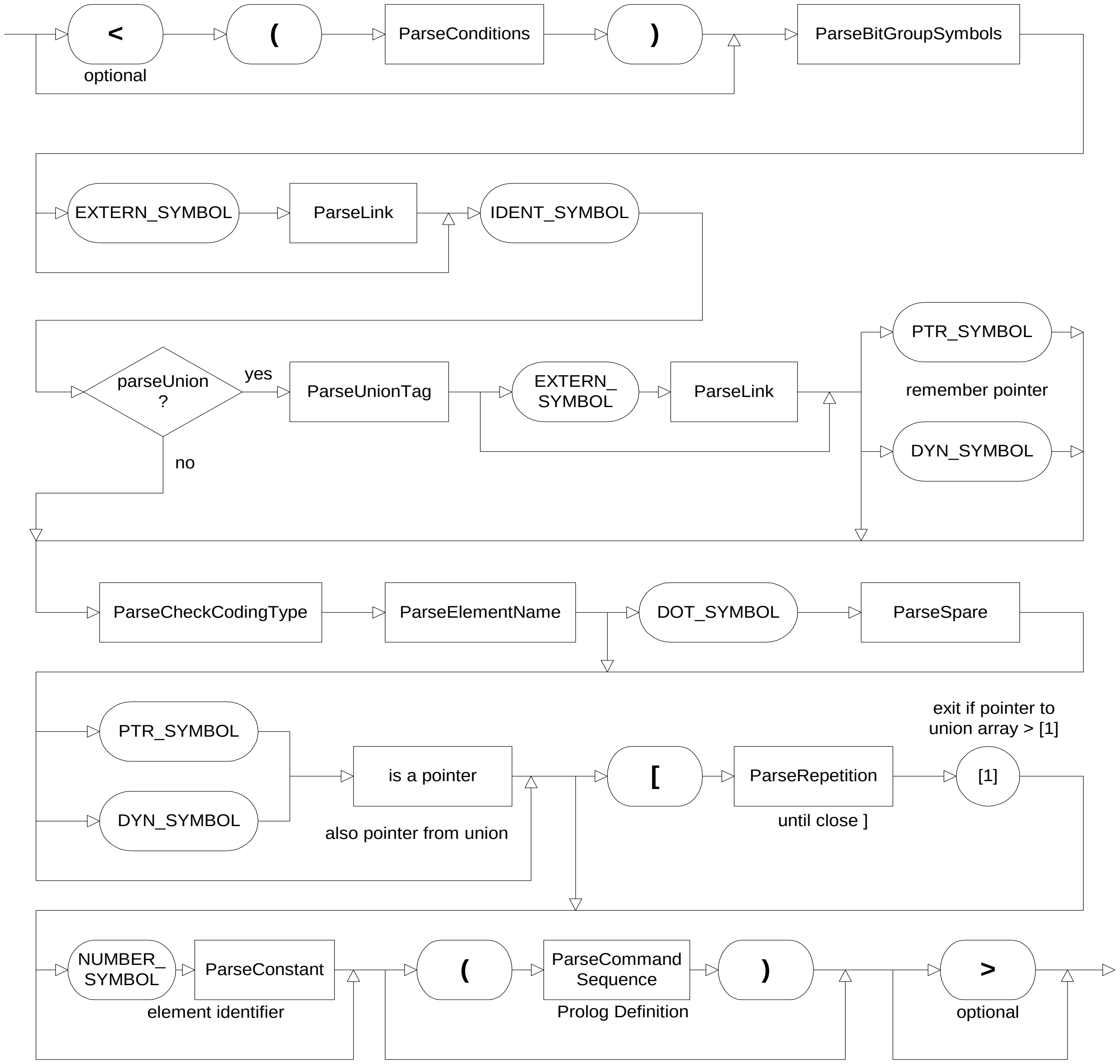
ParseCompositionDef



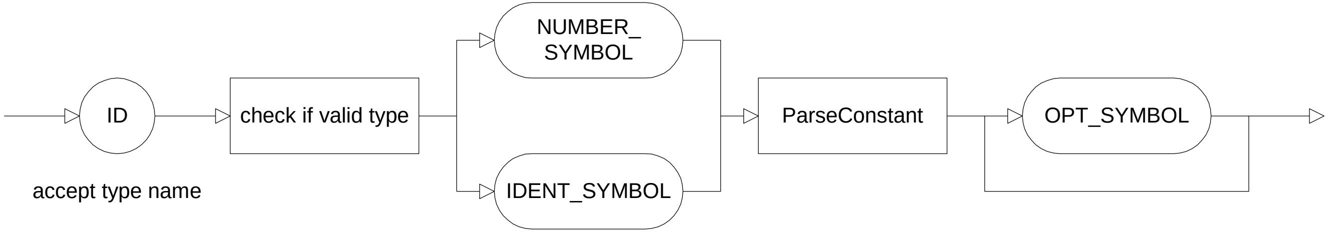
ParseComponents



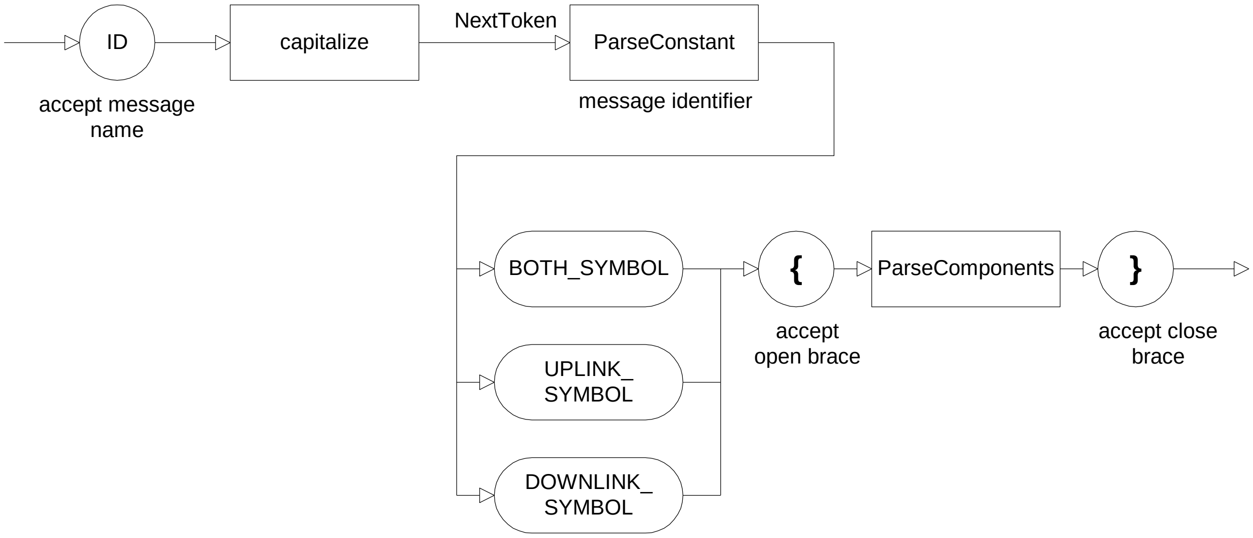
ParseElementDef



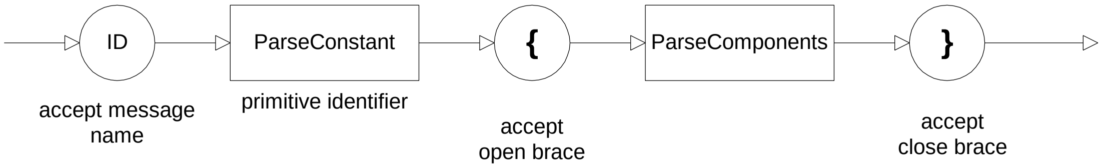
ParseTypeDef



ParseMsgDef

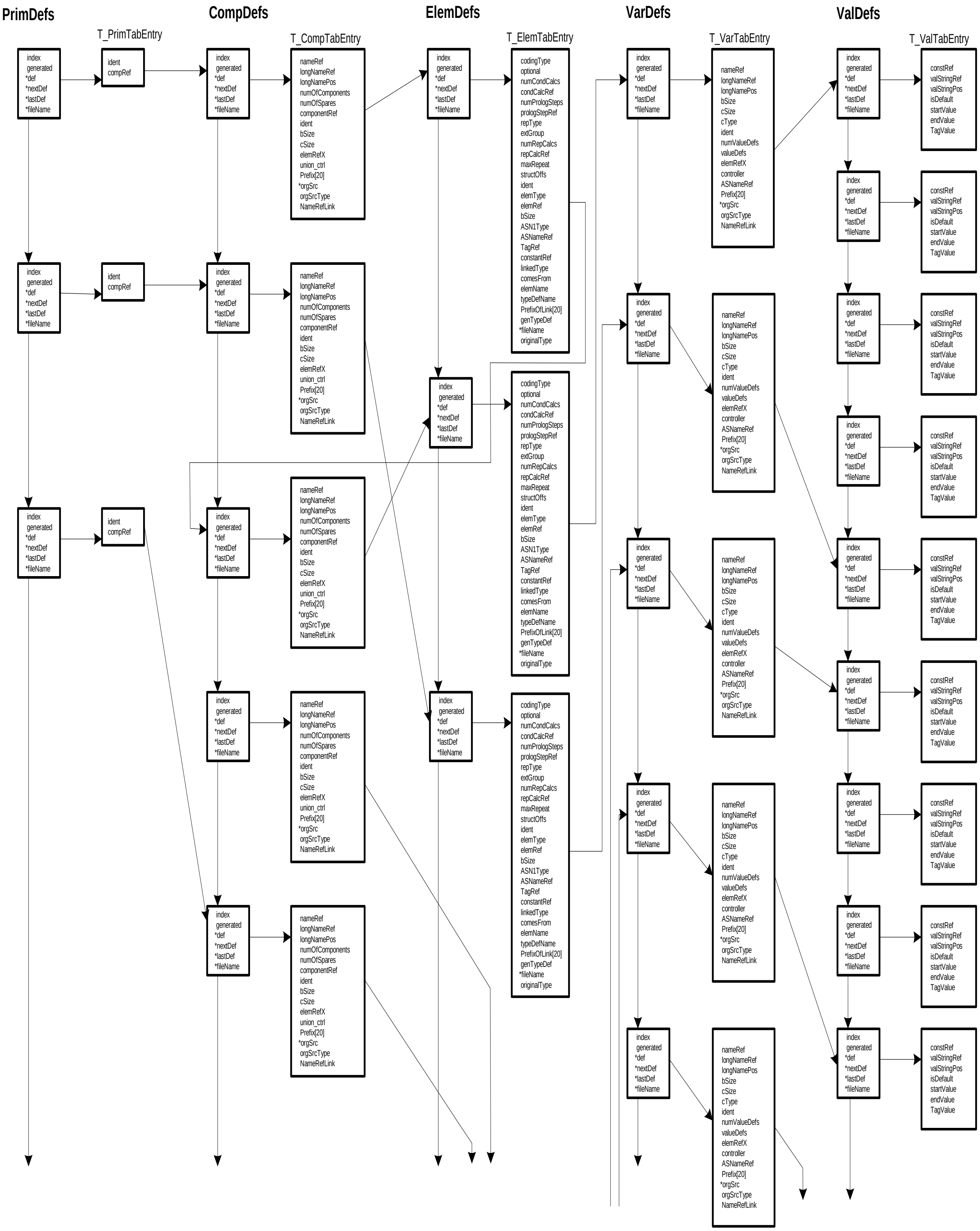


ParsePrimDef



5 The symbol tables in ccdgen

This chapter describes the symbol tables build at runtime during parsing by ccdgen. The coder part of ccdgen then uses this symbol tables to generate the header files, the cdg tables and the sdl files. Here an pseudo example for primitives:



As shown in the example above, the symbol table of ccdgen consists of several linked lists where each list is implemented as a generic container list. The structure of that container is defined in the following way:

```
typedef struct S_DefEntry
{
  USHORT             index;       /* index of table entry */
