The ECID Format

Naming CAN messages and decoding signals in PP2CAN / LogAnalyzer – reference manual

Describes the behaviour of the ECID parser (CANMsgDatabase.cpp) including the November 2026 fixes.

1. Introduction ... 3
1.1 Where ECID is used ... 3
2. File structure ... 3
2.1 Numbers ... 3
2.2 Format version (^ECID) ... 4
3. Control characters ... 4
4. Naming rules (>) ... 5
4.1 Rule and template items ... 5
4.2 How the ID is compared ... 5
4.3 How data bytes are compared ... 6
4.4 Order: the first match wins ... 6
5. Templates and the stack ... 6
5.1 How names are built ... 6
5.2 Sections { } ... 6
5.3 + and - ... 7
5.4 -- and ! ... 7
6. Size indication (S) ... 7
7. Signals (@ and :) ... 8
7.1 Common items ... 8
7.2 Bit numbering ... 8
7.3 Signal types ... 9
7.4 Which signals (EYE) are created ... 10
7.5 Reassembled messages: T:ISOTP, T:ISOTPX and T:J1939 ... 10
7.6 Signals do not depend on rule order ... 12
7.7 Template signals (:) ... 12
7.8 Special values (!) ... 12
8. Value lists [ ] ... 12
9. Including files (&) ... 13
10. Evaluation and performance ... 13
11. Errors and the log ... 14
12. Complete examples ... 15
12.1 J1939 / FMS ... 15
12.2 OBD-II over ISO-TP ... 16
12.3 Ready-made libraries ... 16
13. Common mistakes and tips ... 17
14. Converting DBC to ECID (dbc2ecid) ... 17
14.1 Usage ... 17

14.2 What is converted 18
14.3 What is not converted (report) 18
14.4 Example output 18
14.5 Recommended workflow and building 18
Appendix A – quick reference 19

1. Introduction

ECID (Extended CAN IDentification) is a text format that describes how received CAN messages are named and which values (signals) are decoded from them. One ECID file therefore covers two things: a table of message names and a set of signal definitions.

It replaces the older CID format, which only allowed one fixed naming setting for all messages. ECID supports:

- single IDs, ID ranges, ID lists and ID masks (11-bit and 29-bit),
- conditions on data bytes DB0–DB7, including masks and ranges,
- templates with a stack, so shared conditions are written only once,
- a size indication (e.g. the ISO-TP frame length) appended to the message name,
- unsigned, signed, hex, float, bit and string signals with scale, offset, unit and value list,
- including other files (`&`), so libraries (OBD-II, UDS, …) can be built.

1.1 Where ECID is used

- **Data view** and the **log filter** – when the ID decoding mode „ECID“ is selected, every message shows the name returned by ECID.
- **Reassembled messages** – rules and signals with `T:ISOTP` or `T:J1939` are evaluated by the Data view on the reassembled data of a multi-frame transfer (section 7.5). Other windows (Signal receiver, analyzer) only work with single CAN messages.
- **Signals (EYE)** – every `@/:` line produces one EYE signal. The signals are offered in the list of predefined signals („Templates / ECID signals“) and are shown in the Data view and the Signal receiver.

> Message naming and signals are independent. The name is decided by the first matching `>` rule, but signals are created from all rules, including rules that never „win“ the naming (see section 7.6).

2. File structure

- Plain text, recommended extension `.ecid`. ANSI/cp1250 encoding (accented text, `°C`). CRLF and LF line endings both work.
- Each line is processed on its own. Leading and trailing spaces and tabs are ignored, so lines can be indented freely.
- The **first character** (control character) defines what the line means. A line that does not start with a control character is silently ignored.
- A control character must be **at the start of a new line**. Two control characters cannot share one line (e.g. `%1:x22,%2:xF1` is an error).
- Items on a line are separated by a comma `,`, the parameters of an item by a colon `:`. The decimal separator is a dot.
- Line length is not limited.
- A comment is a line starting with `;`. Trailing comments after items are not supported.

2.1 Numbers

Notation	Example	Meaning
decimal	`123`	Number 123.
hex with x prefix	`x7E8`, `X7e8`	Hexadecimal 0x7E8 (recommended).
hex with 0x prefix	`0x7E8`	Also hexadecimal.
contains A–F	`7E8`	No prefix but contains a letter A–F → treated as hex. Note: `100` is decimal!
range	`xFD:FE`	If the first bound is hex, the second one is read as hex too (`FE` = 0xFE).

> Recommendation: always write IDs, data and masks with the `x` prefix. Write scales, offsets and bit positions in decimal.

2.2 Format version (^ECID)

The main file may state the format version it is written for at its beginning (after the leading comments). The parser in this program version supports version 1.5.

```
; UDS (ISO 14229-1) ...
^ECID:1.5
[UDS_NRC
```

Without a ^ECID line the main file is taken as the parser version (1.5). An included file without ^ECID is taken as the version of the main file – nothing is reported.

Written to the log (chapter 11), loading continues: the file requires a higher version than the parser supports; an included file states a different version than the main file; ^ECID after the first rule, list or include; a second ^ECID in the same file (the first one applies); a wrong format or an unknown ^ directive (the line is skipped).

Versions are compared numerically (major.minor, 1.10 is higher than 1.9). Older program versions skip the ^ECID line without an error. All files of the ECID_knihovna.zip library contain ^ECID:1.5.

