PSAD

A.D.C. GmbH

PDO SW Manual

Version 2.15

1 Introduction

During the course of development of ECU-s various measurement systems are used by the supplier as well as the customer for fine-tuning (calibration) and verification (measurement, locating bugs). For this the ECU provides some interface through which the required data is provided. Usually this interface can be considered a raw byte stream source, thus in order to decode, and assign a meaning to the data, a measurement data description is necessary. Traditionally this measurement description file is created and maintained independently of all other project development. This of course makes it error prone in the sense that consistency between the ECU software and the description is not assured. The main goal of extracting measurement meta-data from source code is, to apply the single source principle: if you have a project, and need to modify it in some form, you only have to have to modify a single point. This practically alleviates the need to maintain several different documents in parallel, thus simplifies maintenance.

The PDO-Toolset was designed for automatic build-process based generation of measurement descriptions. It consists of two basic parts: pdo_scan.exe and pdo_tool.exe.

Pdo_scan.exe uses direct or pre-processed C source code as input, from which it generates an intermediate XML description. Multiple such XML files can serve as input to be 'linked' into a large XML description or transformed into ASAP A2L or SDL 2.0 format descriptions by pdo_tool. The recommended use of the tools is depicted in Figure 1 PDO basic operation.

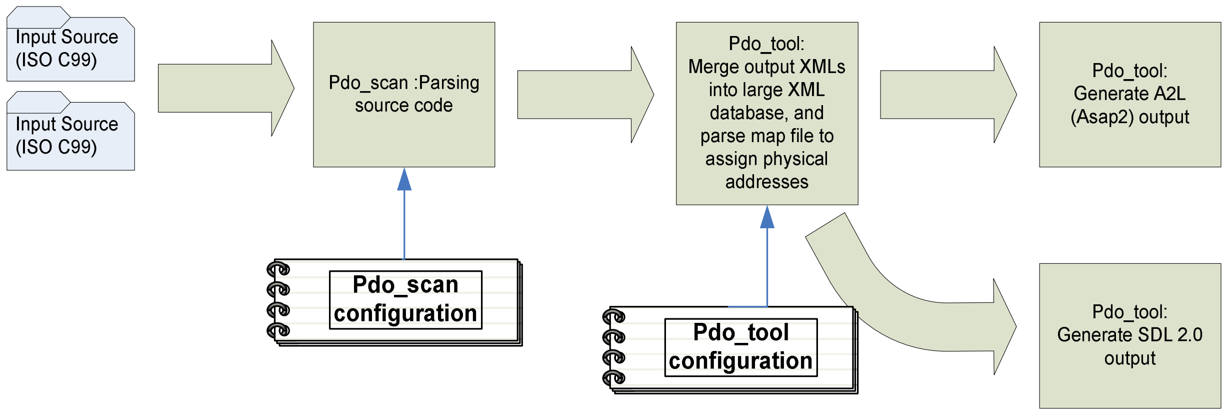

Input Source (ISO C99)
Input Source (ISO C99)
Pdo_scan :Parsing source code
Pdo_tool: Merge output XMLs into large XML database, and parse map file to assign physical addresses
Pdo_tool: Generate A2L (Asap2) output
Pdo_tool: Generate SDL 2.0 output
Pdo_scan configuration
Pdo_tool configuration

Figure 1 PDO basic operation

The role of pdo_scan.exe is to parse C source code and generate intermediate XML databases with the parsed information. Pdo_tool in turn operates on the XML databases: it can "link" (or merge) such XML databases together with map file information into XML databases with address information or generate ASAP2 or SDL2.0 output based on the XML database input read.

The behaviour of both tools can be configured over the command line or over special configuration files. Note that the two executables support different options and have a different syntax for their configuration files. Pdo_scan.exe's configuration file / command line options are for specifying how to parse the input source code, how to mimic the real target compilers behaviour (type allocation, keyword use, etc.).

Pdo_tool's configuration file / command line options are for specifying how to generate output.

This document is divided into several chapters with different target audiences. If you are a developer who writes source code that is parsed by PDO you are recommended to read chapters 2.1, 2.2 and 2.8 which deal with source-code level issues. Maintainers of build/make processes are advised to read chapters 2.3, 2.3, 2.5, 2.6, 2.7, 2.8, 3.1 and 3.2, which deal with integration/configuration of the tool for different architectures.

2 PDO_SCAN

The program pdo_scan.exe is responsible for parsing ISO-IEC 9899:1999 compatible C source code[1]. The input files to the parser may be either pre-processed using an external pre-processor or given directly to pdo_scan.exe for internal pre-processing. The examples in the following chapters assume that the internal (built-in) pre-processor is used. If an external pre-processor is used, and the pre-processed output is then used as input to pdo_scan.exe, then all the define-based functionality will not be available, as the corresponding defines are removed by the pre-processor.
The parser itself parses all global declarations but skips function definitions, thus function local variables are not parsed/lexed[2]. To correctly parse global declarations, as well as mimic the behavior of the target compiler as closely as possible a configuration file needs to be set up as described in chapter 2.6 Configuration file. The output of pdo_scan.exe is an XML file describing the global declarations parsed, which in turn can be used as input for pdo_tool.exe to create A2L, SDL or merged XML output. This output file is described in chapter 2.9 XML Output.

[1] Note: C++ is not supported. The list of reasons for this is long, foremost the languages complexity (templates, namespaces, non-standardized name mangling, non-standardized memory layout etc.).
[2] Experimental parsing of function definitions can be enabled, but due to scoping issues it is currently incomplete, thus not officially supported

2.1 Not commented source code

The tool can extract information from normal C source code without the optional special comments described in chapter 2.2. Without the comments the type information provided in the C declaration is used to derive information such as value range, variable/parameter names etc. A simple example is provided in Figure 2 Uncommented input source code example.

```
typedef unsigned char ui8_t;
typedef unsigned short ui16_t;
typedef unsigned int ui32_t;

typedef enum SomeEnum {
    ZERO,
    ONE,
    TWO,
    FOUR=4
} SomeEnum_t;

typedef struct {
    ui16_t Distance;
    ui16_t DetectionTime;
    SomeEnum_t Scans;
    struct {
        ui16_t Distance;
        ui16_t RCS;
    } PrevScan;
} MyStruct_t;

static MyStruct_t MyInst1, MyInst2;
```

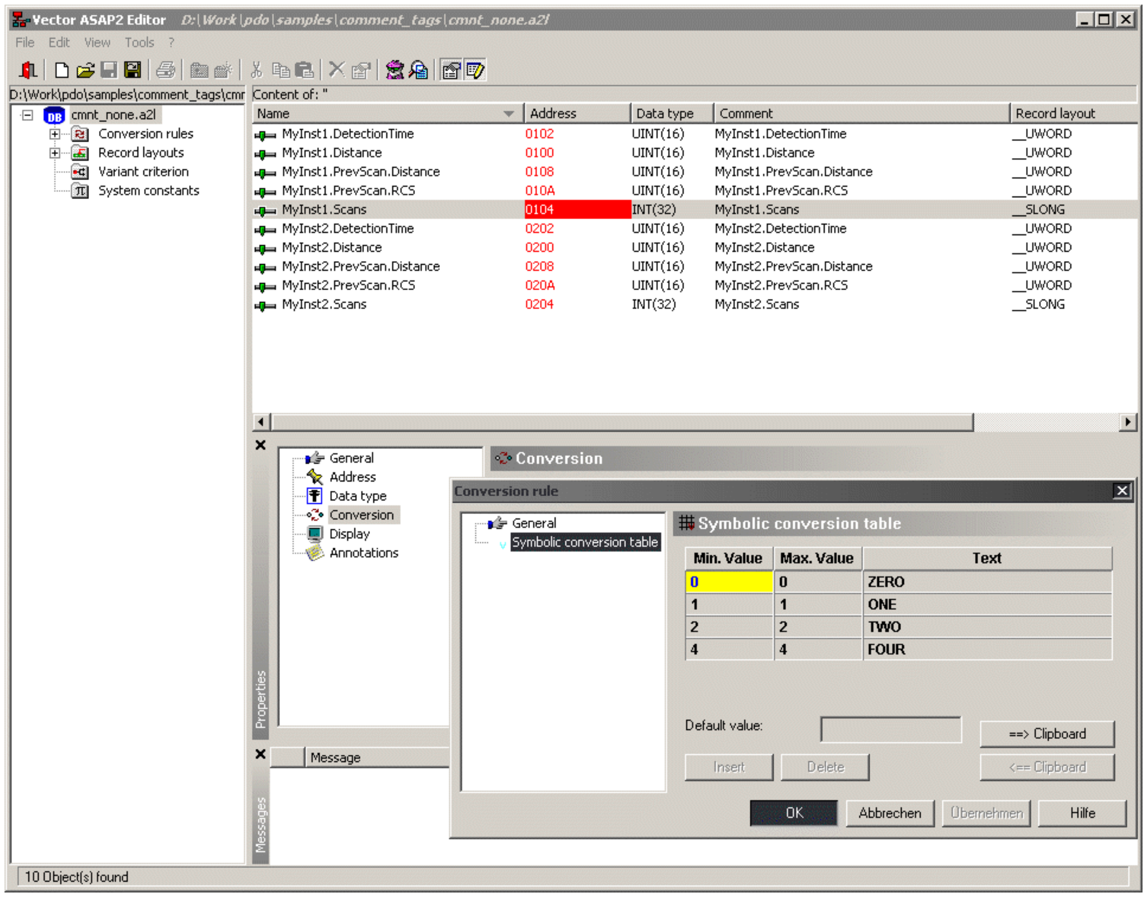
Figure 2 Uncommented input source code example

Based on this input pdo_scan & pdo_tool generated example output is illustrated in Figure 3 Raw generated A2L from uncommented example source code and Figure 3 Asap editor screenshot of uncommented example A2L. As the examples illustrate all variables can be added to the generated ASAP A2L file[3], with automatic generation of textual lookup tables for enumerations and proper minimum/maximum value specifications based on the used types. Without additional meta-data from comments, the functionality is similar to that provided by debuggers: information from the type tree can be displayed.

