# DBC-Database CAN

## 1 Introduction

DBC文件描述了单一CAN网络的通信，这个信息足以去监控和分析这个网络并且模拟物理上不可用的节点（剩余总线模拟）

这个DBC文件也被习惯用于开发作为CAN网络一部分电子控制单元的通信软件，ECU的功能行为不是由DBC文件处理的。

## 2 General Definitions

**在这个文档中使用了以下通用元素(定义)**

`unsigned_integer: an unsigned integer`

`signed_integer: a signed integer`

`double: a double precision float number`（双精度浮点数）

`char_string: an arbitrary string consisting of any printable charac-ters except double hyphens ('"').`由除了双连字符（“”）以外任何可打印字符组成的一个任意的字符串

`C_identifier: a valid C_identifier. C_identifiers have to start with am alpha character or an underscore and may further consist of alpha-numeric characters and underscores. C_identifier = (alpha_char | '_') {alpha_num_char | '_'}`

一个有效的C_标识符 。 C标识符必须以字母符号开始或者下划线，还可以由字母数字 符号 和 下划线组成

C标识符 = （字母符号 | '__'）{ 字母数字符号 | '__'}

在DBC文件中使用C标识符的长度最多达到128个字符，为了兼容旧版本的工具长度应该不超过32个字符

在DBC文件中使用其他字符串长度可以任意

在DBC文件中用于识别对象类型的关键字如下表所示

| Keyword | Object Type |
| --- | --- |
| BU_ | Network Node |
| BO_ | Message |
| SG_ | Signal |
| EV_ | Environment Variable |

**语法描述了目前使用的拓展的BNF符号（(Backus-Naur-Format)**

| symbol | Meaning |
| --- | --- |
| = | A name on the left of the = is defined using the syntax on the right (syntax rule). |
| ; | The semicolon terminates a definition 分号终止定义 |
| \| | The vertical bar indicates an alternative 竖线表示另一种选择 |
| [ ... ] | The definitions within brackets are optional (zero or one occurrence).括号内定义是可选的 |
| { ... } | The definitions within braces repeated (zero or multiple occurrences)大括号内定义重复 |
| ( ... ) | Parentheses define grouped elements圆括号定义分组元素 |
| ' ... ' | Text in hyphens has to appear as defined.连字符中的文本必须按定义显示 |
| (* ... *) | Comment 注释 |

# 3 Structure of the DBC File

**DBC文件格式具有以下总体结构**

| **DBC_file =** | |
|---|---|
| version | 版本 |
| new_symbols | 新符号 |
| bit_timing | 位时钟 (*obsolete but required*) 淘汰了但需要 |
| nodes | 节点 |
| value_tables | 变量表 |
| messages | 消息 |
| message_transmitters | 消息发送器 |
| environment_variables | 环境变量 |
| environment_variables_data | 环境变量数据 |
| signal_types | 信号类型 (很少用) |
| comments | 注释 |
| attribute_definitions | 属性定义 |
| sigtype_attr_list | 信号属性表 (很少用) |
| attribute_defaults | 属性默认值 |
| attribute_values | 属性值 |
| value_descriptions | 变量描述 |
| category_definitions | 类别定义(*obsolete*)(很少用) |
| categories | 类别(*obsolete*)(很少用) |
| filter | 过滤器(*obsolete*)(很少用) |
| signal_type_refs | 信号类型~~(很少用) |
| signal_groups | 信号组 |
| signal_extended_value_type_list ; | 信号拓展值类型表(很少用) |