  BOOL               generated;
  void              *def;         /* contains a reference to a T_XXXTabEntry
*/
  struct S_DefEntry *nextDef;
  struct S_DefEntry *lastDef;
  char              *fileName;
  BOOL                fileType;
} T_DefEntry;
```

There are the following container lists to maintain the symbols. The void pointer *def is used to point to the actual maintained data structure

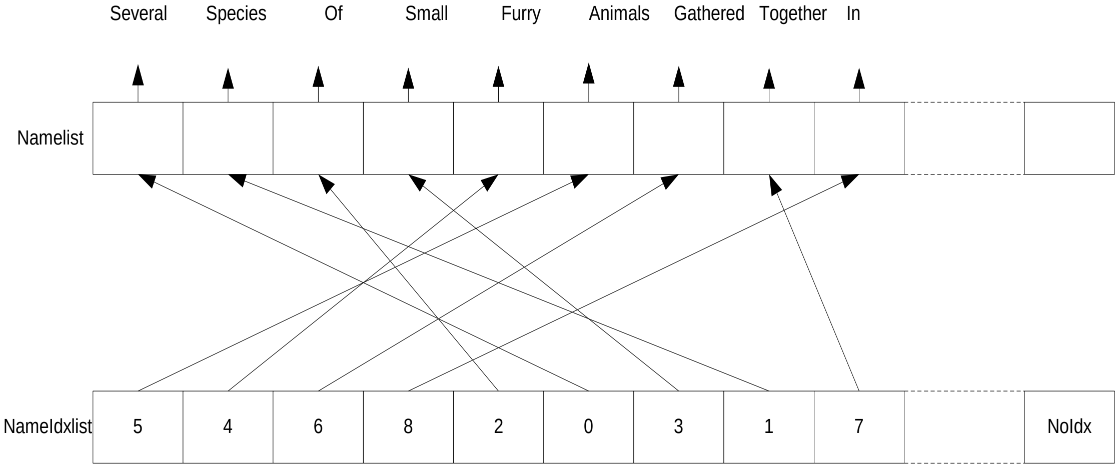
```
T_DefEntry    *ConstDefs;     → T_ConstTabEntry
T_DefEntry    *VarDefs;       → T_VarTabEntry
T_DefEntry    *ValDefs;       → T_ValTabEntry
T_DefEntry    *SpareDefs;     → T_SpareTabEntry
T_DefEntry    *ElemDefs;      → T_ElemTabEntry
T_DefEntry    *TypeDefs;      → T_TypeTabEntry
T_DefEntry    *CompDefs;      → T_CompTabEntry
T_DefEntry    *CalcDefs;      → T_CalcTabEntry
T_DefEntry    *MsgDefs;       → T_MsgTabEntry
T_DefEntry    *PrimDefs;      → T_PrimTabEntry
```

The definition of the T_*TabEntry structures can be found in deflst.h

In addition to the definition lists there are two further data structures. One is used to maintain all names and one is used to maintain strings.

5.1 Namelist

The namelist data struture consists of two large static arrays called Namelist and NameIdxList. Both have the size of 0xFFFF, where NameIdxList is just an array of 0xFFFF USHORTs and Namelist is an array of 0xFFFF T_NameEntry structures (see namelist.c). Namelist is filled with datas at runtime just as they appear. So there is no alphabetical order. To get an alpabetical order for quick searching of a name NameIdxList is used for. With every entry in Namelist the entries in NameIdxList are always reordered in a way, so that the first element in NameIdxList is an index to an element in Namelist which represents the most alphabetically ordered name.