[3] Provided that the compiler/linker assigned addresses are available in the map file. All variables without address information will not be generated into output

```
/begin PROJECT test " ASAP2-File generated by PDO-Tool  Revision: 2.5 RC1   - All rights reserved - Property of
Continental A.G - A.D.C GmbH Lindau"

  /begin MODULE CPP ""

    /begin MOD_COMMON ""
      BYTE_ORDER MSB_FIRST
   /end MOD_COMMON

            /begin COMPU_METHOD CONV_FUN.Identity "special built-in identity conversion function"
                        RAT_FUNC "%.8" ""
                        COEFFS 0 1 0 0 0 1
            /end COMPU_METHOD

            /begin COMPU_METHOD CONV_FUN.enum_SomeEnum "PDO_AutoGen for MyInst1.Scans"
                        TAB_VERB "%0.8" ""
                        COMPU_TAB_REF CONV_FUN.enum_SomeEnum
            /end COMPU_METHOD
            /begin COMPU_VTAB CONV_FUN.enum_SomeEnum "PDO_AutoGen for MyInst1.Scans"
                        TAB_VERB 4
                        0 "ZERO"
                        1 "ONE"
                        2 "TWO"
                        4 "FOUR"
            /end COMPU_VTAB
            /begin CHARACTERISTIC MyInst1.DetectionTime "MyInst1.DetectionTime"
                        VALUE 0x102 __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst1.Distance "MyInst1.Distance"
                        VALUE 0x100 __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst1.PrevScan.Distance "MyInst1.PrevScan.Distance"
                        VALUE 0x108 __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst1.PrevScan.RCS "MyInst1.PrevScan.RCS"
                        VALUE 0x10A __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst1.Scans "MyInst1.Scans"
                        VALUE 0x104 __SLONG 0 CONV_FUN.enum_SomeEnum 0 4
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst2.DetectionTime "MyInst2.DetectionTime"
                        VALUE 0x202 __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst2.Distance "MyInst2.Distance"
                        VALUE 0x200 __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst2.PrevScan.Distance "MyInst2.PrevScan.Distance"
                        VALUE 0x208 __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst2.PrevScan.RCS "MyInst2.PrevScan.RCS"
                        VALUE 0x20A __UWORD 0 CONV_FUN.Identity 0 65535
            /end CHARACTERISTIC
            /begin CHARACTERISTIC MyInst2.Scans "MyInst2.Scans"
                        VALUE 0x204 __SLONG 0 CONV_FUN.enum_SomeEnum 0 4
            /end CHARACTERISTIC
            /begin RECORD_LAYOUT __SLONG
                        FNC_VALUES 1 SLONG ROW_DIR DIRECT
            /end RECORD_LAYOUT
            /begin RECORD_LAYOUT __UWORD
                        FNC_VALUES 1 UWORD ROW_DIR DIRECT
            /end RECORD_LAYOUT
/end MODULE
/end PROJECT
```

Figure 3 Raw generated A2L from uncommented example source code



Name	Address	Data type	Comment	Record layout
MyInst1.DetectionTime	0102	UINT(16)	MyInst1.DetectionTime	__UWORD
MyInst1.Distance	0100	UINT(16)	MyInst1.Distance	__UWORD
MyInst1.PrevScan.Distance	0108	UINT(16)	MyInst1.PrevScan.Distance	__UWORD
MyInst1.PrevScan.RCS	010A	UINT(16)	MyInst1.PrevScan.RCS	__UWORD
MyInst1.Scans	0104	INT(32)	MyInst1.Scans	__SLONG
MyInst2.DetectionTime	0202	UINT(16)	MyInst2.DetectionTime	__UWORD
MyInst2.Distance	0200	UINT(16)	MyInst2.Distance	__UWORD
MyInst2.PrevScan.Distance	0208	UINT(16)	MyInst2.PrevScan.Distance	__UWORD
MyInst2.PrevScan.RCS	020A	UINT(16)	MyInst2.PrevScan.RCS	__UWORD
MyInst2.Scans	0204	INT(32)	MyInst2.Scans	__SLONG

Symbolic conversion table

Min. Value	Max. Value	Text
0	0	ZERO
1	1	ONE
2	2	TWO
4	4	FOUR

Figure 3 Asap editor screenshot of uncommented example A2L

2.2 Supported comments

Using specially formatted comments the generated output can be improved via specifying additional information not available to the C-compiler such as symbolic unit names, scaling factors/offsets etc. Note that specifying additional information is not necessarily required, and derived types automatically inherit attributes over the type chain. This inheritance is illustrated in Figure 4 Comment tag inheritance over type chain, where the variable 'EgoSpeed' inherits the scaling factor specified for the type 'Speed_t', since 'EgoSpeed' is of type 'Speed_t'. If in this example 'EgoSpeed' had an own scaling factor specified, that would overrule the one specified for type 'Speed_t'. This overruling (i.e.: respecifying a the comment tag for a derived type) is generally the rule, with the exception of the Allows and CycleId tags, which exhibit additive behavior (as they describe lists, not singular values).

```
typedef unsigned short Speed_t; /*!< @resolution: 1e-2 @unit:m/s */

Speed_t EgoSpeed;   /*!< Ego speed inherits from Speed_t */
```

Figure 4 Comment tag inheritance over type chain

Depending on the command-line and configuration options pdo_scan.exe supports two different comment styles: doxygen-style comments and Teves-style comments.

2.2.1 Doxygen style comments

Doxygen-style comments have two forms, depending on whether the comment is before the object is applies to, or after it. The below Figure 5 Doxygen-style comment binding illustrates both forms, showing the comments standing before the object they apply to have to begin with "/*!", while comments following the object they apply to have to begin with "/*!<"[4]. There are no requirements for the termination of doxygen-style comments, simply use the normal C comment termination "*/".

```
/*! Comment describing textually TPObject */
struct {
    f32_t fX;   /*!< Comment text describing fX @min: -5 @max: 200 */
    f32_t fY;   /*!< Comment text describing fY */
} TPObject;

/*! Comment binding to 'Time' @unknown: pdo_scan.exe does not care */
int Time;

int Cycles; /*!< Comment binding to 'Cycles' */
```

Figure 5 Doxygen-style comment binding

[4] Note: no white-spaces, escaped new-lines or other characters are allowed between the three (/*!) respectively four (/*!<) characters. Thus for example /***!< will not be accepted as the beginning of a doxygen-style comment, and will be treated as a normal C comment.

Within doxygen style-comments specific attributes are tagged using ‘@’ followed by identifier-like keywords and terminated by a colon[5]. In the example ‘TPObject.fX’ illustrates the use of the ‘min’ and ‘max’ tags. The tag names themselves are case insensitive. Whenever pdo_scan.exe encounters an attribute tag within a doxygen-style comment, it checks whether the tag is supported. Unsupported tags, along with any lexical elements following them are ignored[6], illustrated in the example by the tag unknown in the comment for ‘Time’.
If the read tag is supported by pdo_scan.exe, then the format of the lexical elements following it has to correspond to the rules specified for that tag.

2.2.2 Smart comments

Starting from pdo_scan.exe version 2.8 support for so-called smart comments has been implemented. These comments do not have to have the doxygen-style exclamation mark at the beginning, but have to otherwise adhere to the doxygen keyword style, meaning the same syntax and format keywords as for doxygen comments are supported (e.g.: *@max: 15*). The binding (i.e.: which object a comment belongs to) is determined based on relative position of the comment. If it is in the same line after a declaration, the comment is bound to the declaration object. Otherwise it is assumed that it belongs to the object following the comment.
This functionality basically allows normal (undecorated) C comments to be automatically assigned to declarations as descriptions.

2.2.3 Teves-style comments

Support for Teves-style comments is no longer actively developed, and merely maintained for compatibility purposes with 1.x versions of PDO-Toolset. Teves-style comments have to begin with “/*@” and should be terminated by “@*/”[7], and always follow the object that they bind to.

```
struct {
    f32_t fX;   /*@ Comment text describing fX MIN: -5 MAX: 200 @*/
    f32_t fY;   /*@ Comment text describing fY @*/
} TPObject;     /*@ Comment describing textually TPObject @*/

int Time;   /*@ Comment binding to 'Time' @*/

int Cycles; /*@ Comment binding to 'Cycles'@*/
```

Figure 6 Teves-style comments
Within Teves-style comments specific attributes are tagged using a case sensitive identifier-like keyword followed by a colon. The set of supported tags in Teves-style comments is summarized in Table 1 Corresponding Teves and Doxygen-style tags

[5] Actually the colon is not required, whitespaces are also accepted. However it is always recommended to insert a colon after the @tag.
[6] This allows arbitrary extension of the comments for other purposes, such as automatically generated formatted doxygen code documentation
[7] The normal C comment termination ‘*/’ is also accepted

with their corresponding doxygen-style tags, which are documented in the following chapters. The same syntax rules apply to the contents following the corresponding tags, regardless of within a Teves-style or Doxygen-style comment.

Use	Teves-style tag	Doxygen-style tag
General description (text)	REM:	@description:
Name redefinition	NAME:	@name:
Unit specification	UNIT:	@unit:
Scaling factor specification	LSB:	@resolution:
Offset specification	OFFSET:	@offset:
Minimum value	MIN:	@min:
Maximum value	MAX:	@max:

Table 1 Corresponding Teves and Doxygen-style tags

2.2.4 Comments and macros

If the internal pre-processor is used, then normal C macros can be used within the comments as well, if they expand to the required type for the given comment tag. The rules for expansion in such cases are the same as for C: the macro has to be declared previously (thus there is no late binding). The Figure 7 Bad and good example of macros in comments illustrates this point: the comment for *Torque* is invalid, since at comment lexing time, the macro *TORQ_FACTOR* is not yet known to the pre-processor. The comment for *Distance* however is valid, since the used macro *DIST_FACTOR* has been declared previously.

```
unsigned int Torque;    /*!< @resolution: TORQ_FACTOR */
#define TORQ_FACTOR     1e-3f

#define DIST_FACTOR 0.02
unsigned int Distance;  /*!< @resolution: DIST_FACTOR */
```

Figure 7 Bad and good example of macros in comments

2.2.5 Description

Descriptions are general textual (free-form) descriptions of the given object (variable/field). Upon entering a supported comment pdo_scan.exe assumes at default that a description is being provided (implicit description), until reading the first comment tag specifying a different comment portion.

```
unsigned int MyVar;     /*!< This is an implicit description */
float MyVar2;           /*!< @description: This is an explicit
description */

/*! An implicit description */
double MyVar3;

unsigned int MyVarT;    /*@ This is a Teves-style implicit
description */
float MyVar2T;          /*@ REM: This is a Teves-style explicit
description */
```