3. Control characters

Char	Name	Function
`;`	comment	The line is ignored.
`>`	rule	Adds a naming rule: a copy of the current template + the items on the line + the name `#`.
`%`	template	Adds/changes items of the current template; `#text` appends text to the template name.
`+`	template with push	Pushes the template onto the stack, then works like `%`.
`{`	section start	Pushes the template onto the stack.
`}`	section end	Restores the template from the stack.
`-`	pop	Restores the template from the stack (like `}`).
`--`	reset	Clears the stack and resets the template.
`!`	partial reset	`!` whole template, `!N` name, `!S` size, `!L` ID list, `!I` ID/range, `!M` ID mask, `!m` all byte masks, `!0`–`!7` one byte including its mask.
`=`	template name	Outside a value list: sets (replaces) the template name.
`[ ]`	value list	Start `[Name` and end `]` of a value list. Only `=` lines are allowed inside.
`=`	list entry	Inside `[ ]`: `=value:text`, several entries separated by commas.
`@`	signal	A signal (EYE) attached to the immediately preceding valid `>` rule.
`:`	template signal	A signal attached to the current template (its ID and conditions, no rule needed).
`&`	include	Loads another ECID file at this position.
`^`	format version	^ECID:major.minor – format version of the file (chapter 2.2). Written at the beginning of the file.

Char	Name	Function
`?`	signal description	Description of the immediately preceding signal (@ or :), several lines are joined (chapter 7.1).

4. Naming rules (>)

A rule describes which messages get a given name. It consists of comma-separated items. **The name `#` must be the last item** and is mandatory. A rule without `#` is dropped and an error is reported.

```
>I:x123,#Motor status
>I:x18FEF100,E,M:x3FFFF00,#CCVS
>R:x100:x10F,#Group 100-10F
>L:x200:x201:x205,#List 200/201/205
>I:x7E8,1:x41,2:x0C,#OBD engine speed response
>I:x300,m0:xF0,0:x20,#ISO-TP consecutive frame
>I:x400,0:x10:x1F,#DB0 range 10-1F
>I:x400,#Other 400
```

4.1 Rule and template items

Item	Syntax	Meaning
I	`I:id` `I:*`	A single ID. `I:*` clears the ID (the rule applies to any ID). Also clears an `L` list.
R	`R:od:do`	An inclusive ID range. Clears an `L` list. Error if from > to.
L	`L:id1:id2:…`	A list of IDs. Takes precedence over `I`/`R` from the template.
M	`M:maska` `M:*`	ID mask. Only bits set to 1 are compared. Applies to `I`, `R` and `L`. `M:*` clears the mask.
E	`E`	29-bit (extended) ID. Without `E` the ID is 11-bit. Can only be cleared with `!`.

0 … 7	`N:hodnota`
	`N:od:do`
	`N:*`
		Condition on data byte DBN: an exact value or a range (0–255). `N:*` clears the condition.

m0 … m7	`mN:maska` `mN:*`	Mask for byte DBN. Used only when the byte also has a value. A mask alone checks nothing.
S	`S:bit:délka[:B]`	Size indication – see chapter 6.

T	`T:ISOTP`
	`T:ISOTPX`
	`T:J1939`
	`T:*`
		Transport layer – the rule and its signals apply to the reassembled data of a multi-frame transfer (section 7.5). `T:ISOTPX` = ISO-TP with extended addressing. `T:*` switches back to single CAN messages.

#	`#text`	Name. Last item, may contain commas. An empty `#` is allowed (only the template name remains).

4.2 How the ID is compared

- First the ID type must match: a rule with `E` applies only to 29-bit messages, without `E` only to 11-bit ones.
- Without a mask: `I` means equality, `R` means from ≤ ID ≤ to, `L` means equality with one of the entries.
- With a mask both sides are masked first: `(ID & M) == (I & M)`, for a range `(from & M) ≤ (ID & M) ≤ (to & M)`.

- A rule without any ID (not even in the template) applies to all IDs of that type.

Mask examples:

Syntax	Matching IDs
`I:x7E8,M:x7F8`	7E8 – 7EF
`I:x18DA00F1,E,M:x1FFF00FF`	18DAxxF1 (UDS physical request, any target address)
`I:x00FEF100,E,M:x3FFFF00`	J1939 PGN FEF1 with any priority and source address (e.g. 18FEF100, 0CFEF117)

4.3 How data bytes are compared

Only bytes with a value or a range are checked. For byte N with mask m: `(from & m) ≤ (DBN & m) ≤ (to & m)`. Without a mask m = xFF.

- `m0:xF0,0:x20` – the upper nibble of DB0 must be 2 (ISO-TP consecutive frame 0x20–0x2F).
- `m2:x01,2:x01` – bit 0 of DB2 must be 1 (e.g. an odd SecurityAccess sub-function).
- `2:xFD:FE,3:x00:xFF` – DB2 in the range 0xFD–0xFE and DB3 anything.

> Rules only see bytes DB0–DB7. For CAN FD messages longer than 8 bytes, conditions on further bytes cannot be expressed.

4.4 Order: the first match wins

Rules are evaluated in file order (after all includes are expanded). A message gets the name of the **first** rule that matches both the ID and the data. Specific rules therefore go **before** general ones. A catch-all rule („everything else“) comes last, typically with the wildcard `*`.

```
>I:x400,0:x10:x1F,#DB0 range 10-1F     ; 400 [15 ..] -> "DB0 range 10-1F"
>I:x400,#Other 400                     ; 400 [25 ..] -> "Other 400"
```

> The `;` comments in the examples are for the reader only. In an ECID file a comment must be on its own line.

If no rule matches, the name is empty.

5. Templates and the stack

A template is a „rule in progress“: a set of items (ID, mask, E, bytes, size, name) that is added automatically to every following `>` rule. A rule starts as a copy of the current template, and its own items add to or override it. At the start of a file the template is empty.

5.1 How names are built

Every `#text` on a `%` or `+` line appends text to the template name (with a space). A rule name is then `template name + space + own name`. An empty own `#` adds no space. A `=text` line replaces the template name and `!N` clears it.