When a name is searched the start point for the search begins always in the middle of all indices in NameIdxList. In our example from NameIdxList[4] where we have the index 2 to Namelist. Assume we are looking for "Gathered". The string compare bewteen NameList[NameIdxList[4]] = NameList[2] = "Of" and "Gathered" returns >0. That leads to a new iteration search loop, where the new starting point in NameIdxList is calculated by NameIdxList[4] - NameIdxList[0] = NameIdxList[2]. NameIdxList[2] has an index to NameList[6] = "Gathered". The string compare between NameList[NameIdxList[2]] = NameList[6] = "Gathered" and "Gathered" is successful. The search algorithm is called binary search with succesive approximation.

The structure T_NameEntry which is used for each element in Namelist (see namelist.c) maintains besides the names by a pointer to that name also a NameNode. The structure NameNode keeps a NameClass to the name to distinguish for what data type the name is in use. In addition to a name there is the information from which inputfile the name comes from. The UMTS project has a new link approach where through links form a document to another the type names can be renamed but the element name are identical. In such cases there is a linked list build up from NameNode to maintain that special information.

Currently (Nov.2001) there are about 6000 names when processed UMTS,GPRS and GSM stuff by ccdgen.

5.2 Stringlist

For variables and substructures there are mandatory "long names" which are strings. These strings are maintained in large allocated memory controlled by the ccdgen command line key –mxxx. In case of UMTS we need –m512 which allocates 512 kbytes of memory called StringList. Each string is consecutively stored in StringList and the start address is stored as longNamePos in the related T_CompTabEntry, T_VarTabEntry structure respectively.

e.g.:

longName = STL_GetString (StringList, VTE->longNamePos); /* variable */

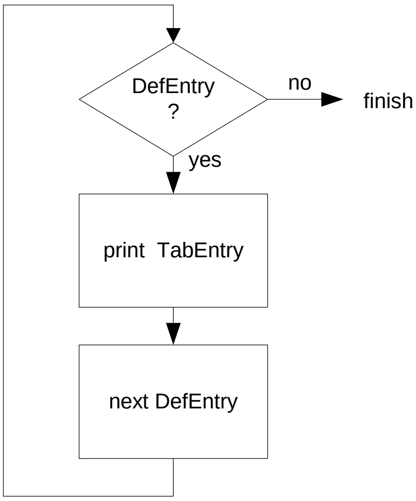
longName = STL_GetString (StringList, CTE->longNamePos); /* substructure */

These long names appear as comment in the *.h files and TAP uses pstr.cdg and mstr.cdg to print that strings on screen during testing of the protocoll stack.

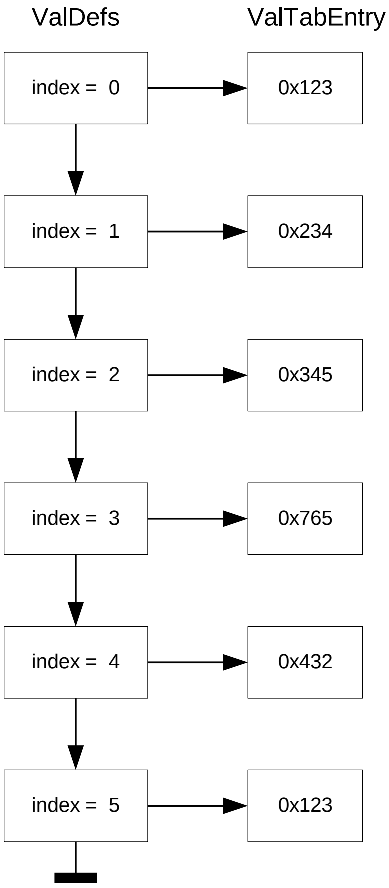
When ccdgen has build up the symbol tables during the parser phase there is a "spider net" of pointers and references among the symbol tables and to the Namelist and StringList.

6 The generation of header files, cdg tables and sdl files

The generation of the cdg tables are relatively straight forward iteration loops where you start at the first entry of a container list, e.g.: ValDefs.



Then print out the necessary information to the appropriate cdg table. If ccdgen was called with all mdf and pdf files at once, then ccdgen remebers the DefEntry.index for all mdf related cdg tables. These indices have to be subtracted when generating the pdf tables so that the pdf related cdg tables starts with index 0.



```
                  mval.cdg
/*  idx valStrRef isDef startVal  endVal    */
/*    0*/ {    11, 0, 0x00000123, 0x00000123},
/*    1*/ {    12, 0, 0x00000234, 0x00000234},
/*    2*/ {    14, 0, 0x00000345, 0x00000345},
```

assume the first 3 entries are from mdf and the last 3 entries are from pdf:
idx(pval) = index(ValDefs) - max. idx(mval)+1

```
                  pval.cdg
/*  idx valStrRef isDef startVal  endVal    */
/*    0*/ {    15, 0, 0x00000765, 0x00000765},
/*    1*/ {    16, 0, 0x00000432, 0x00000432},
/*    2*/ {    19, 0, 0x00000123, 0x00000123},
```