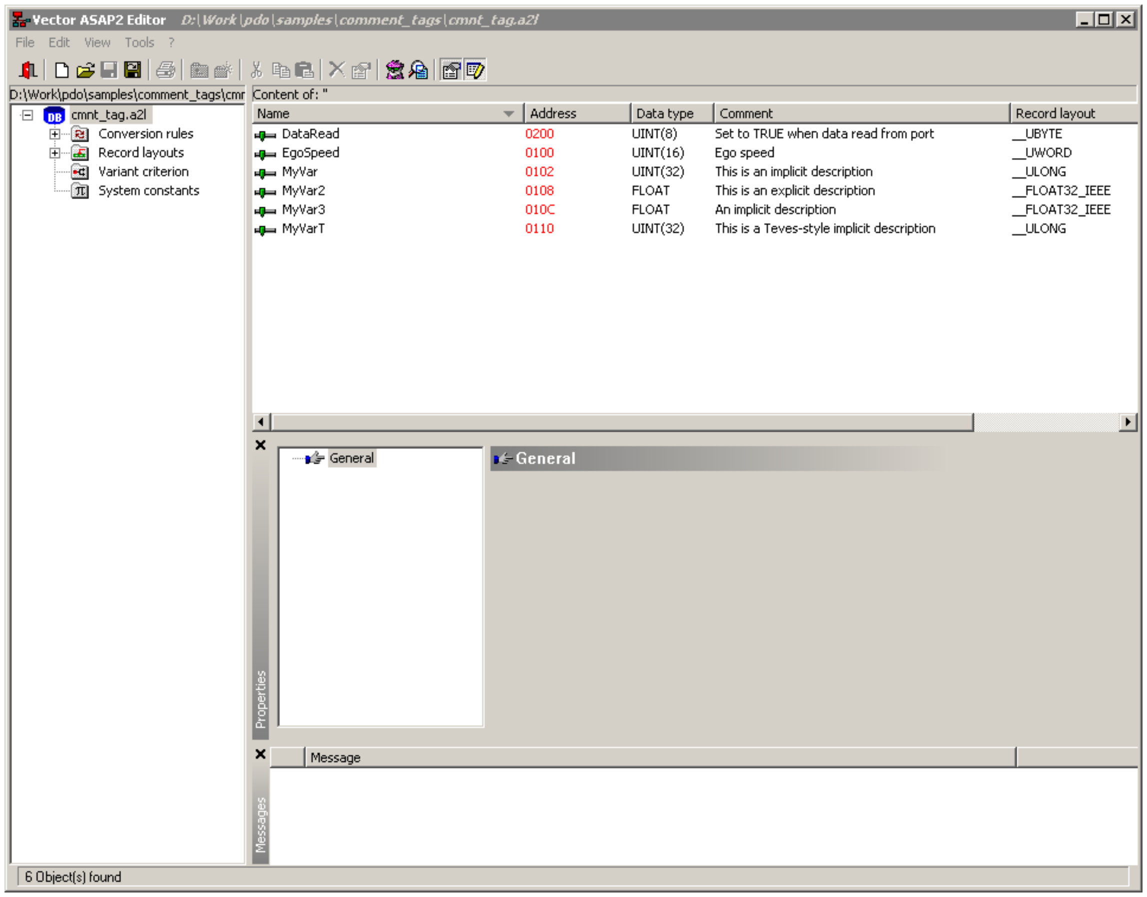
Figure 8 Description examples

There are no syntax rules for descriptions, beyond that no tag-like portions shall be contained within the text, as that will start a different comment portion. Also note that no macro replacement is done for descriptions – this is on purpose to avoid replacing textual references to defines with their values. The example Figure 9 Description macro expansion inhibition example highlights this: most likely the user would want the description as in the text, instead to the expanded “Set to (bool_t)1 when data read from port”.

```
#define TRUE    (bool_t)1
#define FALSE   (bool_t)0

typedef unsigned char bool_t;

static bool_t DataRead; /*!< Set to TRUE when data read from port */
```

Figure 9 Description macro expansion inhibition example

These description fields are used for generating the ASAP A2L Measurement and characteristic object’s long identifier (comment, description) field. An example of this can be seen on the Figure 10 Vector ASAP2 Editor Screenshot of objects with descriptions with the PDO generated A2L for the examples in this chapter.



Name	Address	Data type	Comment	Record layout
DataRead	0200	UINT(8)	Set to TRUE when data read from port	__UBYTE
EgoSpeed	0100	UINT(16)	Ego speed	__UWORD
MyVar	0102	UINT(32)	This is an implicit description	__ULONG
MyVar2	0108	FLOAT	This is an explicit description	__FLOAT32_IEEE
MyVar3	010C	FLOAT	An implicit description	__FLOAT32_IEEE
MyVarT	0110	UINT(32)	This is a Teves-style implicit description	__ULONG

Figure 10 Vector ASAP2 Editor Screenshot of objects with descriptions

When lexing descriptions sequences of whitespaces within the description are replaced by a single space character, allowing multiple-line descriptions when necessary[8].

[8] For the same reason ASCII-art in description fields is strongly discouraged

2.2.6 Name

Name tags have to be followed by a single identifier-like[9] token. This feature can be used to assign a different name to an object, than the C identifier used for it. As using different names in source code and measurement systems can be very confusing, use of this feature should be limited to where necessary (for example for backwards compatibility with previously manually generated A2Ls/SDLs).

2.2.7 Resolution

In spite of what the name might suggest, this is the scaling factor to be applied to the given scalar variable and not the accuracy. Together with the Offset tag this can be used to provide scaling/offset for converting internally used values to physical units. A numeric value, or an expression[10] evaluating to a constant numeric value is expected after the resolution tag. Note that numeric values specified have to be C compatible, notably decimal comas are not supported.

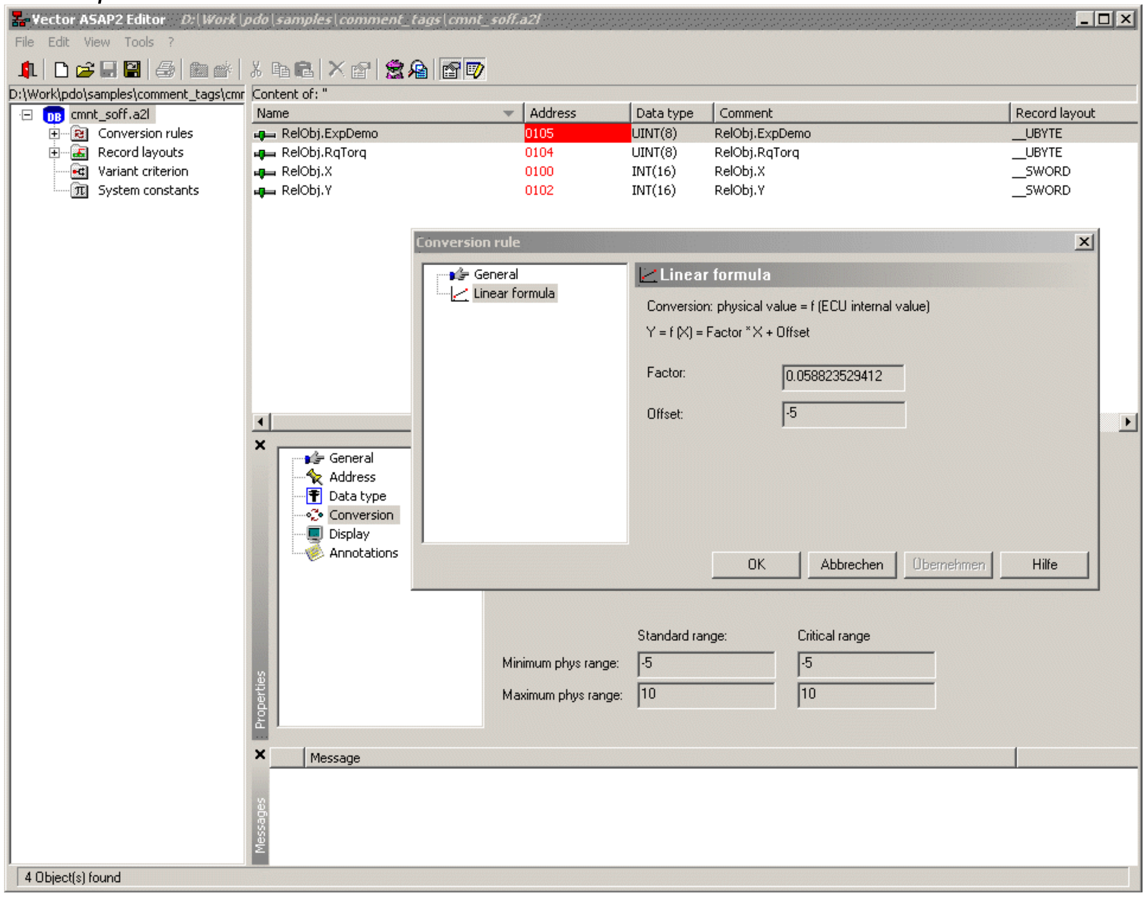
```
#define UCHAR_MAX   255
#define MAX1    10.0
#define MIN1    -5.0
#define FAC1    ((MAX1 - MIN1)/UCHAR_MAX)
#define OFF1    MIN1

typedef signed short int Pos_t; /*!< @resolution:0.01 @unit:m */
typedef unsigned char ReqTorque_t; /*!< @resolution:4 @offset: -100
@unit:NM */
typedef unsigned char ui8_t;

struct Object {
    Pos_t X, Y;
    ReqTorque_t RqTorq;
    ui8_t ExpDemo;  /*!< @offset: OFF1 @resolution:FAC1 */
    ui8_t RoundExp; /*!< @offset: -1e2 @resolution: 100/255 */
};

static struct Object RelObj;  /*!< Relevant object */
```

Figure 11 Example for resolution and offset

Based on source code in Figure 11 Example for resolution and offset a screenshot of the Vector ASAP editor with the generated A2L's conversion function for *RelObj.ExpDemo* is shown in Figure 12 Scaling factor/offset in ASAP2 editor. The resolution specified for the field is FAC1, which is macro that expands to ((10.0 – (-5.0))/255) = 0.058823. As can be seen on the screenshot, this expression was evaluated, and the correct scaling factor has been specified in the conversion function. Note that expression evaluation follows the C syntax rules: that is integer/double propagation is done and arithmetic operations are done on the

[9] Identifier-like tokens for name tags are a slightly broader class than C identifiers: they have to begin with a letter (upper or lower case) or underscore and may be followed by alphanumeric (letters, numbers) characters or underscores or the '%' sign for special purposes.
[10] Note: Expressions in all comments are evaluated according to C syntax rules, more on that later

corresponding propagated type. This means that the resolution specified for the *RoundExp* field is invalid, since the constant expression 100/255 evaluates to zero, since both operands are integers, thus an integer division is done. Obviously specifying a scaling factor of zero is an error and leads to an error message like "*cmnt_soff.c:15: error: Specifying a factor of zero is invalid! Ignoring scaling factor specified!*".



Name	Address	Data type	Comment	Record layout
RelObj.ExpDemo	0105	UINT(8)	RelObj.ExpDemo	__UBYTE
RelObj.RqTorq	0104	UINT(8)	RelObj.RqTorq	__UBYTE
RelObj.X	0100	INT(16)	RelObj.X	__SWORD
RelObj.Y	0102	INT(16)	RelObj.Y	__SWORD

Linear formula

Conversion: physical value = f (ECU internal value)

Y = f (X) = Factor * X + Offset

Factor: 0.058823529412

Offset: -5

	Standard range:	Critical range
Minimum phys range:	-5	-5
Maximum phys range:	10	10

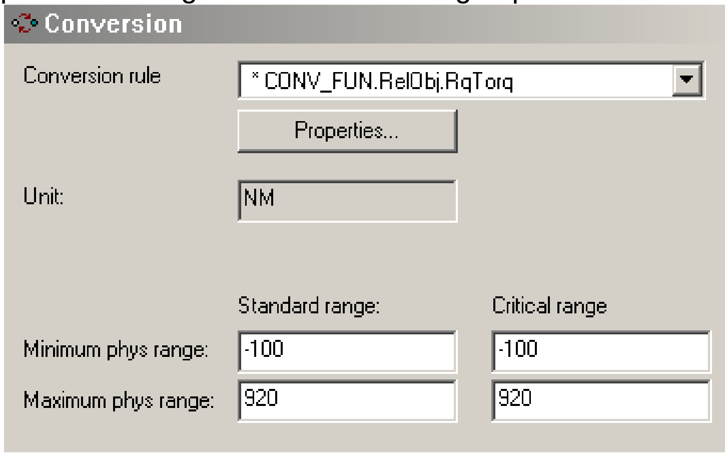
Figure 12 Scaling factor/offset in ASAP2 editor

2.2.8 Offset

The offset tag can be used to specify an offset for the raw value to physical value conversion. For the example code of Figure 11 Example for resolution and offset the offset specified for the field *ExpDemo* uses the macro OFF1, which in turn is a redefine of MIN1, which has a value of -5. As can be seen on the screenshot Figure 12 Scaling factor/offset in ASAP2 editor, the generated conversion function uses this offset of -5. Using the type information and based on the resolution and offset specified for *ExpDemo* the minimum and maximum values are automatically calculated (knowing that unsigned chars have a raw value range of [0 ..255], leads to [(0 * FAC1 + OFF1) .. (255 * FAC1 + OFF1)] = [-5 .. 10]). As can also be seen on the screenshot the raw value (X) to physical value (Y) conversion is done via the formula: Y = Resolution * X + Offset.