5.2 Sections { }

`{` pushes the template onto the stack and `}` restores it. Anything added to the template inside the section is gone after `}`. Sections can be nested freely.

```
--
%I:x500
%m0:xF0,0:x00,#SF
%S:0:4
{
%1:x10,#Session
>2:x01,#default
>2:x03,#extended
>2:*,#other
}
>1:x3E,#TesterPresent
```

Message 0x500	Name
`02 10 03`	`SF Session extended size:2`
`02 10 05`	`SF Session other size:2`
`01 3E`	`SF TesterPresent size:1`
`12 10 …`	(empty – DB0 & F0 ≠ 0)

5.3 + and -

`+` is a shortcut for `{` followed by `%`. The saved state is restored with `-` (or `}`).

```
--
%I:x700,#Base
+1:x01,#A
>2:x01,#x
-
>2:x02,#y
```

Message 700 `00 01 01` is named `Base A x`, message 700 `00 xx 02` is named `Base y`.

5.4 -- and !

`--` empties the stack and the template, so everything starts from scratch. It is useful as a separator between independent blocks (e.g. the individual IDs in a main file). `!` resets only a selected part of the template:

```
--
=Fixed name
%I:x600
>0:x01,#one          ; 600 [01] -> "Fixed name one"
!N
>0:x02,#two          ; 600 [02] -> "two"
```

> An unclosed `{` at the end of the file, and a `}` or `-` with an empty stack, are reported as errors.

6. Size indication (S)

`S:bit:length[:B]` tells where the transfer length is stored in the message. The value is read like an unsigned signal (bit = LSB position, length in bits, `B` = Motorola, otherwise Intel) and `size:N` is appended to the message name.

Use	Syntax	Example name
ISO-TP Single Frame	`S:0:4`	`02 10 03` → `… size:2`
ISO-TP First Frame	`S:8:12:B`	`10 14 62 …` → `… size:20`

Valid values: bit 0–63, length 1–64; for Intel the field must end by bit 63 (bit + length ≤ 64). An invalid `S:` item in a template leaves the template unchanged (the previous value stays). `!S` turns the indication off in the template.

7. Signals (@ and :)

An `@` line defines a signal – a value decoded from the message and displayed. It belongs to the **immediately preceding valid `>` rule** and takes over its ID, mask, ID type and data conditions. If that rule has an error, the signal is rejected, so it can never end up attached to a different message by mistake.

A `:` line has the same syntax but belongs to the **current template**. It is useful for values shared by a whole group of rules (e.g. SID and NRC in a negative response). The template must have an ID.

```
[OnOff
=0:Off,=1:On
=2:Error,=3:N/A
]
>I:x00FEF100,E,M:x3FFFF00,#CCVS
@U:8:16:L:0.00390625:0,*km/h,#Wheel speed
@H:24:2:L,$OnOff,#CC active
@B:7:N,#Bit7 negated
```

7.1 Common items

Item	Meaning
`#text`	Signal name. Without ? the name of the last rule becomes the signal description (shown in the Data view tooltip).
`##`	Signal name = the name of the last rule.
`*text`	Unit (e.g. `*km/h`, `*°C`).
`?text`	Signal description (Data view tooltip, EYE window). Must be the last item and may contain commas. Write a longer description on following ? lines.
`$jméno`	A value list, which must be defined earlier (chapter 8).

`!hodnota:text`
`!od..do:text`
`!J`
	Special raw value (e.g. N/A, error) – only the text is shown and the value is left out of min/max/average (section 7.8). Several `!` items are allowed.

type	Exactly one of the items `U S H F B T` (see 7.3). Without a type the error „Unknown data type“ is reported.

The description can also be written on separate lines starting with ? right after the signal (@ or :). Several ? lines are joined with a space, after a description from the signal line if any. Comments (;) in between do not matter; any other line ends the description of the signal. A ? line without a preceding signal is written to the log and skipped. For a signal with an ID range or list, all its copies get the description.

```
@U:24:16:L:0.125:0,*rpm,#Engine speed,?SPN 190
?Actual engine speed, calculated over
?a minimum crankshaft angle of 720 degrees.
```

The description length is not limited in ECID. The Data view tooltip shows about 480 characters; saving to .sye truncates the description to 255 characters.

Unlike a `>` rule, the order of items on an `@` line does not matter.

7.2 Bit numbering

A bit position is `byte × 8 + bit within the byte`, with bit 0 being the least significant bit of the byte. The `bit` of a signal always means the **least significant bit (LSB)** of the signal, for Motorola too.

Byte	b7	b6	b5	b4	b3	b2	b1 … b0
DB0	7	6	5	4	3	2	1 … 0
DB1	15	14	13	12	11	10	9 … 8
DB2	23	22	21	20	19	18	17 … 16

Byte	b7	b6	b5	b4	b3	b2	b1 … b0
DB3	31	30	29	28	27	26	25 … 24
DB4	39	38	37	36	35	34	33 … 32
DB5	47	46	45	44	43	42	41 … 40
DB6	55	54	53	52	51	50	49 … 48
DB7	63	62	61	60	59	58	57 … 56

Intel (L, little endian)

Starting at the LSB, bits are taken upwards; after bit 7 of a byte the next bit is bit 0 of the **next** byte. A 16-bit value in DB1 (LSB) and DB2 (MSB) is written `U:8:16:L`.

Motorola (B, big endian)

Starting at the LSB, bits are taken upwards; after bit 7 of a byte the next bit is bit 0 of the **previous** byte. For a byte-aligned value in DBa (MSB) … DBb (LSB): `bit = b × 8`.