**DBC文件描述的Can网络的基本通信包括了以下部分**

bit_timing 这个部分是需要但是基本为空

nodes 这个部分是需要并且定义了网络节点

message 这个部分定义了消息和信号

**下列部分没有用在普通的DBC文件中，他们被定义在这里仅仅是为了完整性**

signal_types

sigtype_attr_list

category_definitions

categories

filter

signal_type_refs

signal_extended_value_type_list

# 4 Version and New Symbol Specification（规范，说明书）

**DBC文件头包含了版本和新加入的符号。版本要么是空要么是“CANdb”编译器使用的一个字符串**

```
version = ['VERSION' '"' { CANdb_version_string } '"' ];

new_symbols = [ '_NS' ':' ['CM_'] ['BA_DEF_'] ['BA_'] ['VAL_'] ['CAT_DEF_']

['CAT_'] ['FILTER'] ['BA_DEF_DEF_'] ['EV_DATA_']

['ENVVAR_DATA_'] ['SGTYPE_'] ['SGTYPE_VAL_'] ['BA_DEF_SGTYPE_']

['BA_SGTYPE_'] ['SIG_TYPE_REF_'] ['VAL_TABLE_'] ['SIG_GROUP_']

['SIG_VALTYPE_'] ['SIGTYPE_VALTYPE_'] ['BO_TX_BU_']

['BA_DEF_REL_'] ['BA_REL_'] ['BA_DEF_DEF_REL_'] ['BU_SG_REL_']

['BU_EV_REL_'] ['BU_BO_REL_'] ];
```

# 5 Bit Timing Definitions

**位定时 部分定义了波特率和网络BTR寄存器的设置，已经过时没有用了，但关键字“BS”必须出现在DBC文件中。*只出现但是不用***

```
_bit_timing = 'BS_:' [baudrate ':' BTR1 ',' BTR2 ] ;

baudrate = unsigned_integer ;

BTR1 = unsigned_integer ;

BTR2 = unsigned_integer ;
```

# 6 Node Definition

**这个部分定义了所有参与了的节点的名字。在这个部分内名字的定义必须是独一无二的**

```
nodes = 'BU_:' {node_name} ;

node_name = C_identifier ;
```

# 7 Value Table Definition

**值表部分定义了全局值表。值表中的值描述定义了信号原始值的值编码。在常用的DBC文件中，不使用全局值表，而是分别为每个信号定义值描述。**

```
value_tables = {value_table} ;

value_table = 'VAL_TABLE_' value_table_name {value_description} ';' ;

value_table_name = C_identifier ;
```

### 7.1 Value Descriptions(Value Encoding)

**值描述为单个值定义文本描述。该值要么是在总线上传输的信号原始值，要么是剩余总线(虚拟节点)模拟中环境变量的值。**

```
value_description = double char_string;
```

# 8 Message Definition

**消息部分定义了集群中所有帧的名称以及它们的属性和在这些帧上传输的信号。**

```
messages = {message};
```

```
message = BO_  message_id  message_name':'message_size  trans-mitter  {signal} ;
```

```
message_id = unsigned_integer;
```

**该消息的CAN-ID。 CAN-ID在DBC文件中必须是唯一的。如果CAN-ID的最高有效位被设置，则该ID是扩展的CAN ID。扩展的CAN ID可以通过用掩码0xCFFFFFFF屏蔽掉最高有效位来确定。**

```
message_name = C_identifier ;
```

**在这个部分定义的名字在这组消息中必须是唯一的。**

```
message_size = unsigned_integer;
```

**message_size以字节为单位指定消息的大小。**

```
transmitter = node_name | 'Vector__XXX';
```

**发送者名称指定发送消息的节点的名称。发件人名称必须在节点部分的节点名称集中定义。如果消息没有发送者，则必须在这里给出字符串Vector__XXX。**

### 8.1 Signal Definitions

**消息的信号部分列出消息中放置的所有信号，消息数据字段中的位置及其属性。**

```
signal = 'SG_' signal_name multiplexer_indicator ':' start_bit '|' signal_size '@' byte_order value_type '(' factor ',' offset ')' '[' minimum '|' maximum ']' unit receiver {',' receiver} ;
```

```
signal_name = C_identifier ;
```

**这里定义的名称对于单个消息的信号必须是唯一的。**

```
multiplexer_indicator = ' ' | 'M' | m multiplexer_switch_value ;
```

**多路复用器指示器定义信号是正常信号、多路复用信号的多路开关，还是多路复用信号。“M”（大写）字符将信号定义为多路复用器开关。多路复用器交换机中只能有一个信号内的信号。一个“m”（小写）字符，后跟一个无符号整数，将信号定义为复用器开关复用的信号。如果多路复用器信号的开关值等于multiplexer_switch_value 其则多路复用信号被传送到消息中。**

```
start_bit = unsigned_integer ;
```

**start_bit值指定帧的数据字段内信号的位置。对于字节顺序为英特尔（little endian）的信号给出了最低有效位的位置。对于具有字节顺序Motorola（big endian）的信号，给出最高有效位的位置。这些位以锯齿形方式计数。起始位必须在0到（8 * message_size - 1）的范围内。**

```
signal_size = unsigned_integer;
```