2.2.9 Unit

The unit tag can be used to specify a textual physical unit description. Canape for example displays this unit value in numerical tables. A screenshot of the conversion function for the field *RqTorq* from Figure 11 Example for resolution and offset is provided in Figure 13 Unit and range specification for *RqTorq*.



Conversion

Conversion rule: * CONV_FUN.RelObj.RqTorq

Unit: NM

	Standard range:	Critical range
Minimum phys range:	-100	-100
Maximum phys range:	920	920

Figure 13 Unit and range specification for *RqTorq*

One typical problem with units is that in certain cases physical unit symbols require special characters[11] such as the ° or µ sign. This problem is exacerbated by the fact that certain poorly designed file formats (such as ASAP2) do not even consider character encoding problems or provide mechanisms to specify the encoding of files. Due to this careful evaluation of the tools/operating systems/locales used is recommended prior to using non alphanumeric characters, not due to any issues within PDO (it supports multiple encodings), but for example due to missing documentation on how third party tools handle A2L files encoded in different character sets. At least for Vector Canape on locale=DE computers, UTF-8 A2Ls don't seem to be supported, on the other hand a couple of special symbols tested in ISO 8859-1 encoded files were displayed properly.

2.2.10 Min/Max

The min/max tag can be used to specify the minimum respectively maximum allowed physical value for a signal. The tag shall be followed by a constant expression or an explicit numeric value. If an explicit minimum/maximum is not specified for a given variable, then it is automatically calculated based on the range of the C type, optional resolution and offset values specified, demonstrated in Figure 13 Unit and range specification for *RqTorq*. For enumerations the automatic minimum/maximum value corresponds to the smallest/greatest enumerator specified. For an explicit minimum/maximum specification to make sense the explicit minimum has to be greater equal the otherwise automatically calculated minimum value, while the explicit maximum has to be less or equal the automatically calculated minimum value, since use of the complete possible value range would lead to the automatic

[11] Pdo_scan supports some input character encodings as well as UTF-8 input. For details see xyz

min-max value[12]. Since this mechanism can be used to limit the valid range of values allowed, it is especially useful for calibration parameters[13].

[12] Currently no diagnostic is implemented in PDO, thus no messages will be produced by specifying too wide ranges that can not be represented by the type
[13] Since imposing a minimum maximum value limit makes extra checks in ECU SW unnecessary

2.2.11 Values

The values tag can be used to specify alternative values representations for display in measurement/calibration applications. It's goal is to allow easy access to relevant information for cases where the C declaration(s) of the given type do not represent the 'real' data layout/type[14]. It has three different forms:

- An enumeration typedef or tag-name reference
- A coma separated list of integer or integer constant expression defines
- Or a layout declaration

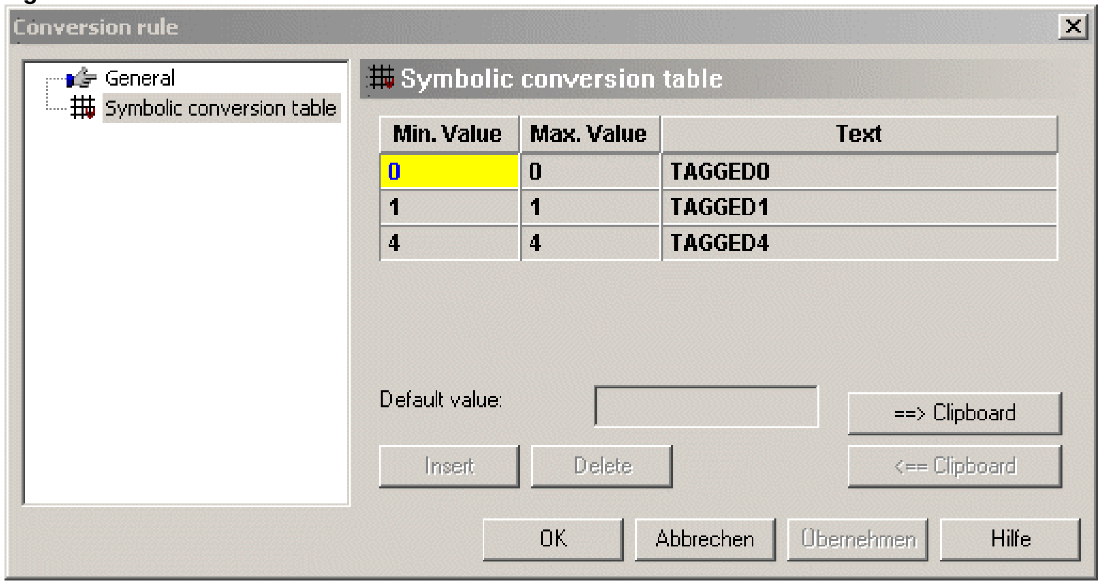
Basic examples of these three forms are provided in Figure 14 Basic forms of comment values declaration below. The unsigned byte variable "Demo1" has a values declaration referencing the tagged enum *EnumByTag*, which leads to an A2L conversion function as displayed in Figure 15 Tag name enumeration use conversion table. The unsigned byte *Demo2* is completely analogous, it merely uses a typedef name to reference an enumeration. In both cases the enumerators specified in the respective enums are displayed instead of the raw values[15].

[14] A good example of this is the case when for RAM considerations for a compiler allocating 'ints' for enumerations, instead of using enumerations other smaller integer types are used.

[15] Note: if description comments for enumerators are provided, then output A2Ls can be configured to use those instead of the enumerator names.

```
#define P_ONE   1
#define P_TWO   2
#define P_THREE (P_ONE + P_TWO)
#define P_FOUR  (P_TWO * 2)

typedef unsigned char ui8_t;
typedef unsigned int ui32_t;

enum EnumByTag {
    TAGGED0,
    TAGGED1,
    TAGGED4 = 4,
};

typedef enum {
    ENUM0,
    ENUM1,
    ENUM2
} EnumByTypedef_t;

static ui8_t Demo1;     /*!< Demo of tagged enumeration use @values:
enum EnumByTag */
static ui8_t Demo2;     /*!< Demo of typedef enumeration use @values:
EnumByTypedef_t */
static ui8_t Demo3;     /*!< Demo of define list use @values:
P_ONE,P_TWO,P_THREE,P_FOUR */
static ui8_t Demo4[16]; /*!< Demo of layout declaration @values:
               struct {
                   ui8_t x;
                   ui8_t y;
                   [7] = ui32_t Odometer : 24;
               }
           */
```

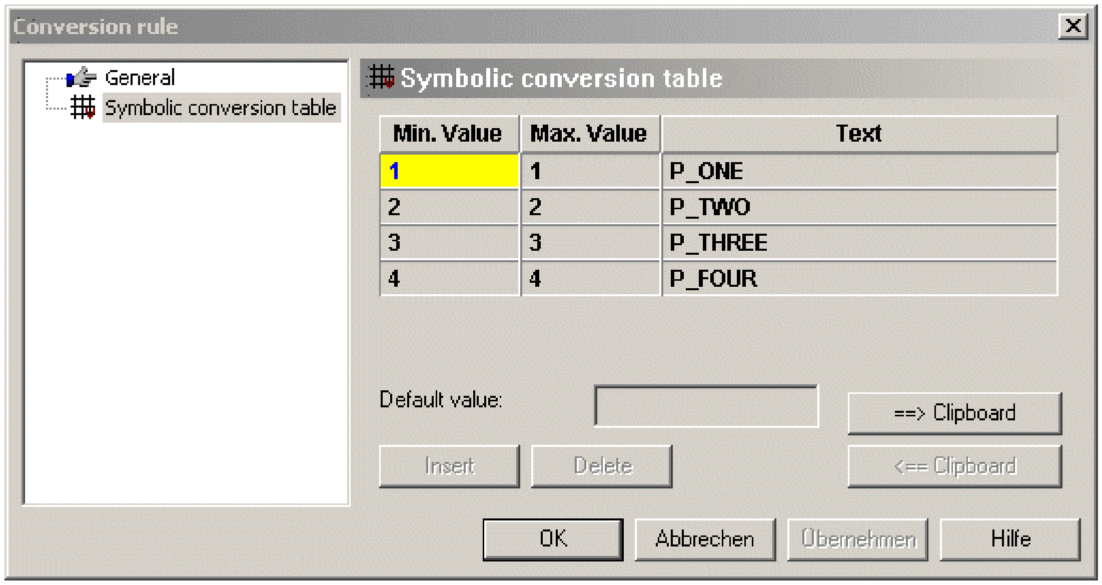
Figure 14 Basic forms of comment values declaration



Min. Value	Max. Value	Text
0	0	TAGGED0
1	1	TAGGED1
4	4	TAGGED4

Figure 15 Tag name enumeration use conversion table

The unsigned byte *Demo3* demonstrates the use of coma separated list of integer constants or constant expressions defines, and leads to the creation of a conversion table as illustrated in Figure 16 Coma separated define list conversion table example. In this case the names of the defines are displayed (mapped) to the corresponding values that they expand to.



Min. Value	Max. Value	Text
1	1	P_ONE
2	2	P_TWO
3	3	P_THREE
4	4	P_FOUR

Figure 16 Coma separated define list conversion table example

The third form of values declarations, using layout specifications is more complex, and a simple example is provided for the variable *Demo4* in Figure 14 Basic forms of comment values declaration. It basically allows specification of arbitrary layout/fields for the object it binds to through an extended C-like struct or union declarations. Thus a layout declaration must always be a struct or a union declaration without tag name. In the example the C array of 10 bytes is declared to actually be a structure with three fields: *x*, *y* and *Odometer*. The offsets/bitmasks are calculated as for normal C struct or union declarations: thus *x* gets offset 0, *y* gets offset 1. The field *Odometer* shows use of the special designator feature, which is the extension compared to normal C declarations: it allows to artificial specifying offsets/fields/bit-masks for the alternative layout. In the example it specifies that starting from the normal C declaration's 7th element a 24 bit wide integer value *Odometer* shall be used. More detailed example for special designators and layout specification are provided in Figure 17 Special designator example below.

```
#define SEVERITY_MASK   0x01
#define CUR_IGN_MASK    0x0E
#define EEP_FLUSH_BIT   0x10

static struct {
    struct {
        ui8_t State;
        ui8_t Id;
    } ExtData;
    struct {
        struct {
            struct {
                ui8_t X;
                ui8_t Y;
            } Sub2NextEx;
        } SubNextEx;
    } NextEx;
    ui16_t Age;
    ui8_t Data[16];
} Demo5;/*!< Demo of special designators @values:
          struct {
            .ExtData.State = struct {
                .mask(SEVERITY_MASK) = ui8_t severity;
                .mask(CUR_IGN_MASK)  = ui8_t ign_cnt;
                .mask(EEP_FLUSH_BIT) = ui8_t eeprom_flush;
             } State;
             .NextEx = struct {
                .SubNextEx = struct {
                    .Sub2NextEx.X = ui8_t X;
                    .Sub2NextEx.Y = ui8_t Y;
                } SubExample;
             } Example;
             .ExtData.Id = ui8_t Id;
             .Data[0].mask(0x01) = ui8_t SomeBit;
             .Data[0].mask(0x02) = ui8_t SomeOther;
             .offset(0x10) = ui16_t OffsetDemo;
            }
        */
*/
```

Figure 17 Special designator example

As the example shows the special designators can be used ahead of normal C struct/union field declarations and are always followed by the equals sign. They are similar to initializer designators of ISO C99, with the addition of two new keywords: *mask* and *offset*. There are four forms:

- *.fieldname* to specify binding to the original C struct/union declaration
- *[index]* to specify binding to an element of the original C array declaration
- *.mask(value)* to explicitly specify a bit-mask
- *.offset(value)* to explicitly specify an offset relative to the current C parent

These special designators nest, as demonstrated by the sub-struct declarations within the *Example* field of the values struct: as the *Example* struct was already specially designated to bind to *NextEx*, within the struct we can use the special designator *.SubNextEx*, as the original C *NextEx* field had a sub-field *SubNextEx*. The special designators can also be chained, as demonstrated in the designator for *SomeBit*, which uses *.Data[0].mask(0x01)* as designator: the field *Data*'s element index *0* within the C declaration masked with the mask *0x01*. Note also that normal macro expansion is performed (if internal pre-processor is being used), allowing use of macros within the layout declaration.