Value	Syntax
16 bits, DB3 = MSB, DB4 = LSB (OBD engine speed)	`@U:32:16:B:0.25:0`
32 bits, DB3 … DB6	`@H:48:32:B`
12 bits: DB1 bits 0–3 (MSB) + all of DB2 (LSB)	`@S:16:12:B:1:0`
ISO-TP FF length: DB0 bits 0–3 + DB1	`S:8:12:B`

> Note the difference from DBC: in DBC the Motorola start bit is the MSB. In ECID (and EYE) it is always the LSB.

7.3 Signal types

Type	Syntax	Description
U	`U:bit:délka:L|B:váha:offset`	Unsigned integer. All 5 parameters are mandatory.
S	`S:bit:délka:L|B:váha:offset`	Signed (two's complement, sign = the most significant bit of the signal).
H	`H:bit:délka:L|B`	Shown in hex (status words, DTCs, IDs). Scale and offset are ignored even if given.
F	`F:bit:32|64:L|B:váha:offset`	IEEE 754 float (32) or double (64).
F (x)	`F:bit:x:exponent:mantisa:L|B:váha:offset`	User float with the given number of exponent and mantissa bits.
B	`B:bit[:N]`	A single bit. `N` = negation (the „on“ state corresponds to value 0).
T	`T:bajt:počet[:R]`	Text (ASCII). Counted in **bytes** from DB0. `R` = reversed byte order.
T (*)	`T:*:index`	Field of a `*`-delimited text (J1939 VI, DI, CI, ECUID: `field0*field1*`), index from 0. With `T:J1939` it works for a single message as well as a reassembled transfer (BAM).

Type	Syntax	Description
J	`J:bit` `J:bit:verze`	J1939 DTC – 32 bits Intel (SPN, FMI, CM, OC) → `SPN 100 FMI 3 OC 1`. CM bit = 0: SPN version 4. CM bit = 1: SPN in version 1, 2 or 3 (J1939-73 of 1996), which cannot be told from the data – give it as `version` (1, 2, 3). Without a version the SPN is shown per version 1, followed in brackets by the SPN for versions 2 and 3: `SPN 100 FMI 1 OC 3 (CM1 v1; v2: SPN 24580, v3: SPN 265216)`; a known SPN tells which version applies.
P	`P:bit`	PGN reference – 24 bits Intel → `PGN FECA00`.
O	`O:bit`	OBD DTC (SAE J2012) – 2 bytes Motorola, `bit` = LSB → `P0301`.
D	`D:bit`	UDS DTC (ISO 14229) – 3 bytes Motorola, `bit` = LSB → `P0301-17` (last byte = failure type).

The physical value is computed as:

```
value = raw × scale − offset
```

The offset is **subtracted**. A temperature with a –40 °C offset (raw 0 = –40 °C) is therefore written `U:24:8:L:1:40`. The J1939 TD year (raw 0 = 1985) is written `U:40:8:L:1:-1985`.

> Difference from DBC: in DBC `phys = raw × factor + offset`. When converting from DBC, flip the sign of the offset.

7.4 Which signals (EYE) are created

- A rule with one ID: one EYE with the same ID, mask and ID type.
- A rule with an `L` list or `R` range: one EYE for **each** ID. IDs that are equal after masking are created only once. At most 4096 IDs per signal; the rest is reported and ignored.
- Data conditions are transferred to the EYE (byte check with mask), so the signal is shown only for messages that meet them.
- EYE does not support a **byte range** (`N:from:to`). The byte is then not checked at all and the parser prints a warning.
- A signal cannot be created without an ID (not even in the template) – error.

7.5 Reassembled messages: T:ISOTP, T:ISOTPX and T:J1939

A response longer than one CAN message is sent in parts. The `T:` item in a rule or template says that the rule and its signals apply to the **reassembled data** of the whole transfer. The reassembly is done by the Data view.

Syntax	Meaning
`T:ISOTP`	ISO-TP (ISO 15765-2): Single Frame, First Frame + Consecutive Frames, classic CAN and CAN FD. Byte 0 = first byte after the PCI (the SID for diagnostics). Signal positions count from it and may lie far beyond byte 8 (max. 8192 bytes).
`T:ISOTPX`	ISO-TP with extended / mixed addressing (ISO 15765-2): byte 0 of the CAN message is an address (N_TA of the target ECU, the tester address in responses), the PCI and data start at byte 1. A classic frame therefore carries 6 data bytes instead of 7. Transfers on the same CAN ID with different addresses are reassembled separately. Byte 0 of the reassembled data = SID as with `T:ISOTP`. Used e.g. by BMW (tester 6F1, ECUs 6xx).

Syntax	Meaning
`T:J1939`	J1939 TP (BAM, RTS/CTS). Positions refer to the PGN data – identical for a single message (8 B) and for a reassembled transfer. The PGN is added to the list of reassembled PGNs automatically.
`T:*`	Back to single CAN messages (default).

```
--
%I:x7E8,M:x7F8,#UDS Resp
{
%T:ISOTP
>0:x62,1:xF1,2:x90,#VIN
@T:3:17,#VIN
>0:x59,1:x02,#DTC by status
@D:40,#DTC 1
@D:72,#DTC 2
>0:x49,1:x17,#OBFCM
@U:144:32:B:0.01:0,!xFFFFFFFF:N/A,*l,#Lifetime fuel
}
--
>I:x00FECA00,E,M:x3FFFF00,T:J1939,#DM1
@J:16,#DTC 1
@J:48,#DTC 2
>I:x00FE6B00,E,M:x3FFFF00,T:J1939,#DI Driver Identification
@T:*:0,#Driver 1 identification
@T:*:1,#Driver 2 identification
--
{
%I:x6F1,#Diag Req
%T:ISOTPX
>0:x22,#ReadDataByIdentifier
}
{
%I:x600,M:x700,#Diag Resp
%T:ISOTPX
>0:x62,1:xF1,2:x90,#VIN
@T:3:17,#VIN
}
```