The generation of the header files is a bit more complex and the following function trace shows simplified which functions are called during the coding phase.

```
>COD_Code
```

```
|   >WriteHeader
|   |   >WriteBuffer
|   |   <WriteBuffer
|   |   >WriteSubStructs
|   |   |   >WriteStruct
|   |   |   |   >WriteDeclaration
|   |   |   |   |   >WriteDeclarationV
|   |   |   |   |   |   >WriteDeclrValues
|   |   |   |   |   |   <WriteDeclrValues
|   |   |   |   |   <WriteDeclarationV
|   |   |   |   <WriteDeclaration
|   |   |   <WriteStruct
|   |   <WriteSubStructs
|   |   >WritePrimitives
|   |   |   >WriteStruct
|   |   |   |   >WriteDeclaration
|   |   |   |   |   >WriteDeclarationCU
|   |   |   |   |   |   >WriteDeclrStruct
|   |   |   |   |   |   <WriteDeclrStruct
|   |   |   |   |   <WriteDeclarationCU
|   |   |   |   <WriteDeclaration
|   |   |   <WriteStruct
|   |   <WritePrimitives
|   <WriteHeader
<COD_Code
```

```
typedef struct
{
  U8   var_in_substru;  /* Variable in Substructure */
} T_substru;
```

```
typedef struct
{
  T_substru   substru;  /* Substructure with a variable
*/
} T_ANY_REQ;
```

The common way in ccdgen to generate structures is to generate a type name for that structure with the typedef statement, as well. The name for the type is derived from the given name in *.pdf or *.mdf. For the example above the *.pdf looks like:

```
                                                              comments
VAR var_in_substru   "Variable in Substructure"  B                 B = unsigned char = U8
VAL  1  ONE  "Variable in Substructure comes with a value"           values go to *.val
COMP substru  "Substructure with a variable"
{
  var_in_substru                                                  composition elements are
}                                                             qualified by names only !
PRIM  ANY_REQ   0x1500                                                That is a huge pile of bullshit !!
{                                                             In addition all variables are
  substru                                                       defined globally, which is the most
}                                                             huge pile of bullshit !!!
```

In case of links from SAP documents to MSG documents the type name can be renamed ! example:

org.mdf	*fir.pdf*	*sec.pdf*
COMP original "..."	COMP first "..."	COMP second "..."

```
{                      {                                  {
 ...                     EXTERN COMP @org-original@ rename    EXTERN  COMP @fir-rename@ newname
}                      }                                  }


M_org.h                  P_fir.h                          P_sec.h
                       #include "M_org.h"                 #include "M_fir.h"
typedef struct           typedef struct                    typedef struct
{                       {                                 {
 ...                      T_original  rename;               T_rename  newname;
} T_original;            } T_first;                       } T_second;
                       typedef T_original  T_rename;
```

In this example we have a nested link where fir.pdf wants to use the type `original` from org.mdf but as element name there is used `rename`. The link from sec.pdf to first.pdf wants to use a type `rename` but as element name `newname`. Because the type `rename` is nowhere explicitely defined, first.pdf must typedefing `T_original` to `T_rename`. Note, that the header file P_fir.h includes M_org.h to get the definition of `T_original` and the header file P_sec.h includes P_fir.h to get the definition of `T_rename`.

Further inforamtion about links in chapter 8 The include\link approach.

7 The ctrl column in documents

instruction in column ctrl	**meaning of instruction**	**example**
[0..CONSTANT]	array of bytes (also USHORT in *.pdf)	[0..MAX_RFL_NUM_LIST]
[varname+number..CONSTANT]	array of bytes (also USHORT in *.pdf)	[rfl_cont_len+3..19]
[.CONSTANT]	array , dot marks a bitarray	(SETPOS){ident_type = ID_TYPE_TMSI} [.32]
BCDODD[numbers]	BCD numbers starting with digit1	
BCDEVEN[numbers]	BCD numbers starting with digit2	BCDEVEN[2] or BCDEVEN[0..20]
{ ... }	conditional	{flag=1 AND flag2=1 OR flag=0}
(...)	command sequence	(GETPOS,:,4,+,:,1,+,SETPOS)
GETPOS	get the bitstream pointer	(GETPOS,:,4,+,:,1,+,SETPOS)
SETPOS	set bit stream pointer	(SETPOS) {type_of_identity # ID_TYPE_NO_IDENT AND type_of_identity # ID_TYPE_TMSI} BCDODD [0..16]
KEEP,regNr	keep value of a variable in ccd register	(KEEP,1) see GRR.doc chapter 5.65
TAKE,regNr	Take the value of ccd register	[.(TAKE,1)+1..8] see GRR.doc chapter 5.136
MAX,regNr	Compare and keep the maximum in ccd register from a variable and ccd register	(MAX,2) see GRR.doc chapter 5.73 and 5.74
:	duplicate the element	(GETPOS,:,4,+,:,1,+,SETPOS)

^	swap the two elements	see CC.doc chapter 5.4 bearer capability
+ * -	first, middle, last octett	see CC.doc chapter 5.4 bearer capability

8 The include\link approach

This chapter describes how to set a link in a document to use definitions of constants, variables or structures that are decribed in another document.