2.2.12 Allows

The allows comment tag has to be followed by a coma separated list of identifiers or macros that expand to a coma separated list of identifiers. This expanded list of identifiers can be then provided to pdo_tool to filter[16] the output. An example is provided in Figure 18 Allows tag use example.

```
#define POS_ALLOW_CUSTOMER bmw,psa

typedef unsigned short ui16_t;

typedef struct {
    ui16_t X;     /*!< @allow: POS_ALLOW_CUSTOMER, ford */
    ui16_t Y;     /*!< @allow: POS_ALLOW_CUSTOMER */
    ui16_t Age;    /*!< @allow: ford */
} Object_t;        /*!< @allow: apia */

static Object_t Objs[20];
```

Figure 18 Allows tag use example

In this example for the fields *X* and *Y* the allow list specified expands to *bmw, psa, ford* and *bmw, psa* respectively. Since the variable *Objs* is derived from the type *Object_t* with mentioned fields, when generating output with pdo_tool, filtering with allowed for *bmw* all 20 instances of X and Y will be generated in the output, but the *Age* field will not. Similarly when generating output with *apia* allowed, all three fields of *Object_t* will be visible, as the complete structure has the allow tag bound to it. Note that allow tags maintain lists, thus specifying the allow tag in a derived type adds the identifiers specified to the list and does not replace the list inherited from the type derived from. Multiple identifiers can be provided when generating output with pdo_tool, and if any of the specified identifiers is specified for a given variable or it's type tree, then it is generated in the output.

The user is free to choose allow identifiers, but it is recommended to establish an allow identifier management strategy, as the matching/desired allow identifiers need to be specified when generating the output via pdo_tool.exe.

2.2.13 Vaddr

The vaddr tag has to be followed by a single or coma separated list of integers (addresses). Each of these addresses introduces a *virtual object instance*. The main differences between a *virtual object instance* and a *normal object instances* is that virtual object instances do not actually have to exist in the C program, their addresses are artificially specified by the vaddr tag. *Normal object instances* are normal C variable instances, which get their addresses from the target linkers map file. *Virtual object instances* can be introduced in comments bound to variables or

[16] There are other filters as well, for details see chapter 3

type declarations (which in C do not have instantiations). This feature can be used to create artificial objects for measurement purposes in cases where normal C instances are not desired/possible, for example: EEPROM layout specifications, or specifying hard-wired dump addresses for objects in SDL files[17]. An example for the use of vaddr (and the closely related vname and varrlen tags) is provided in Figure 19 Vaddr, Vname and Varrlen example source code.

```
#define MYVIRTADDR      0x20030000
#define MYSPECVIRTADDR  0x20030000 + sizeof(Example1_t)*4

typedef unsigned int ui32_t;
typedef unsigned char ui8_t;

typedef struct {
  ui32_t X,Y;
  ui8_t Obs[3];
} Example1_t;  /*!< first example @vaddr: MYVIRTADDR @varrlen: 4
@vname:Ex1 */

typedef struct {
  ui8_t NumFilled;
} Example2_t;  /*!< second example @vaddr: MYSPECVIRTADDR @vname:Ex2
@cycleid:MainCyc */

typedef struct {
    Example1_t SubEx1;
    ui8_t Num;
} Example3_t[20];   /*!< @vaddr: 0x300000, 0x400000 @vname: Ex3, Ex4
@varrlen:4 */
```

Figure 19 Vaddr, Vname and Varrlen example source code

The example contains four virtual object instantiations with the names *Ex1*, *Ex2*, *Ex3* and *Ex4*, with virtual addresses set to 0x20030000, 0x20030030 (assuming that the size of the structure Example1_t is 12 bytes), 0x300000 and 0x400000. The example illustrates that you can use numeric defines, constant expressions or immediate values for specifying virtual addresses. If multiple virtual objects need to be instantiated, just create a coma separated list of addresses. Note: that due to the way virtual addresses are specified, an external virtual address management needs to be implemented to avoid clashes between different virtual objects (i.e.: the address-range management usually done by a linker needs to be done manually in this case).

2.2.14 Vname

The vname comment tag is closely coupled with the vaddr tag, and specifies a name for the virtual instance created. It may only be used together with a vaddr tag and shall be followed by a C identifier token, i.e.: start with a letter or underscore followed by alphanumeric characters or underscores. It's use is optional, if not specified then

[17] The advantage of specifying hard-coded dump addresses over specialized output interfaces, is that the dump layout/address stays constant as long as the type is not changed. This is opposed to normal C variables which get new addresses assigned with each linker run.

the name of the commented C object is used, for example had there been no vname specified in Figure 19 Vaddr, Vname and Varrlen example source code for the vaddr in *Example2_t*, then the virtual instance name had been *Example2_t*. If multiple virtual instances are specified, then multiple vnames can be specified as well in the form of a coma separated list, but not more than virtual addresses specified.

2.2.15 Varrlen

The varrlen comment tag is closely coupled with the vaddr tag (it may only be used in a comment in conjunction with a vaddr entry), and specifies an optional array length for the virtual instance. For the source code in Figure 19 Vaddr, Vname and Varrlen example source code, the virtual instance named *Ex1* with virtual address 0x20030000 will be an array of four *Example1_t* typed objects. Note that the *Ex3* and *Ex4* virtual instances are also arrays of 20 structures, not because of an varrlen specification, but because the object they are bound to (*Example3_t*) is an array on its own. Had a varrlen been specified for either of them (*Ex3* or *Ex4*), then the result would have been a multi-dimensional array[18] with the first dimension of the varrlen value specified and second dimension of 20.
As with the vaddr and vname tags, multiple virtual instance's array lengths can be specified in the form of a coma separated list, but not more than virtual instances specified.

2.2.16 CycleId

The optional CycleId tag introduces an identifier (or a coma separated list of identifiers) that identify the special output view information of the SDL 2.0 file format. When generating SDL 2.0 output with pdo_tool these identifiers[19] can be assigned to a view with a name and a numeric cycle id. The use of the cycleid tag is demonstrated for the *Ex2* virtual instance in Figure 19 Vaddr, Vname and Varrlen example source code.
Starting from version 2.9.1.7 of PDO-Scan, the configuration file option *cycleid_per_inst* controls how cycle-ids are treated. When the option is disabled, then the behavior is as in older versions: the complete set of cycle-ids apply to all virtual instances (all virtual addresses) specified in the same comment. When the option is enabled, then the list of identifiers following the CycleId tag are treated the same way as virtual address (see chapter 2.2.13) or virtual name (see chapter 2.2.14) lists: they are assigned in the order of specification to the individual virtual instances.

[18] Note: certain final output formats (e.g.: SDL 2.0) do not support multi-dimensional arrays. In such a case the final output will be a single dimensional array with the multiplied product of the sub-dimensions of the multidimensional array.
[19] It is possible to assign several source-code identifiers to a single view, as well as to assign cycleid specification-less objects to a default view. Due to this please do not use the identifier *default* for cycle-ids. See PDO_TOOL

2.3 Language support

The parser supports the ISO-IEC 9899:1999 C programming language standard, with some commonly used extensions. As the standard itself provides a concise description of requirements for C programs, these are not listed here. For a detailed description of limitations and unsupported features of ISO-IEC 9899:1999, refer to chapter 2.8 Known issues/limitations. The remaining sub-chapters list extensions to the ISO-IEC C programming language that PDO supports.

2.3.1 Predefined symbols (macros)

The parser provides an additional predefined symbol __PDO__ with a replacement list of 1. This can be used to skip or process code special to PDO. In Figure 20 Predefined symbol use example for example, the type *Pdo_specific_enum_t* will be parsed by PDO, since the predefined __PDO__ is preset to 1.
A function-like predefined macro with the name __PDO_DEBUG_LEVEL is also provided, it accepts a single argument which should be an integer to set the debugging level to. This macro can be used limit debug output when diagnosing issues: simply use the macro to set the debug message verbosity to high at the given location where additional information is needed, resetting it to zero afterwards. The declaration of the field *Num* in the type *MyS_t* illustrates this in Figure 20 Predefined symbol use example.

```
typedef unsigned char ui8_t;

typedef enum MyE {
      ZERO,
      ONE,
      TWO,
      THREE,
      FOUR,
      FIVE,
      SIX,
} MyE_t;

#if __PDO__
typedef enum {
    NULL,
    EINS,
    ZWEI,
} Pdo_specific_enum_t;
#endif

typedef struct MyS {
      ui8_t Byte;
__PDO_DEBUG_LEVEL(9)
      MyE_t Num;
__PDO_DEBUG_LEVEL(0)
} MyS_t;
```

Figure 20 Predefined symbol use example

2.3.2 Near/Far pointers

When enabled via the configuration file, pdo_scan.exe provides support for the *near* and *far* keywords for use as type specifiers/qualifiers for pointers. It will only accept these specifiers/qualifiers in combination with pointer types, in order to support architectures where there are different pointer types for objects ‘near’ (usually some other register relative) or ‘far’ (usually a fully qualified pointer). Note that some compilers use the keywords near/far not as pointer type specifiers/qualifiers, but as linkage specifiers – as in this case the byte-layout of objects is not influenced by the keyword (only the address linked to, which is obtained from the map file anyway), please do not enable near/far keyword support, rather configure the parser to ignore any near/far keywords encountered (using empty defines for example, see chapter 2.6.

2.3.3 Pragma pack

Several compilers support the special pack pragma to declare types/variables with a different packing attribute (alignment). Example code is provided in Figure 21 Pragma pack code example, pdo_scan basically supports the alignment (packing) stack in the same manner as MS-VC does: providing a setting mode and operations to push/pop packing setting. The special *nopack* keyword as supported by IBM compilers is also supported.

```
typedef struct {
      unsigned char byte;
      #pragma pack(1)
      union {
            unsigned int ip;
            unsigned char bytes[4];
      } address;
      #pragma pack()

      #pragma pack(push, 2)
      void * pData;
      #pragma pack(push, 1)
      void * pOneAlignedPtr;
      #pragma pack(pop)
      void * pTwoAlignedPointer;
      #pragma pack(pop)
      void * pNormallyAlignedPtr;

      #pragma pack 2
      double dTime;
} packtest_t;
```

Figure 21 Pragma pack code example

2.3.4 Line number information

While reading source code, PDO supports reading source line number information specification in GCC and ISO standard format (useful when using external pre-processor, or for generated files, where input lines are taken from some other source than the C file passed to the compiler). The GCC line number information has following format:

# number filename [1|2]

Where number is a decimal number starting from 1, filename if the input file name and the terminating 1 or 2 indicates fresh entry into file or reentry after processing other input. The ISO standard line number information has following format:

# line number filename

Where number is a decimal number starting from 1 and filename is the input file name.