How it behaves in the Data view:

- ISO-TP is reassembled only for IDs that have a `T:ISOTP` / `T:ISOTPX` signal or (with ECID name decoding) a `T:ISOTP` / `T:ISOTPX` rule. No EYE or DBC signals are needed for that – ECID alone gives names and data. Signal values are shown when the signals are loaded as EYEs. The addressing mode is decided by the first signal or rule for the ID.
- The same applies to J1939 TP: the PGNs of `T:J1939` rules are added to the list of reassembled PGNs automatically with ECID name decoding (even without EYEs), and the list is refreshed when another ECID is loaded. The `MULTI PGN` row then has the name, length and reassembled data.
- Transfer errors: a lost, swapped or duplicated Consecutive Frame discards the transfer; a Consecutive Frame without a First Frame is ignored; a new Single Frame or First Frame replaces a transfer in progress; a gap of more than 2 s between frames of a transfer (or time going backwards – a new log) discards it. An incomplete transfer updates nothing.
- A reassembled message has its own row `ISO-TP 7E8` (with `T:ISOTPX` including the address, e.g. `ISO-TP 612 [F1]`) with length and data. ISO-TP signals are shown only under this row. The single CAN messages (SF/FF/CF/FC) keep their own rows.
- With ECID name decoding every rule (service, DID) has its own row: reading the VIN, the serial number, DTCs and a negative response `7F … 78` (ResponsePending) stay visible side by side and a later response does not overwrite an earlier one. Messages without a rule get a row per first byte (SID). Without ECID there is one row per ID.
- `T:ISOTP` rules are skipped when naming single CAN messages and vice versa – the two sets do not affect each other.

- Byte conditions `0`–`7` and `m0`–`m7` with `T:` refer to the first 8 bytes of the reassembled data. For ISO-TP the mask of a standard ID applies too (e.g. `I:x7E8,M:x7F8` = 7E8–7EF).
- A signal lying beyond the end of the received data (e.g. DTC 5 in a response with 3 faults) is not updated.
- A signal for a masked ID (7E8–7EF, 18DAF1xx) has its own copy for every ECU (the actual ID, with `T:ISOTPX` also the address) and every row. If several ECUs answer, the values do not mix and each is shown under the row of its ECU – in normal and in replicate mode.

> Without `T:` a signal is decoded from a single CAN message. For ISO-TP diagnostics without `T:ISOTP`, signals only make sense in a Single Frame (positions including the PCI byte).

7.6 Signals do not depend on rule order

A rule that never wins the naming (because a more general rule with the same ID comes before it) still creates its signals. This can be used deliberately to define names once and signals separately, but it also means you have to make sure the same signal is not created twice.

```
>I:x00FEF100,E,M:x3FFFF00,#CCVS
>I:x00FEF100,E,M:x3FFFF00,#CCVS signals
@U:8:16:L:0.00390625:0,*km/h,#Wheel speed
```

The message is named `CCVS`, yet the „Wheel speed“ signal exists.

7.7 Template signals (:)

```
{
%1:x7F,#Negative response
:H:16:8:L,$UDS_SID,#Rejected SID
:H:24:8:L,$UDS_NRC,#NRC
>3:x78,#ResponsePending
>3:*,#
}
```

Both signals are created once for the whole group, not once per rule.

7.8 Special values (!)

Many protocols reserve some raw values for „not available“ or „error“ (J1939 FE/FF, OBFCM FFFFFFFF). The `!` item on a signal line marks them: the Data view shows only the text and leaves the value out of min/max/average and scaling.

```
@U:144:32:B:0.01:0,!xFFFFFFFF:N/A,*l,#Lifetime fuel
@U:0:8:L:1:40,!xFE:error,!xFF:not available,*°C,#Coolant
@U:8:16:L:0.00390625:0,!J,*km/h,#Wheel based speed
```

- `!value:text` or `!from..to:text`. Without text `N/A` is shown. The **raw** value is compared (before scale and offset); for signed signals the bit pattern of the signal.
- `!J` adds the J1939-71 rules according to the signal length: 2 bits 2/3, 4 bits E/F, 8/16/24/32 bits FE…/FF… = error / not available.
- Special values are also saved to the .eye file.

8. Value lists [ ]

A value list maps raw signal values to text. A signal refers to it with the item `$name`.

```
[Switch
=0:switched off,=1:switched on
=x2:error: really
=3:not available
]
```

- `[name` starts the list and `]` ends it. Only `=` lines are allowed inside. Any other control character in a list is an error.

- An entry is `value:text`. The value can be decimal or hex (`x2`). The text is split at the **first** colon, so it may itself contain colons (`error: really`).
- Several entries can share a line, separated by commas; further entries may start with `=` but do not have to. The text therefore **must not contain a comma**.
- A list must be defined **before** it is first used. Lists are global and also apply in included files.
- A value that is not in the list is shown as a number only.
- **Range** `=from..to:text` (e.g. `=x80..xFF:manufacturer specific`). A range of up to 256 values is expanded into single values (works everywhere); a larger range is stored as a range and shown by the Data view. An exact value takes precedence over a range, a range of up to 256 values over a larger one, and among larger ranges the narrower one wins – regardless of the order in the list. Among overlapping ranges of up to 256 values the one written first wins.