The general syntax to set a link is:

a) absolutePath\FileName.extension – name_of_the_object

b) relativePath\ FileName.extension – name_of_the_object

If the documents are in the same directory the path information can be missed.

8.1 How to set a link

There are two ways to set a link from one file to another. You can use a combination of a textmark and hyperlink provided by WinWord or a hand written link.

8.1.1 Hyperlinks provided by WinWord

WinWord is able to set hyperlinks from one document to another and then to navigate through the documents by pressing the mouse button on the hyperlinks. In chapter 8.2 is a given example and the green coloured word is a textmark(bookmark) and the blue colored stuff is a hyperlink.

The textmark is set by *Ctrl-Shift-F5* and by pressing *Ctrl-K* in the other document there is a popup menu which guides you to choose the Path\File selection and then to choose a textmark that must be set in the file where the link should point to.

8.1.2 Link written by hand

You always can write a link by hand in the link column according to the syntax given above. But then WinWord is not able to navigate through the documents. For xGen100 and ccdgen it does no matter how the links were set.

8.2 Link example from a SAP document to another SAP document

Assume that in SAP document FileA is to describe a structure from it it is known that the structue in FileB has the same content. The link tells ccdgen to use the type for b_struct that was defined in FileA.

FilA

4.7
Description: any description

Definition:

type	short name	commment
STRUCT	a_strcut	any comment

Elements:

long name	short name	ctrl	type
A variable	a_var		UBYTE

History:
31-Dec-00 NNN Initial

FileB

4.23
Description: any description

Definition:

type	short name	commment	link
STRUCT	b_strcut	any comment	FileA – a_struct

History:
29-Feb-01 NNN Initial

8.3 Link example from a SAP document to a *.mdf file

It is also possible to set a link to a mdf file. In case of UMTS there are no MSG documents due to the ASN2MDF compiler already generated the mdf file with a given ETSI specification of the messages described in ASN1.

If you want you can open the *.mdf file in WinWord an even there you can set a textmark. (the green coloured stuff is just for illustration and remains in black, when this is the default colour in WinWord).

ASN1.mdf

```
COMP gsm_classmark_2 "GSM-Classmark2 (array) 5" ; level=1, size=6, dir=UL
{
  ASN1_INTEGER octet
}
```

FileB

4.23
Description: any description

Definition:

type	short name	commment	link
STRUCT	b_strcut	any comment	..\Path\ASN1.mdf – gsm_classmark_2

History:
29-Feb-01 NNN Initial

8.4 What is checked by xGen100 for link information

When xGen100 reads the *.txt files which were converted from *.doc to *.txt by a Winword macro and found a link, then it open this file and just check if the given type and the given name_of_the_object are the same in the linked file. Form the example above it checks whether gsm_classmark_2 and COMP can be found in ASN1.mdf (in this special case xGen100 treats STRUCT and COMP as the same type information).

XGen100 will not check if the content of gsm_classmark_2 is right ! If there are any conflicts the C compiler will give you an error message.

8.5 What is generated by ccdgen for link information

In the generated P_FILEB.h you will find for b_struct, which itself is a member of at least a primitive the following.

```
typedef struct
{
   T_gsm_classmark_2  b_struct; /*  any comment (type from ASN1.mdf)
*/

   ...

}T_ANY_PRIM;
```

9 Unions and their memory layout

This chapter decribes how to use unions inWord documents and the different memory layouts for unions depending of the used command line options of ccdgen.

9.1 How to use unions in Word documents

There is the keyword UNION for the type and in the element table there is a column tag id.

Description:

The description for which the union is in use.

Definition:

Type	short name	Comment
UNION	any_short_name	Any Comment For This

Elements:

tag id	long name	short name	Ref	Type
ANY_ID1	Any comment 1	any_short1	x.y	UBYTE | USHORT | ULONG | STRUCT
ANY_ID2	Any comment 2	any_short2	x.y	UBYTE | USHORT | ULONG | STRUCT
ANY_ID3	Any comment 3	any_short3	x.y	UBYTE | USHORT | ULONG | STRUCT
ANY_IDN	Any comment N	any_shortN	x.y	UBYTE | USHORT | ULONG | STRUCT

History:

14-Apr-00	NNN	Initial

9.2 The associated controller to a union

The tool chain xgen and ccdgen that convert the description of a primitive from a word document to a C header file will generate a controller with the prefix ctrl_ to ctrl_unionname.

This controller holds the appropriate symbolic tag id name to the current active union member on target at runtime.

The controller always has the type enum. The enum related stuff is described in the following chapters.

9.3 Memory layout of a union

The tables in the subsequent chapters shows the description in a *.mdf or *.pdf file, the layout in memory and the offset information in *.cdg tables. The offset information is important for CCD and TAP. Because together with the union there always is a generated controller for the union to distinguish which union member is active. This controller have to be seen a s part of the union in memory but on C level this controller is separatly to the union. The controller always appears in front of a union.

The type of the controller is a enum and the size of the enum is controlled by the ccdgen options –vei4, vei2 and veb.

These option are very compiler and target dependend.