2.3.5 Empty input

While strictly speaking the ISO standard prohibits an empty translation unit, that is a C file that does not contain any parser object (an external declaration level object), most compilers silently accept this kind of input. To prevent annoying errors stemming from this PDO can also accept empty input files, depending on the setting of the *empty_input* option in the configuration file.

2.4 Output messages

During operation pdo_scan may generate output to the standard error stream. By default it is designed to produce no default output to stdout or stderr, unless an error or warning condition occurred, allowing clean integration into make/build processes. To ease this integration and allow source navigation, several output formats are supported, as well as the sensitivity of output messages can be configured. It is definitely not recommended to ignore this output, as several messages may indicate errors that may break operation, or lead to unexpected results. This is due to the fact that the tool employs extensive error recovery strategies to try to continue parsing code with errors: if within a structure the parsing of a field failed[20], all successive (correctly parsed) field offsets may be wrong, due to misinterpretation of the missing field.

2.4.1 Output line formats

To support easy integration into code-browsing environments, several line-number output formats are supported, which can be selected over the command line (for details see chapter 2.5 Command line) or configuration file (for details see chapter 2.6.3 Options).

[20] The ISO C 9899:1999 parsing interface has been extensively tested, thus errors in standard code are highly unlikely. Several compilers however employ special compiler-specific types, leading to the frequent cause of such errors in case of an omitted configuration for these special types in the configuration file.

2.5 Command line

Pdo_scan.exe treats everything that is not an option (begin with a ‘-‘ mark or part of a multi-word option) as an input C source file name. Historically older versions of PDO tools had options that both for short and long forms always began with a single ‘-‘. This has been changed for the 2.x line for better adherence to common computing standards: short options are prefixed by a single dash, while long options are prefixed by double dashes. For compatibility the current version still supports the single dash version of long options, but will emit a warning message. This support will be discontinued soon, so please use the corresponding long options (the short versions are not mentioned in this document).

Option name	Description
-h	Display help text and exit program
-o *filename*	Specify output *filename*
-l *filename*	Specify log *filename*
-d *number*	Specify debug level (verbosity of logging output)
-f	Filter mode: use standard in for C source code input
-p *filename*	Save internal pre-processor output to *filename*
-D*macro[=replacementlist]*	Define pre-processor *macro* with optional *replacement list*. When using whitespaces anywhere within this option, make sure to use appropriate shell quoting to pass as single argument.
-U*macro*	Undefine pre-processor *macro*
-I*path*	Add *path* to include path list. Include paths specified first are searched first. If whitespaces are within *path* make sure to use appropriate shell quoting to pass as single argument.
-c *filename*	Specify configuration *filename* to use
--include-once	Cache header file symbols, see chapter 2.7 Performance considerations
--no-include-once	Do not cache header file symbols
--use-reinclude *filename[,filename[,filename…]]*	Special option for use if file caching is enabled, to enable re-include for files needing different pre-processing in different stages. For details why this may be necessary see chapter 2.7 Performance considerations. Multiple coma separated files can be specified.
--textdat	Output old textual dat file format (not XML output), this option is for backwards compatibility only, please use XML format output in all new configurations. Also note that certain features (allow lists, cycleid, virtual instances, enumeration support) are not available with old dat support.

--xmldat	Output new XML based file format (default)
--mt-error-ignore	Do not output any messages for lexer errors encountered within comments
--mt-error-warning	Output warning for lexer errors encountered within comments
--mt-error-error	Output error for lexer errors encountered within comments
--enable-doxygen-comment	Enable lexing of doxygen comments
--enable-smart-comment	Enable lexing of smart-comments
--enable-teves-comment	Enable lexing of Teves-style comments
--enable-vsmsg	Enable MS-VS compatible line-number output for messages. These are formatted as *filename(lineno): message*
--enable-tevesmsg	Enable Teves-style line-number output for messages. These are formatted as *** *error in file ‘filename’ line number:* *message*
--enable-gccmsg	Enable GCC-style output messages. These are formatted as: *filename:lineno: message* (note: this is the default message format)
--enable-cc32rmsg	Enable CC32R-style output messages. These are formatted as *“filename”, line number: message*
--enable-fun-parse	Enable function parsing. This is experimental and only listed here for completeness, please don’t use this switch at present.
--disable-fun-parse	Disable function parsing. This is the default and the existence of the switch is experimental, merely listed here for completeness, please don’t use.

Table 2 Pdo_scan.exe options (summary)

The command line arguments passed are evaluated in-order, meaning that only options specified before a given source file will have influence on the behaviour of the program while parsing that given source file. For this reason it is generally best to specify options first and input files afterwards. While several input source files can be specified in a single invocation of the program, only a single output file can be generated.

2.5.1 Return code

The program will return a non-zero return code if an error[21] was encountered during parsing of input. If no errors were encountered than the program always returns zero.

[21] Errors are messages that are prefixed by *error:*, thus warning or informational messages do not influence the return code

2.5.2 Command line example: parsing single file

For parsing a single C file into a single XML database file:

```
pdo_scan.exe -c ../myparser.cfg -I ../abs -I ../ayc -D_APPL=1
myfile.c –o ../tmp/myfile.dat
```

This command line specifies to use the configuration file `../myparser.cfg`, use additional include directories `../abs` and `../ayc` (in that order, when searching for includes), define a macro `_APPL` with replacement list consisting of 1 and then parse the file `myfile.c`, outputting parsed information into the XML file `../tmp/myfile.dat`.

2.5.3 Command line example: parsing multiple files

An example for parsing multiple C files (using the same compilation settings/defines etc):

```
pdo_scan.exe –c ../myparser.cfg –I ../abs –I “C:\Program
Files\Microsoft Visual Studio 8\VC\include\” –DTP_SIMU=1 –
DABS(x)=((x>=0)?x:-x) ./src/tp/ars3xx/tp_main.c
../src/tp/ars3xx/tp_simu.c ../src/tp/ars3xx/tp_tracking.c
../src/tp/ars3xx/tp_maintenance.c –o ../tmp/tp/ars3xx/tp.dat
```

This command is very similar to the previous one, but parses the information found in 4 C files (`./src/tp/ars3xx/tp_main.c`, `../src/tp/ars3xx/tp_simu.c`, `../src/tp/ars3xx/tp_tracking.c`, `../src/tp/ars3xx/tp_maintenance.c`) into a single DAT output file (`../tmp/tp/ars3xx/tp.dat`). The example also illustrates the command line specification of a function-like ABS macro, as well as the possibility to specify absolute include paths[22].

[22] Take care to use the quoting style appropriate for your command interpreter/environment for paths containing spaces

2.6 Configuration file

The configuration file can be used to adapt pdo_scan.exe's operation to the given compiler in use: most notably to define basic C data types and their layouts and pre-defined pre-processor defines (if any), and alias compiler-specific types. The syntax of the configuration file is (somewhat) similar to C's syntax. Comments use the same syntax as C comments, with support for both traditional and single line (C++) style comments. The following chapters delve into the details of the three main parts of configuration files: pre-processor settings, type/layout definitions and controlling options.

2.6.1 Pre-processor settings

Compiler specific defines/symbols can be defined with the help of C-like pre-processor define statements. If compiler-specific keywords are used, which do not have significant influence for XML generation[23], then these can be defined to expand to no tokens. An example for the M32R is provided in Figure 22 Configuration file pre-processor defines section.

```
/* The CC32R compiler has a pre-set __M32R__ macro */
#define __M32R__ 1

/* Just in case 'inline' was specified with an alternative spelling
as supported by CC32R, redefine it to ISO C standard ‘inline’ */
#define __inline inline
```

Figure 22 Configuration file pre-processor defines section

2.6.2 Type definitions

Type definitions can be used to specify the layouts of standard C types as used by the compiler. An example is provided in Figure 23 Type definitions example.

```
type ( unsigned int | unsigned long )
{
      alignment   = 4;        /* 4-byte aligned always */
      size        = 4;        /* Size is 4 bytes */
      byteorder   = motorola; /* Byte-order is motorola */
      reptype     = uint32;   /* Representation is unsigned 32-bit
integer */
}
```

Figure 23 Type definitions example

[23] A good example is the near/far keyword as it is used by some compilers: if it is not a type modifier (i.e.: influences the layout of a type), then it can be safely ignored, as the sizes/layouts of types can be safely determined without parsing these. A good example of this is the TMS 470 compiler, where it acts like a linkage specifier, thus only influences the absolute address assigned to a variable.

Fundamentally a type definition has to begin with the type keyword, followed by a parenthesized C types separated by the '|' symbol[24], followed by a block enclosed by '{' and '}'. Within the block C-like assignment expressions on keywords can be used to specify size (in bytes, keyword *size*), alignment requirement (in bytes, keyword *alignment*), byte-order (keyword *byteorder*) and representation type (keyword *reptype*).

Type	Description
unsigned char	Unsigned character type
signed char	Signed character type
char	Default character. Note: signedness handling is determined by associated representation type.
unsigned short int	Unsigned short integer
signed short int / short int	Signed short integer. Note: as specified by ISO C99 default short int is signed.
unsigned int	Unsigned integer
signed int /int	Signed integer. Note: as specified by ISO C99 default int is signed
unsigned long int / unsigned long	Unsigned long integer
signed long int / long int / long	Signed long integer
unsigned long long int / unsigned long long	Unsigned long-long integer (if supported). Note: if compiler does not support long longs, just do not add type definition for long long. In this case error message will be emitted on attempt to use long long type.
signed long long int / long long int / long long	Signed long-long integer (if supported). Note: if compiler does not support long longs, just do not add type definition for long long. In this case error message will be emitted on attempt to use long long type.
float	Float type
double	Double type
long double	Long double type.
struct / union / array	Certain compilers empose special rules for arrays/structs/unions (such as special alignment). If this is the case, please specify the rules to enforce (for example alignment) only[25].
enum	Settings for enumerated types

Table 3 Types specifiable in configuration file

[24] Thus if a single type is specified, just write the type between the parenthesis, type listing provided in Table 3 Types specifiable in configuration file.

[25] For example some compilers always place arrays 4-byte aligned, even if the elements themselves would only require 1 byte alignment. In such a case please specify alignment.

For alignment and size specify numbers. For byte-order specify either *motorola* / *intel*, *bigendian* / *littleendian* or *msbfirst* / *lsbfirst* (depending on taste). For representation types supported see Table 4 Representation types supported.

Reptype	Description
ubyte	Unsigned 8 bit integer type [0 .. 255]
sbyte	Signed 8 bit type [-128 .. 127]
uint16	Unsigned 16 bit integer type [0 .. 65535]
sint16	Signed 16 bit integer type [-32768 .. 32767]
uint32	Unsigned 32 bit integer type [0 .. (2^{32}-1)]
sint32	Signed 32 bit integer type [- 2^{31} .. (2^{31} -1)]
uint64	Unsigned 64 bit integer type [0 .. 2^{64}-1]
sint64	Signed 64 bit integer type [- 2^{63} .. (2^{63} – 1)]
float32	32 bit floating point type
float64	64 bit floating point type
float80	80 bit floating point type
auto	Special representation type only to be used for *enum* types. It can be used for compilers that automatically choose different representation types for enumerations based on the represented value range.

Table 4 Representation types supported

2.6.3 Options

Most behavioral aspects of pdo_scan can be controlled via the options specifiable in the configuration file, such as verbosity of output messages, output message formatting, comment support options etc. The Table 5 Configuration file options lists the summary of supported options.