9. Including files (&)

```
&OBD2_resp.ecid
&C:\ECID\common\lists.ecid
```

- A relative path is resolved from the folder of the file that contains the `&`. Both `\` and `/` work as separators.
- An absolute path (`C:\…`, `\\server\…`, `/…`) is used as it is.
- The included file is processed exactly at the position of the `&`, with the **same template and stack**. It can use the template (e.g. a service body included into a block with an ID), but it can also change it. It should therefore have balanced `{ }` and should not contain `--`.
- The same file can be included several times (e.g. a UDS body for 11-bit and 29-bit IDs).
- Nesting is limited to 16 levels. Deeper nesting (typically a cycle) is reported as an error.

Typical use – one body for several IDs:

```
--
%I:x7E0,M:x7F8,#UDS Req
&UDS_req.ecid
--
%I:x7E8,M:x7F8,#UDS Resp
&UDS_resp.ecid
--
%I:x18DAF100,E,M:x1FFFFF00,#UDS Resp
&UDS_resp.ecid
--
```

10. Evaluation and performance

When a message is displayed, its name is looked up as follows:

1. Cache lookup by ID and only those data bits that actually matter for this ID.
2. If the message is not cached, the rules are scanned in file order. The first rule matching the ID type, ID and data wins.
3. If the rule has `S`, the size is read and `size:N` is appended to the name.
4. The result (including „nothing found“) is stored in the cache.

Tips for fast and readable files:

- Split by ID as early as possible (`--` / `%I:` blocks in the main file). Rules for other IDs are then not considered at all.
- Limit data conditions to the bytes that really distinguish messages. Fewer bytes mean a smaller cache.
- Avoid large ID ranges (`R:0:xFFFFFFFF`) for signals; they would create thousands of EYEs. Use a mask instead.

11. Errors and the log

Load errors are written to `<file>.ecid.log` next to the main file. If there are any errors, the log is opened in Notepad automatically after loading. Each entry contains the file, the line number, the line text and a description. A faulty rule or signal is skipped; the rest of the file is loaded.

```
Filename: C:\ECID\err.ecid Line: 1
>I:x100
Name (#) not found, the rule is ignored
---------------
Filename: C:\ECID\err.ecid Line: 2
@U:0:8:L:1:0,#Orphan
An identifier, list of identifiers, or range of identifiers was not specified (or the previous
> line has an error)
---------------
```

Message	Cause / fix
Name (#) not found, the rule is ignored	A `>` rule without `#`. Add a name (an empty `#` is fine).
An identifier … was not specified (or the previous > line has an error)	`@` without a valid preceding rule, or `:` in a template without an ID.
Unknown parameter: … (control character must be at the beginning of a new line)	Unknown item, or a second control character on the same line.
Unknown parameter (ignored)	Unknown item on an `@` line; it is skipped.
Unknown data type	The `@` line has no type `U S H F B T`.
Wrong number of parameters for …	Type parameters are missing (e.g. U needs bit:length:endian:scale:offset).
Wrong FLOAT type paramater (32, 64 or x)	An F length other than 32, 64 or x.
Data byte: value out of range 0..255 or from > to	Byte value outside 0–255 or a reversed range.
Data mask: value out of range 0..255	Byte mask outside 0–255.
Range: from > to	`R` with reversed bounds.
Size: wrong first bit or length	`S` out of range (bit 0–63, length 1–64).
Not found name section.	`$name` refers to a list that has not been defined yet.
Naming section - …	Value list error: missing `]`, `]` without `[`, empty list, or an illegal character inside.
Stack is empty	`}` or `-` without a matching `{` / `+`.
Section { not closed with } at the end of file	Unclosed section.
Unknown command	Unknown variant of `!` or `-`.
File not found / File include error	The file or an included file cannot be opened; check the path.
Include nested too deep (cyclic include?)	More than 16 levels of `&`, usually a cycle.
Too many identifiers for one signal (max 4096)	A signal on a too large ID range or list.
Warning: data byte range is not supported for signals	A signal on a rule with a byte range; the byte is not checked in the EYE.
STRING field index out of range (T:*:0..127)	The field index of `@T:*:index` is greater than 127.
Unknown transport (T:ISOTP, T:ISOTPX, T:J1939 or T:*)	Unknown value of the `T:` item.
J1939 DTC SPN version for CM=1 must be 1, 2 or 3	The version in `@J:bit:version` is outside 1–3.

Message	Cause / fix
Wrong number of parameters for DTC/PGN	The bit position is missing for `@J`, `@P`, `@O`, `@D`.
Naming section - range from > to	Reversed range in a value list (`=from..to:text`).
Special value: …	Invalid `!` item (missing value, reversed range, `!J` with an unsupported length).
File requires ECID x.y, parser supports 1.5 - some items may not be loaded correctly	The file is for a newer format version. Update the program, or expect that some items will not be loaded.
Included file has ECID version x.y, main file a.b	The included file states a different version than the main file. Unify ^ECID, or omit it in the included file.
ECID version should be at the beginning of the file (before rules, lists and includes)	^ECID comes after a rule, list or &. The version still applies; move the line to the beginning.
ECID version already set in this file, ignored	A second ^ECID in the same file. The first one applies.
Wrong ECID version format (^ECID:major.minor, e.g. ^ECID:1.5), ignored	Wrong version format, e.g. ^ECID:1 or ^ECID:1.x.
Unknown directive (^ECID:major.minor expected), ignored	The line starts with ^ but is not ^ECID:.
Description (?) without preceding signal (@ or :), ignored	The ? line is not right after a signal (e.g. after a rule > or at the beginning of the file).