-vei4 - Treat enums as 32 bit integer (Default)

-vei2 - Treat enums as 16 bit integer

-veb - Best fit for enums - refer to your compiler manual !!!

best fit for enums means that ccdgen calculates the size information of the enum by all given TAGs. If more than 256 TAGs are given the size for the enum goes from 8bit to 16bit. The used C compiler has to work in the same way !

For all given tables below the generic union example looks like this in the *.mdf|pdf file:

```
UNION union_name
{
  TAG1 member1
  TAG2 member2
}
```

where the member can be any type with the exeption of union.

The TAGs are any symbolic names in capital letters. These tags are the enumerators of the generated enum.

e.g.: (*.h)

```
 typedef enum
{
  TAG1 = 0x0,
  TAG2 = 0x1
}T_ctrl_union_name
```

A union is always wrapped by a structure, at least a message or a primitve is the wrapper.

e.g.: (*.mdf|pdf)

```
COMP comp_name
{
  union_name
}
```

9.4 Ccdgen with alignment option –a0

9.4.1 Union is mandatory and ccdgen option –vei4 –a0

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** ctrl_union_name (4bytes)	0
{	{	1	
union_name	T_ctrl_union_name ctrl_union_name	2	
}	T_union_name union_name	3	
	}T_comp_name	**0** union_name	
		1	

9.4.2 Union is mandatory and ccdgen option –vei2 –a0

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** ctrl_union_name (2bytes)	0
{	{	1	
union_name	T_ctrl_union_name ctrl_union_name	2 union_name	
}	T_union_name union_name	3	
	}T_comp_name	**0**	
		1	

9.4.3 Union is mandatory and ccdgen option –veb –a0

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** ctrl_union_name (1byte)	0
{	{	1 union_name	
union_name	T_ctrl_union_name ctrl_union_name	2	
}	T_union_name union_name	3	
	}T_comp_name	**0**	
		1	

9.4.4 Union is optional and ccdgen option –vei4 –a0

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** v_union_name (1byte)	0
{	{	1 ctrl_union_name (4bytes)	
<() union_name >	U8 v_union_name	2	
	T_ctrl_union_name ctrl_union_name	3	
}	T_union_name union_name	**0**	
	}T_comp_name	1 union_name	

9.4.5 Union is optional and ccdgen option –vei2 –a0

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** v_union_name (1byte)	0
{	{	1 ctrl_union_name (2bytes)	
<() union_name >	U8 v_union_name	2	
}	T_ctrl_union_name ctrl_union_name	3 union_name	
	T_union_name union_name	**0**	
	}T_comp_name	1	

9.4.6 Union is optional and ccdgen option –veb –a0

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** v_union_name (1byte)	0
{	{	1 ctrl_union_name (1byte)	
<() union_name >	U8 v_union_name	2 union_name	
}	T_ctrl_union_name ctrl_union_name	3	
	T_union_name union_name	**0**	
	}T_comp_name	1	

9.5 Ccdgen with alignment option –a1

9.5.1 Union is mandatory and ccdgen option –vei4 –a1

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** ctrl_union_name (4bytes)	0
{	{	1	
union_name	T_ctrl_union_name ctrl_union_name	2	
}	T_union_name union_name	3	
	}T_comp_name	**0** union_name	
		1	

9.5.2 Union is mandatory and ccdgen option –vei2 –a1

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** ctrl_union_name (2bytes)	0
{	{	1	
union_name	T_ctrl_union_name ctrl_union_name	2 union_name	
}	T_union_name union_name	3	
	}T_comp_name	**0**	
		1	

9.5.3 Union is mandatory and ccdgen option –veb –a1

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** _align0 (1byte)	
{	{	1 ctrl_union_name (1byte)	1
union_name	U8 _align0	2 union_name	
}	T_ctrl_union_name ctrl_union_name	3	
	T_union_name union_name	**0**	
	}T_comp_name	1	

9.5.4 Union is optional and ccdgen option –vei4 –a1

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** _align0 (1byte)	
{	{	1 v_union_name (1byte)	1
<() union_name >	U8 _align0	2 ctrl_union_name (4bytes)	
	U8 v_union_name	3	
}	T_ctrl_union_name ctrl_union_name	**0**	
	T_union_name union_name	1	
	}T_comp_name	2 union_name	

9.5.5 Union is optional and ccdgen option –vei2 –a1

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** _align0 (1byte)	
{	{	1 v_union_name (1byte)	1
<() union_name >	U8 _align0	2 ctrl_union_name (2bytes)	
}	U8 v_union_name	3	
	T_ctrl_union_name ctrl_union_name	**0** union_name	
	T_union_name union_name	1	
	}T_comp_name		

9.5.6 Union is optional and ccdgen option –veb –a1

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** _align0 (1byte)	
{	{	1 _align1 (1byte)	
<() union_name >	U8 _align0	2 v_union_name (1byte)	2
}	U8 _align1	3 ctrl_union_name (1byte)	
	U8 v_union_name	**0** union_name	
	T_ctrl_union_name ctrl_union_name	1	
	T_union_name union_name		
	}T_comp_name		

9.6 Ccdgen with alignment option –a2

9.6.1 Union is mandatory and ccdgen option –vei4 –a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary % 4)	Offset information
COMP comp_name	typedef struct	**0** ctrl_union_name (4bytes)	0
{	{	1	
union_name	T_ctrl_union_name ctrl_union_name	2	
}	T_union_name union_name	3	
	}T_comp_name	**0** union_name	
		1	