Option name	Description	Possible values (equivalent options separated by /)
msg_format	Selects output message format. The default is GCC-style line number information.	*"msvs"/"msvs_msg"* or *"gcc"/"gcc_msg"* or *"teves"/"teves_msg"* or *"cc32r"/"cc32r_msg"*
doxygen_comments	Doxygen-style comments option	*enable*/*on* or *disable*/*off*
smart_comments	Smart-comments option	*enable*/*on* or *disable*/*off*
teves_comment	Teves-style comment option	*enable*/*on* or *disable*/*off*
blank_line_reset_comment	Setting if a blank line between a comment and declaration shall prevent the comment from being bound to the declaration.	*enable*/*on* or *disable*/*off*
define_line_reset_comment	Setting if a pre-processor #define line between a comment and a declaration shall prevent the comment from being bound to the declaration.	*enable*/*on* or *disable*/*off*
merge_multiple_comments	Setting if multiple comments seemingly belonging to the same declaration shall be merged or only the nearest/first be kept.	*enable*/*on* or *disable*/*off*

a_info_support	Special obsolete A_INFO and ARR_INFO support	*enable*/*on* or *disable*/*off*
macros_in_comments	Option to enable macro expansion in comments	*enable*/*on* or *disable*/*off*
cycleid_per_inst	New option for version 2.9.1.7. Specifies if identifiers following cycle-id specifications in comments shall be assigned to individual virtual instances, allowing a single comment to specify multiple objects in different SDL cycles.	*enable*/*on* or *disable*/*off*
mixed_bit_fields	Enable mixing of different types of bit-fields. For example certain compilers allow the use of signed/unsigned integer bit fields in the same integer allocation unit – for these enable this option. Other compilers always start a new allocation unit for a different bit-field type, for these disable this option.	*enable*/*on* or *disable*/*off*
bit_field_piggyback	Enable piggybacking of bit-fields onto non-bitfield types within the same allocation unit. Some compilers support packing bit-fields into partially used allocation units, where previously non-bit-field types were allocated. For example if a struct starts with a byte followed by an unsigned integer bit-field with a width of 1, then these compilers will use the integer with offset 0 (i.e.: the same integer the byte field was in) with an adjusted mask. For compilers that allocate like this, enable this feature, otherwise disable it.	*enable*/*on* or *disable*/*off*
bit_field_auto_type	Enable auto-determination of bit-field base type, ignoring C-level specification. Some compilers auto-decide the allocation unit to be used for a bit-field based on the necessary number of bits. (Example: DIAB compiler)	*enable/on* or *disable/off*
bit_field_max_comp_size	Certain compilers enable bit-field compression. Meaning if a new bit-field declaration would span more then 'bit_field_max_comp_size' bytes if shared with previous bit-fields allocation units, then a new allocation unit is started. Example of this is the DIAB compilers – Xbit-fields-compress option.	*1,2,4,8 (integer number)*
asm_keyword	Enable support for the *asm* keyword as specified by ISO-IEC 9899:1999: the keyword has to be followed by a parenthesized string literal constant. If the used compiler uses assembler statements in that form, then set to enabled. Otherwise set to disabled, in which case *asm* will be treated as an identifier.	*enable*/*on* or *disable*/*off*
inline_keyword	Select *inline* keyword support. If the compiler supports the function specifier inline, set to enabled.	*enable*/*on* or *disable*/*off*

far_keyword	Select *far* keyword support. Certain compilers treat the identifier-like token *far* as a type modifier keyword for pointers. If the used compiler does this (and only for pointers, if it is a linkage-style specifier, then use an empty define in the defines section), then enable this option.	*enable*/*on* or *disable*/*off*
near_keyword	Select *near* keyword support. Certain compilers treat the identifier-like token *near* as a type modifier keyword for pointers. If the used compiler does this (and only for pointers, if it is a linkage-style specifier, then use an empty define in the defines section), then enable this option.	*enable*/*on* or *disable*/*off*
restrict_keyword	Select *restrict* keyword support as specified by ISO-IEC 9899:1999: this is specified to be a function parameter modifier keyword. Enable if the compiler treats *restrict* as a keyword. If disabled *restrict* is treated as an identifier.	*enable*/*on* or *disable*/*off*
typeof_keyword	Enables support for the 'typeof' keyword. This allows using typeof expressions for type declarations that some compilers support (experimental).	*enable*/*on* or *disable*/*off*
strict_float_prec	Strict float precision option: if enabled then all floating point arithmetic is used in the precision of the compiler-type (specified in the Type definitions chapter 2.6.2). If disabled then the default precision of the host running pdo_scan is used. Most embedded compilers seem to use the highest available precision for constant expression evaluation, thus usually set to off.	*enable*/*on* or *disable*/*off*
auto_unsigned_const	Automatically lex large integers not representable as signed ints as unsigned integers even without the *u* suffix. This is generally a good idea that most compilers seem to do, so usually enable this option.	*enable*/*on* or *disable*/*off*
parse_functions	Enable parsing of functions. Option only listed here for completeness, set to disabled/off!	*enable*/*on* or *disable*/*off*
file_caching	Enable file caching functionality. For details see chapter 2.7 Performance considerations	*enable*/*on* or *disable*/*off*
reinclude_file	List of file names to exclude from caching (reinclude/relex on every include). For details see chapter 2.7 Performance considerations	coma sperated list of strings with file names[26]
bitorder	Set bit allocation order for bit-fields. Select *motorola/msbfirst/big-endian* for compilers allocating the highest order bit	*motorola/msbfirst/big-endian* or *intel/lsbfirst/little-endian*

[26] When using relative file names in config file, please use relative path to current working directory of pdo_scan.exe process. File names specified here have to match the specifications for include paths from the command line, i.e.: when using relative paths for include dirs, specify relative paths for files.

	first, *intel/lsbfirst/little-endian* for compilers allocating the lowest order bit first.	
comment_errors	Reaction to exhibit, when a lexing error occurs within comments.	*error*, *warning* or *ignore*
macro_ident_redef	Reaction to exhibit when an identical macro redefinition is made (allowed by ISO-C). Note: non identical redefinitions are always treated as errors.	*error*, *warning* or *ignore*
macro_useless_undef	Reaction to exhibit for useless undefined statements (where identifier being undefined wasn't defined).	*error*, *warning* or *ignore*
undef_ppc_condition	Reaction to exhibit for identifiers in pre-processor conditionals that do not have definitions. The ISO standard specifies replacing all such identifiers with 0 – this is always done by pdo_scan, this option controls whether a diagnostic message shall be emitted.	*error*, *warning* or *ignore*
float_in_ppc_cond	Reaction to exhibit for floating-point numbers in pre-processor conditions. The ISO standard forbids this (they are not supported), thus generally compilers do not support floating point numbers in pre-processor conditionals. Pdo_scan can evaluate pre-processor conditionals using floating point expressions, treating all non-zero expressions as true.	*error*, *warning* or *ignore*
string_in_ppc_cond	Reaction to exhibit for string constants in pre-processor conditions. The ISO standard forbids use of string literals in pre-processor conditions. Pdo_scan evaluates all non empty strings to true.	*error*, *warning* or *ignore*
char_in _ppc_cond	Reaction to exhibit for character literal constants in pre-processor conditions. While the ISO standard allows this, it's use is strongly discouraged, as encoding issues may provide unexpected results.	*error*, *warning* or *ignore*
type_info_fail	Reaction to exhibit when internal allocator fails to provide type information for C declaration. Do not set this to values other than *error*, unless you really know what you're doing. (Very special use case of not using internal allocator, but specifying complete type layouts over map file – this is obsolete, as it prevents parsing of several C expressions)	*error*, *warning* or *ignore*
eof_in_ppc_ctrlline	Reaction to exhibit if end of file is encountered within a pre-processor control line. This is strictly forbidden by ISO C, but most compilers do not emit any diagnostic message for it.	*error*, *warning* or *ignore*
empty_input	Reaction to exhibit for empty input files. Compilers show great variance in how they handle this situation. The ISO standard specifies this to be an error.	*error*, *warning* or *ignore*
implicit_int_type	Reaction to exhibit if an implicit int is declared. The ISO C standard allows this (for backwards compatibility only), but	*error*, *warning* or *ignore*

	ISO C++ for example forbids it. Use of implicit ints should be generally considered dangerous as future standards may remove this feature completely.	
symbol_redefinition	Reaction to exhibit if a symbol on C parser level is redefined. This can happen for example when multiple C modules are being parsed, with two C modules declaring different meanings for the same symbol. Generally dangerous practice, should be avoided as is very misleading, thus usually set this option to warning or error.	*error*, *warning* or *ignore*
incomplete_fwd_ref	Reaction to exhibit for incomplete forward type references (struct/union/enum tag name references). This is allowed by the ISO C standard and may be completely valid, though experience shows that it is often unintended.	*error*, *warning* or *ignore*
fun_par_name_redef	Reaction to exhibit when function is declared with different parameter names. Some compilers treat this as an error, thus it may be wise to have a diagnostic for this situation.	*error*, *warning* or *ignore*
comment_desc_clash	Reaction to exhibit when description strings in comments clash for a certain object. (For example: a different description is provided in external header file than in C file).	*error*, *warning* or *ignore*
comment_min_clash	Reaction to exhibit when minimum specifications in comments clash (for example: different minimum value provided in header file than in C file).	*error*, *warning* or *ignore*
comment_max_clash	Reaction to exhibit when maximum specifications in comments clash (for example: different maximum value provided in header file than C file).	*error*, *warning* or *ignore*
comment_offset_clash	Reaction to exhibit when offset specifications in comments clash.	*error*, *warning* or *ignore*
comment_factor_clash	Reaction to exhibit when resolution specifications in comments clash.	*error*, *warning* or *ignore*
comment_unit_clash	Reaction to exhibit when unit string specifications in comments clash.	*error*, *warning* or *ignore*
comment_name_clash	Reaction to exhibit when name specifications in comments clash.	*error*, *warning* or *ignore*
comment_virtual_clash	Reaction to exhibit when virtual instances in comments have conflicting information.	*error*, *warning* or *ignore*
comment_values_clash	Reaction to exhibit when comment values declarations contain errors or conflicting information.	*error*, *warning* or *ignore*
comment_designator_error	Reaction to exhibit for invalid special designators in comment values declarations	*error*, *warning* or *ignore*
comment_values_parse_error	Reaction to exhibit on parsing errors in comment values declarations.	*error*, *warning* or *ignore*
comment_logical_error	Reaction to exhibit when logical errors	*error*, *warning* or *ignore*

	are encountered in comments (for example minimum specified greater than maximum etc.)	
comment_unbound_virtinst	Reaction to exhibit when a comment with a virtual instance can not be bound to any object (ambiguous comment)	*error*, *warning* or *ignore*
charset	Input character set	*“iso8859-1”, “utf-8”, “cp437”, “cp852”* or *“cp1252”*
sys_inc_messages	System include message control: if you don’t wish for messages to be emitted by system includes (#include <x>), then you can control the parsers behavior via this option. New for version 2.7	*enable* or *disable*
sys_include_dirs	System include directories. Specify a coma separated list of quoted strings after this option. Relative and absolute paths may be used, the relative paths have to be relative to the current working directory where pdo_scan.exe is launched from. New for version 2.7	Coma separated quoted strings. Example: “C:\M32R\include”, “C:\M32R\include\sys”
extended_type_compat	Extended type compatibility setting. When enabled (the default), then objects of the same name that evaluate to the same C type are treated as compatible merged accordingly. If disabled, then only exactly matching types (i.e.: no additional typedef indirection allowed) are treated as compatible. For example: typedef void VOID_t; void fun(void); VOID_T fun(VOID_t); would be compatible when enabled, but incompatible when disabled.	*enable*/*on* or *disable*/*off*
multi_instance_statics	Option to control creation of ‘static’ instances. Basically if enabled, each new static declaration encountered in a new compilation unit creates a new instance (as in a real compiler). Usefullness of this is limited, since most tools don’t support differentiation based on compilation unit scope.	*enable*/*on* or *disable*/*off*

Table 5 Configuration file options

2.7 Performance considerations

In large projects with complicated include hierarchies traditional compilation schemes like calling the compiler/parser for each C source file can become a significant bottleneck, as each C module can include megabytes of headers. To ease the process start/stop overhead as well as output file creation overhead pdo_scan allows passing multiple C sources for parsing in a single invocation of the program. This feature can always be used and has no additional requirements for the input.