12. Complete examples

12.1 J1939 / FMS

```
; J1939 - priority and SA are ignored
[Pedal
=0:released,=1:pressed,=2:error,=3:not available
]
--
%E,M:x3FFFF00
>I:x00FEF100,#CCVS Cruise Control/Vehicle Speed
@U:8:16:L:0.00390625:0,*km/h,#Wheel based vehicle speed
@H:28:2:L,$Pedal,#Brake switch
>I:x00F00400,#EEC1 Electronic Engine Controller 1
@U:16:8:L:1:125,*%,#Actual engine percent torque
@U:24:16:L:0.125:0,*rpm,#Engine speed
>I:x00FEEE00,#ET1 Engine Temperature 1
@U:0:8:L:1:40,*°C,#Engine coolant temperature
--
```

12.2 OBD-II over ISO-TP

```
--
%I:x7E8,M:x7F8,#OBD Resp
{
%m0:xF0,0:x00,#SF
%S:0:4
{
%1:x41,#Mode01
>2:x0C,#Engine speed
@U:32:16:B:0.25:0,*rpm,#Engine speed
>2:x05,#Coolant temperature
@U:24:8:L:1:40,*°C,#Coolant temperature
>2:*,#other PID
}
}
>m0:xF0,0:x10,S:8:12:B,#FF
>m0:xF0,0:x20,#CF
--
```

Message 0x7E8	Result
`04 41 0C 1A F8`	`OBD Resp SF Mode01 Engine speed size:4`, Engine speed = 1726 rpm
`03 41 05 7B`	`… Coolant temperature size:3`, 83 °C
`10 14 49 02 …`	`OBD Resp FF size:20`
`21 …`	`OBD Resp CF`

12.3 Ready-made libraries

File	Content
UDS_ISO14229.ecid	UDS (ISO 14229) + complete OBD-II (~180 PIDs, Mode 01–0A, DIDs F4xx) main file. 11-bit and 29-bit IDs per ISO 15765-4. Includes UDS_req/resp, UDS_DID_SF/FF and OBD2_req/resp.
KWP2000_CAN.ecid	KWP 2000 (ISO 14230-3) over CAN + OBD-II main file. Includes KWP_req/resp and OBD2_req/resp.
FMS.ecid	FMS-Standard / J1939: CCVS, EEC1/2, LFC, HRLFC, DD, ET1, HOURS, VDHR, SERV, TCO1, AMB, DC1/2, DM1 (DTC 1–10), VI, DI, CI, ECUID (BAM) and more.
OBD2_J1979.ecid	Standalone OBD-II (SAE J1979) main file without UDS: 11-bit and 29-bit IDs per ISO 15765-4, single SF/FF/CF/FC frames and assembled ISO-TP with signals of all PIDs, negative response 7F. Includes OBD2_frames_req/resp and OBD2_req/resp(_TP).
CANopen_CiA301.ecid	CANopen CiA 301 (pre-defined connection set): NMT, SYNC, TIME, EMCY (error codes, error register), SDO (command, index/sub-index, data, abort codes), heartbeat/boot-up, LSS and TPDO/RPDO 1–4. Names include node 1–127; signals are shared by all nodes (ID mask) – with several nodes the values alternate. PDOs names only (content per EDS). Includes CANopen_SDO_*.ecid.
CHAdeMO.ecid	CHAdeMO (11-bit, 500 kbit/s): vehicle 100h–102h, charger 108h–109h (voltages, currents, SOC, times, status and fault bits), V2H 200h, 208h, 209h. Dedicated connector bus only.
GBT27930.ecid	GB/T 27930-2015 (Chinese DC charging, J1939 29-bit): CHM, BHM, CRM, BRM, BCP, CTS, CML, BRO, CRO, BCL, BCS, CCS, BSM, BMV (256 cells), BMT, BST, CST, BSD, CSD, BEM, CEM, TP.CM/TP.DT. Multi-frame messages via T:J1939.

File	Content
XCP_CAN.ecid CCP.ecid	XCP on CAN and CCP 2.1: names of commands, responses, errors and events, parameters of the main commands (MTA, DAQ). IDs are not standardized – adjust the sample IDs to the A2L. Responses and DAQ names only (content depends on command / A2L). Includes XCP_cmd/XCP_res.
UDS_VW.ecid	UDS + OBD-II + VW group: identification DIDs, Simos18 engine, MEB (ID.3/ID.4/Enyaq) – battery, cells, temperatures, DC-DC, climate.
UDS_Ford_MachE.ecid UDS_GM_Bolt.ecid UDS_Jaguar_IPace.ecid UDS_MG_ZSEV.ecid UDS_Renault_Zoe.ecid	UDS + OBD-II + manufacturer DIDs of the given EV (SOC, voltage, current, battery temperatures…). One file per make, because makes share IDs with different DIDs.
examples/	Commented examples for this manual: 01 basics, 02 templates, 03 signals and special values, 04 ISO-TP for UDS and OBD-II, 05 J1939 TP.
X_TP.ecid	Each body (UDS_req/resp, KWP_req/resp, OBD2_req/resp, UDS_DID, UDS_OBD_F4) has a _TP version for reassembled ISO-TP messages: names and all signals – longer responses too (VIN, OBFCM, DTC 1–10, all OBD PIDs). Files without _TP name the single CAN messages SF/FF/CF/FC.

13. Common mistakes and tips

- **General rule before a specific one** – the specific rule never wins. Fix: always put `*` and „other“ at the end of a block.
- `100` **instead of** `x100` – without a prefix it is decimal 100 (0x64).
- **Offset sign as in DBC** – in ECID the offset is subtracted.
- **Motorola start bit as in DBC (MSB)** – in ECID it is the LSB.
- **A comma in a signal name or list text** – commas separate items. In a rule name `#` (the last item) a comma is fine.
- **Value list after the signal** – the list must be defined first. Put lists at the top of the main file.
- `--` **in an included file** – it wipes the calling file's template (ID).
- **Mask without a value** – `m0:xF0` alone checks nothing; add the value `0:x20` as well.
- **Signals in First/Consecutive Frames** – data is not reassembled; decoding only makes sense in Single Frames.
- **Byte range on a rule with signals** – the EYE then does not check the byte. Use an exact value or a mask for the signal.