**signal_size以位为单位指定信号的大小**

```
byte_order ='0'| '1'; （* 0 =小端，1 =大端*）
```

**如果信号的字节顺序是Intel（小端），则byte_format是0;如果字节顺序是Motorola（大端），则byte_format是1。(这里可能有错误，从其他渠道获取的消息是1=intel，0=Motorola)**

```
value_type ='+'| ' - '; （* + =无符号， - =有符号*）
```

**value_type将信号定义为unsigned（ - ）或signed（ - ）类型。**

```
factor = double;
```

```
offset = double;
```

**系数和偏移定义了线性转换规则，将信号原始值转换为信号的物理值，反之亦然：**

```
physical_value = raw_value * factor + offset
```

```
raw_value =（physical_value - offset）/ factor
```

**从转换规则公式中可以看出，该因子不能为0。**

```
minimum = double ;
```

```
maximum = double ;
```

**最小值和最大值定义信号有效物理值的范围。**

```
unit = char_string;
```

```
receiver = node_name | 'Vector__XXX';
```

**接收者名称指定信号的接收者。接收者名称必须在节点部分的一组节点名称中定义。如果信号没有接收器，字符串'Vector__XXX'必须在这里给出。**

**值为“float”和“double”的信号在sig-nal_valtype_list部分有附加条目。**

```
signal_extended_value_type_list = 'SIG_VALTYPE_' message_id signal_
```

```
name signal_extended_value_type ';' ;
```

```
signal_extended_value_type = '0' | '1' | '2' | '3' ; (* 0=signed or
```

```
unsigned integer, 1=32-bit IEEE-float, 2=64-bit IEEE-double *)
```

### 8.2 Definition of Message Transmitters

**消息发送器部分使得能够定义单个节点的多个发送器节点。这用于描述更高层协议的通信数据。这不用于定义CAN二层通信。**

```
message_transmitters = {message_transmitter} ;
```

```
Message_transmitter = 'BO_TX_BU_' message_id ':' {transmitter} ';' ;
```

### 8.3 Signal Value Descriptions (Value Encodings)

**信号值描述定义了特定信号原始值的编码。**

```
value_descriptions = { value_descriptions_for_signal |
 value_descriptions_for_env_var } ;
value_descriptions_for_signal = 'VAL_' message_id signal_name
 { value_description } ';' ;
```

# 9 Environment Variable Definition

**在环境变量部分中定义了用于系统仿真和剩余总线仿真工具的环境变量。**

```
environment_variables = {environment_variable}
environment_variable = 'EV_' env_var_name ':' env_var_type '[' minimum
 '|' maximum ']' unit initial_value ev_id access_type access_
 node {',' access_node } ';' ;
env_var_name = C_identifier ;
env_var_type = '0' | '1' | '2' ; (* 0=integer, 1=float, 2=string *)
minimum = double ;
maximum = double ;
initial_value = double ;
ev_id = unsigned_integer ; (* obsolete *)
access_type = 'DUMMY_NODE_VECTOR0' | 'DUMMY_NODE_VECTOR1' |
 'DUMMY_NODE_VECTOR2' | 'DUMMY_NODE_VECTOR3' ; (*
 0=unrestricted, 1=read, 2=write, 3=readWrite *)
access_node = node_name | 'VECTOR_XXX' ;
```

**环境变量数据部分中的条目将此处列出的环境定义为数据类型“数据”。这种类型的环境变量可以存储给定长度的任意二进制数据。长度以字节为单位给出。**

```
environment_variables_data = environment_variable_data ;
environment_variable_data = 'ENVVAR_DATA_' env_var_name ':'
 data_size ';' ;
data_size = unsigned_integer ;
```

### 9.1 Environment Variable Value Descriptions

**环境变量的值描述提供了变量特定值的文本表示。**

```
value_descriptions_for_env_var = 'VAL_' env_var_aname
 { value_description } ';' ;
```

# 10 Signal Type and Signal Group Definitions

**信号类型用于定义几个信号的公共属性。它们通常不用于DBC文件。**

```
signal_types = {signal_type} ;
signal_type = 'SGTYPE_' signal_type_name ':' signal_size '@'
 byte_order value_type '(' factor ',' offset ')' '[' minimum '|'
 maximum ']' unit default_value ',' value_table ';' ;
signal_type_name = C_identifier ;
default_value = double ;
value_table = value_table_name ;
signal_type_refs = {signal_type_ref} ;
signal_type_ref = 'SGTYPE_' message_id signal_name ':' signal_
 type_name ';' ;
```