9.6.2 Union is mandatory and ccdgen option –vei2 –a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary % 4)	Offset information
COMP comp_name	typedef struct	**0** _align0 (1byte)	
{	{	1 _align1 (1byte)	
union_name	U8 _align0	2 ctrl_union_name (2bytes)	2
}	U8 _align1	3	
	T_ctrl_union_name ctrl_union_name	**0** union_name	
	T_union_name union_name	1	
	}T_comp_name		

9.6.3 Union is mandatory and ccdgen option –veb –a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary % 4)	Offset information
COMP comp_name	typedef struct	**0** _align0 (1byte)	
{	{	1 _align1 (1byte)	
union_name	U8 _align0	2 _align2 (1byte)	
}	U8 _align1	3 ctrl_union_name (1byte)	3
	U8 _align2	**0** union_name	
	T_ctrl_union_name ctrl_union_name	1	

	T_union_name union_name		
	}T_comp_name		

9.6.4 Union is optional and ccdgen option –vei4 –a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	0 _align0 (1byte)	
{	{	1 _align1 (1byte)	
<() union_name >	U8 _align0	2 _align2 (1byte)	
	U8 _align1	3 v_union_name (1byte)	3
}	U8 _align2	0 ctrl_union_name (4bytes)	
	U8 v_union_name	1	
	T_ctrl_union_name ctrl_union_name	2	
	T_union_name union_name	3	
	}T_comp_name	0 union_name	

9.6.5 Union is optional and ccdgen option –vei2 –a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	0 _align0 (1byte)	
{	{	1 v_union_name (1byte)	1
<() union_name >	U8 _align0	2 ctrl_union_name (2bytes)	
}	U8 v_union_name	3	
	T_ctrl_union_name ctrl_union_name	0 union_name	
	T_union_name union_name	1	
	}T_comp_name		

9.6.6 Union is optional and ccdgen option –veb –a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct		
{	{	0 _align0 (1byte)	
<() union_name >	U8 _align0	1 _align1 (1byte)	

}	U8 _align1	2 v_union_name (1byte)	2
	U8 v_union_name	3 ctrl_union_name (1byte)	
	T_ctrl_union_name ctrl_union_name	**0** union_name	
	T_union_name union_name	1	
	}T_comp_name		

9.7 More complicated examples

The generation will be more complicated when a structure (COMP) starts instead of a union with any other type as member of the structure.

9.7.1 Struct starts with U16 variable and union is optional and ccdgen option –vei2 and alignment -a2

Here the union must be aligned to a long word boundary. The union controller has to be in front of the union and the valid flag must be in front of the controller. There is a gap of three alignment bytes between the U16 variable and the valid flag for the union.

Description in *.mdf|pdf file	**Generated C code**	**Memory layout** (0 is a long word boundary %4)	**Offset information**
COMP comp_name	typedef struct	**0** variable_name (2bytes)	0
{	{	1	
variable_name	U16 variable_name	2 _align0 (1byte)	
<() union_name >	U8 _align0	3 _align1 (1byte)	
}	U8 _align1	**0** _align3 (1byte)	
	U8 _align2	1 v_union_name (1byte)	5
	U8 v_union_name	2 ctrl_union_name (2bytes)	
	T_ctrl_union_name ctrl_union_name	3	
	T_union_name union_name	**0** union_name	
	}T_comp_name		

9.7.2 Struct starts with three U8 variables and union is optional and ccdgen option –veb and alignment -a2

Description in *.mdf|pdf file	Generated C code	Memory layout (0 is a long word boundary %4)	Offset information
COMP comp_name	typedef struct	**0** variable_name1 (1byte)	0
{	{	1 variable_name2 (1byte)	1
variable_name1	U8 variable_name1	2 variable_name3 (1byte)	2
variable_name2	U8 variable_name2	3 _align0 (1byte)	
variable_name3	U8 variable_name3	**0** _align1 (1byte)	
<() union_name >	U8 _align0	1 _align2 (1byte)	
}	U8 _align1	2 v_union_name (1byte)	6
	U8 _align2	3 ctrl_union_name (1byte)	
	U8 v_union_name	**0** union_name	
	T_ctrl_union_name ctrl_union_name		
	T_union_name union_name		
	}T_comp_name		

10 Enums derived from variables with defined values

Contrary to unions where the controller type is always a enum, the developer has the choice to set the type of a variable from U|S8|16|32 to enum by following possibilities.

10.1 Keyword ENUM in MSG or SAP documents

If it makes sense to have a continual enum type instead of plain type as U8 and so on, the developer can set in the definition table in column type the keyword ENUM. The size of the enum is controlled by the ccdgen options –vei4, vei2 and veb.

Description:

Any text for description.

Definition:

type	short name	Comment
ENUM	a_values	A values

Values:

value	c-macro	Comment
1	XYZ_A_VALUES__A_1	A 1 comment
2	XYZ_A_VALUES__A_2	A 2 comment
3	XYZ_A_VALUES__A_3	A 3 comment

History:

15-May-01	SKA	Initial

Appendices

A. Acronyms

DS-WCDMA	Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)

Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>