14. Converting DBC to ECID (dbc2ecid)

dbc2ecid.exe is a Windows command-line program that converts a Vector DBC file to ECID 1.5. It needs no installation or additional libraries. The result is an ECID file (cp1250, CRLF, ^ECID:1.5) and a report listing what was not converted.

14.1 Usage

```
dbc2ecid input.dbc [output.ecid] [/j1939] [/prefix] [/en] [/report:file.txt]

dbc2ecid engine.dbc
dbc2ecid C:\DBC\truck.dbc C:\ECID\truck.ecid /j1939 /report:truck.txt
```

- output.ecid – if omitted, the input name with the .ecid extension is used.

- /j1939 – all 29-bit messages are converted as J1939 (PGN mask). Without the switch only messages with VFrameFormat = J1939PG, or all with ProtocolType = J1939.
- /prefix – signal name becomes Message.Signal (otherwise just Signal). Useful when signal names repeat across messages.
- /en – ECID comments and the report in English.
- /report:file – the report is also saved to a file.
- Exit code: 0 = OK, 1 = missing parameters, 2 = input cannot be opened, 4 = output cannot be written.

14.2 What is converted

- Messages BO_ → rule >I:x…,[E],#name. Message comment (CM_ BO_) → comment ; above the rule. The long name (SystemMessageLongSymbol) takes precedence.
- J1939 → PGN mask: PDU2 (PF ≥ F0h) M:x3FFFF00, PDU1 M:x3FF0000 (no priority, SA or DA). A J1939 message longer than 8 B gets T:J1939 (TP assembly in the Data view).
- Signals SG_ → @U / @S: the Motorola start bit is converted from MSB (DBC) to LSB (ECID), the offset sign is flipped (DBC –40 → ECID 40). SIG_VALTYPE_ float/double → @F:…:32 / 64. A byte-aligned (Intel) signal longer than 64 bits → text @T (e.g. VIN).
- Value tables VAL_ → lists [VT_Message_Signal]; identical tables are written only once.
- Signal comment (CM_ SG_) → ?description lines (a multi-line comment = several ? lines).
- Multiplex (M / mN) → rule with a condition on the multiplexor bits (value and mask, any bits in DB0–DB7) and the name "Message [Mux=N]". Extended multiplexing (SG_MUL_VAL_, nested multiplexors) → condition on all multiplexors in the path, ranges are expanded (max. 64 values). Non-multiplexed signals are attached to the generic rule of the message, placed after the rules with a condition.
- Encoding: UTF-8 input is converted to cp1250. A comma in a signal name, unit or table text is replaced by a semicolon (a comma separates ECID items) – listed in the report.

14.3 What is not converted (report)

- signal min/max, nodes (BU_, transmitter/receivers), attributes (GenMsgCycleTime, GenSigStartValue, …), signal groups (SIG_GROUP_), environment variables (EV_), standalone VAL_TABLE_.
- A multiplexor beyond DB7 (ECID conditions cover DB0–DB7 only), a multiplex with more than 64 values and a signal longer than 64 bits that is not byte aligned – the signal is converted without a condition or skipped and listed in the report.
- Signals without a message (VECTOR__INDEPENDENT_SIG_MSG) are skipped.

14.4 Example output

```
; ---------- Mux (11 bit x200, DLC 8) ----------
>I:x200,0:x01,#Mux [Sel=1]
@U:8:16:L:0.1:0,*V,#A
?Voltage when Sel=1
>I:x200,0:x02,#Mux [Sel=2]
@S:8:16:L:1:0,*A,#B
>I:x200,#Mux
@U:0:8:L:1:0,$VT_Mux_Sel,#Sel
```

14.5 Recommended workflow and building

- Convert the DBC, read the report and open the ECID in the program. Load errors are in <file>.ecid.log (chapter 11).
- The generated file is ordinary ECID – it can be edited further (templates, own names, lists, ?descriptions) or included (&) in your own main file.
- The conversion is one-way; after changing the DBC regenerate the file (keep your own changes in a separate file that includes the generated one).

- Building from source: Visual Studio (Developer Command Prompt) cl /EHsc /O2 /std:c++17 dbc2ecid.cpp, or MinGW g++ -O2 -std=c++17 -static -o dbc2ecid.exe dbc2ecid.cpp.

Appendix A – quick reference

```
;  comment
>  rule         >I:x123,E,M:x1FFFFF00,0:x01,m1:xF0,1:x20,S:0:4,#Name
%  template     %I:x7E8,M:x7F8,#Prefix
+  push+templ.  +1:x22,#RDBI
{  push         }  pop          -  pop          --  reset all
!  reset        !N !S !L !I !M !m !0..!7
=  templ. name  =Fixed name
[  value list   [Name  /  =0:Off,=1:On,=x80..xFF:Range  /  ]
@  signal       @U:bit:length:L|B:scale:offset,*unit,$list,#Name
:  templ. sig.  :H:16:8:L,#Name
!  special val. @U:0:8:L:1:0,!xFF:N/A,!xF0..xFE:error,!J
&  include      &file.ecid
^  version      ^ECID:1.5
?  description  ?Longer description of the previous signal

Items:    I:id  I:*  R:from:to  L:a:b:c  M:mask  M:*  E
          N:val  N:from:to  N:*  mN:mask  mN:*  S:bit:length[:B]  #name
          T:ISOTP  T:ISOTPX  T:J1939  T:*
Types:    U S H F B T J P O D   (T:*:n = field of *-delimited text)      value = raw * scale -
offset
Bits:     bit = byte*8 + bit_in_byte, always the LSB of the signal
```