**信号组被用来定义消息内的一组信号，例如，以确定一个组的信号必须共同更新。**

```
signal_groups = 'SIG_GROUP_' message_id signal_group_name repetitions
 ':' { signal_name } ';' ;
signal_group_name = C_identifier ;
repetitions = unsigned_integer ;
```

# 11 Comment Definitions（注释（备注）定义）

注释部分包含对象注释。对于每个有注释的对象，本节定义了一个带有对象类型标识的条目。

```
comments = {comment} ;
comment = 'CM_' (char_string |
 'BU_' node_name char_string |
 'BO_' message_id char_string |
 'SG_' message_id signal_name char_string |
 'EV_' env_var_name char_string)
 ';' ;
```

# 12 User Defined Attribute Definitions

用户定义的属性是扩展DBC文件的对象属性的一种手段。这些附加属性必须使用具有属性默认值的属性定义进行定义。对于每个具有为该属性定义的值的对象，都必须定义一个属性值条目。如果没有为对象定义属性值条目，则该对象的属性值是该属性的默认值。

### 12.1Attribute Definitions

```
attribute_definitions = { attribute_definition } ;

attribute_definition = 'BA_DEF_' object_type attribute_name attribute_

 value_type ';' ;

object_type = '' | 'BU_' | 'BO_' | 'SG_' | 'EV_' ;

attribute_name = '"' C_identifier '"' ;

attribute_value_type = 'INT' signed_integer signed_integer |

 'HEX' signed_integer signed_integer |

 'FLOAT' double double |

 'STRING' |

 'ENUM' [char_string {',' char_string}]


attribute_defaults = { attribute_default } ;

attribute_default = 'BA_DEF_DEF_' attribute_name attribute_value

 ';' ;

attribute_value = unsigned_integer | signed_integer | double |

 char_string ;
```

### 12.2Attribute Values

```
attribute_values = { attribute_value_for_object } ;

attribute_value_for_object = 'BA_' attribute_name (attribute_value |

 'BU_' node_name attribute_value |

 'BO_' message_id attribute_value |

 'SG_' message_id signal_name attribute_value |

 'EV_' env_var_name attribute_value)
```

# 13 Examples

**VERSION ""**

**NS_ :**

`NS_DESC_`

`CM_`

`BA_DEF_`

`BA_`

`VAL_`

`CAT_DEF_`

`CAT_`

`FILTER`

`BA_DEF_DEF_`

`EV_DATA_`

`ENVVAR_DATA_`

`SGTYPE_`

`SGTYPE_VAL_`

`BA_DEF_SGTYPE_`

`BA_SGTYPE_`

`SIG_TYPE_REF_`

`VAL_TABLE_`

`SIG_GROUP_`

`SIG_VALTYPE_`

`SIGTYPE_VALTYPE_`

`BO_TX_BU_`

`BA_DEF_REL_`

`BA_REL_`

`BA_DEF_DEF_REL_`

`BU_SG_REL_`

`BU_EV_REL_`

`BU_BO_REL_`

`BS_:`

```

`BU_: Engine Gateway`

```

`BO_ 100 EngineData: 8 Engine`

```
SG_ PetrolLevel : 24|8@1+ (1,0) [0|255] "l"  Gateway

SG_ EngPower : 48|16@1+ (0.01,0) [0|150] "kW"  Gateway

SG_ EngForce : 32|16@1+ (1,0) [0|0] "N"  Gateway

SG_ IdleRunning : 23|1@1+ (1,0) [0|0] "" Gateway

SG_ EngTemp : 16|7@1+ (2,-50) [-50|150] "degC" Gateway

SG_ EngSpeed : 0|16@1+ (1,0) [0|8000] "rpm"  Gateway
```

```
CM_ "CAN communication matrix for power train electronics
**********************************************************
```

```
implemented: turn lights, warning lights, windows";
```

```
VAL_ 100 IdleRunning 0 "Running" 1 "Idle" ;
```