Additionally a second feature called *file caching* has been implemented in pdo_scan. When enabled, this feature caches the pre-processor defines and external declaration objects parsed for each file. Due to the way it works (caching on pre-processor define level and parser external declaration level) it requires header files to fulfill a set of requirements:

- A header file has to consist of complete external declaration level declarations.
- Pre-processing of the header file must yield the exact same token sequence on first inclusion when included in different C files. Subsequent includes of the header file shall not lead to any tokens being passed to the C parser or modification of the pre-processor state (through include guard ifndefs for example).
- Header files shall not undef any pre-processor defines not defined within the same file. (i.e.: the state change of the pre-processor by inclusion of a header file shall only add additional definitions, not delete existing ones).

The following figures illustrate a header file useable with caching, as well as three examples violating the above requirements.

```
#ifndef SI_OUTPUT_H
#define SI_OUTPUT_H

#include "glob_cfg.h"

/*! Some structure */
typedef struct {
    ui32_t Flags;
} SIFlags_t;

extern void SIProcess(void);

/*! counter of executed SI cycles */
extern ui32_t SICycleCounter;

extern

#endif
```

Figure 24 Example of header file useable with file caching

```
int MyArray[] = {
#include "MyArrayVals.h"
};
```

Figure 25 Example 1 of header inclusion not useable for caching (not external declaration)

```
#ifndef BAD_FILE_INCLUDE_1
#define BAD_FILE_INCLUDE_1

/* First include processes this */
typedef struct {
    ui32_t Bad;
} Bad_t;

#define EXTERN extern

#else
#ifndef BAD_FILE_INCLUDE_2
#define BAD_FILE_INCLUDE_2

/* Second include processes this */
extern void BadFunc(void);

#else
/* Third include gets here */
#endif
#endif
```

Figure 26 Example 2 of header file not useable with caching (different processing on reinclude)

```
#ifndef MY_INCLUDE_FILE
#define MY_INCLUDE_FILE

#undef SOME_DEF_FROM_OTHER_FILE
#define SOME_DEF_FROM_OTHER_FILE 15

#endif
```

Figure 27 Example 3 of header file not useable with caching (modifying defines from other files)

Most C source code fulfills these requirements, since most of it is just common sense. Since usually only a small subset of files does not satisfy these requirements, a special command line option or configuration file setting can be used to force reinclusion (and thus relaxing/parsing) of these files when file caching is enabled[27]. An example for enabling file caching and specifying two reinclude files is provided on Figure 28 Example config file excerpt for file caching.

[27] When file caching is disabled, then traditional pre-processing is done, thus each include reads to relaxing/parsing

```
file_caching = enable;              /* Flag to enable/disable file
caching. Possible values: enable/on, disable/off */

reinclude_file = "../src/os/ars3xx/tcb.h", "../src/os/vpu2xx/tcb.h";
      /* Coma seperated list of files to reinclude every time */
```

Figure 28 Example config file excerpt for file caching

The speed improvements through file caching are most pronounced when multiple C files are being scanned in a single invocation of the program, since this allows caching header contents for parsing multiple C files.

2.8 Known issues/limitations

2.8.1 64 bit type support

While providing basic support for 64 bit data types, this support can not be considered complete. The main problem is that most platforms for which PDO can be compiled have incomplete/partial library support for large integer conversions. The MS-VS environment for example does not support ISO-IEC strtoimax family of functions, but does provide signed 64 bit support via the proprietary _strtoi64 function. This precludes the use of unsigned 64 bit values not representable on 63 bits.
Additionally to simplify internal code of PDO, min-max values are represented using the 'double' C type, which usually has 53 bits of precision, leading to possible loss of the least significant bits. These issues are of secondary importance, as all currently tested measurement environments have more severe precision / accuracy restrictions than PDO itself, thus PDO is unlikely to become the bottleneck.
Note: basic 64 bit support has been reworked in version 2.9. Some limitations still exist, but 64 bit values can now be used for SDL bit-masks as well as wide (wider then 32 bits) wide bit-fields are now supported.

2.8.2 Wide string literal support

Fundamental support for wide strings is implemented, but has not been extensively tested, as wide string literal support in embedded apps is rare (actually no known use-case).

2.8.3 Function parameter parsing errors

The current parser implementation emits spurious error messages when a parsing error within a function parameter list is encountered. As these error messages are only emitted when an actual C syntax error is found, and having multiple error messages due to the same error is only a nuisance, no fix is planned and no workarounds are provided[28].

2.8.4 Values comment declaration errors

Due to internal architecture of the scanner detailed line number information for parsing errors encountered within comment values declarations can not be provided. This is annoyance when multi-line values declarations are being used, as regardless of which line the error occurred in, the position indicator will point to the end of the values declaration. Currently no fix is planned, as no real influence on parsing, would only provide better output messages.

[28] Providing a pure parser level fix is very difficult due to error recovery rules potentially clashing. Fixing via a lexer-parser feedback loop seems easier, but not really worth the hassle

2.8.5 Wide floating point numbers

The parser currently provides support for floating point numbers with up to 80 bits precision (when compiled with GCC on x86) or 64 bit precision (when compiled with MSVC on x86). Due to this floating point calculations can be done internally with a precision of at least 64 bits. When parsing for architectures with higher floating point precision (EG: DEC Alpha with 128 bits), round-off errors may occur, and no proper representation type can be specified in configuration file. As currently even embedded MCUs with double precision are extremely rare, there are no plans to extend precision.

2.8.6 Function local static variables

The current implementation does not parse function definitions, thus only global declarations are parsed. Thus while technically function local statics (with constant addresses) can be described in measurement files, PDO currently has no support for them.

2.8.7 Nested variable length type instantiations

Currently no error messages are emitted for nested variable length types specified in invalid positions (for example code, see Figure 29 Variable length types). As these are errors that should be caught by normal C compilers, the significance of this is very little.

```
struct VarLenStruct {
      int fieldi;
      char str[];
};

struct IllegalStruct {
    struct VarLenStruct;    /*!< Variable length type use */
    int x;                  /*!< Variable length type may not be
followed by other fields!!!! */
} InvalidVarNesting;
```

Figure 29 Variable length types

2.8.8 Textdat output

With the option —*textdat* the current version of pdo_scan can output information in the old textual database format. As this format can not store all information present in the newer XML file format, it shall be only used for legacy projects, where compatibility with 1.xx version of PDO tools is a must. The following features are not supported with textual dat output:

- Virtual instance related comments (vaddr, vname, varrlen)
- Cycleid comments
- Allow lists in comments
- Values declarations in comments

- Enumeration output (information which enumerators belong to which enumeration type are lost)

While care has been taken to maintain as much compatibility as possible in the textual dat file, some differences should be noted:

- Enumerators are no longer output (since old dat file output them to global namespace, which is wrong)
- Enumeration types are no longer marked with type 00h (undefined), but are rather set to 03h (integer) in accordance with common convention. Note: ISO-IEC conformant C compilers do not have to allocate the same for an int and an enum type, the enum only has to be convertible to an int.
- Due to bugs in old 1.xx versions, storage class comparisons may yield differences. Old versions did not properly merge storage class information from different declarations

2.8.9 Escaped newlines

Support for escaped new-lines is only provided outside of tokens or within string literals. The comment starting symbol and comment end symbol may also not be interrupted by an escaped new-line. Thus placing an escaped newline within a C level token (other than a string) will cause errors. As this is generally not recommended, as several compilers/editors also show incomplete support for it, this should not cause trouble in "normal" source code, just place new-line escape sequences between tokens.

```
/\
* Bad idea to comment like this \
*\
/

#def\
ine M(x)    MY_MACRO(x)

d\
o {
    i++;
} whi\
le
```

Figure 30 Unsupported escaped new-lines

The Figure 30 Unsupported escaped new-lines illustrates the escaped new-lines within tokens are generally not supported. The example also highlights that the MS-VS editor also does not recognise these escaped tokens.

```
const char MyStr[] = "this is a long\
multiline string, with escaped newlines";

#define GOOD(x) \
    MY_MACRO(x)

#define SOMECODE    \
    do {\
        i++; \
    } while (0)
```

Figure 31 Supported escaped new-lines

The Figure 31 Supported escaped new-lines shows the positions where escaped new-lines are supported: between tokens or within string literals.

2.8.10 Predefined symbols

Currently pdo_scan.exe predefines the symbols __TIME__, __DATE__, __LINE__ and __FILE__ as defined by the ISO-IEC 9899:1999 standard. It however does not predefine the standard symbols __STDC__, __STDC_HOSTED__, __STDC_VERSION__, __STDC_IEC_559__, __STDC_IEC_559_COMPLEX__, __STDC_ISO_10646__. This is on purpose, so that customization of the configuration file can be made to mimic the behavior of the target compiler used as closely as possible.

2.8.11 System include message suppression

Using the configuration file option *sys_inc_messages*, the messages produced for code found in files included through system include statements of the form #include <filename> can be suppressed. This mechanism usually works fine with one notable exception: as the parser uses one token of look ahead, if the last external declaration level statement of a system include contains code which leads to messages, these messages may be output even if system include messages are otherwise suppressed. As the problematic compiler specific statements are usually not last in file this normally does not pose a significant issue.

2.9 XML Output

The output XML is always encoded in UTF-8 encoding and has been designed to be human readable as far as possible. To avoid generation of invalid XML files, all special characters are transcoded where necessary. This XML output should be readable by most external tools, a schema for it is provided in the file *xdat.xsd*. The following two subchapters illustrate the XML output format for a simple input file.

2.9.1 Example input code

```
/*! Possible vehicle stabilization interventions */
typedef enum { ABS, AYC, TCS, } Intervention_t;

/*! Brake pressure unsigned representation type
    @unit: Bar @resolution: 0.01 @offset : 0.5
    @max:500.0 @min:0.8 */
typedef unsigned short BrakePressure_t;

typedef struct IntvChainTag {
    Intervention_t IntV;
    float Time;     /*!< Intervention length @unit:s @max:2
@min:0.01*/
    BrakePressure_t FrontBrake;
    BrakePressure_t RearBrake;
    const struct StWithTagName * pNext;
} InterventionChain_t;

static InterventionChain_t IntvChainHead;

extern void ProcessIntv(const InterventionChain_t * pIntv);
```

2.9.2 Example output XML file

```
<?xml version="1.0"?>
<PDO_DATA gentime="Wed Aug 12 17:33:26 2009" toolname="PDO-Scan  Revision: 2.5 RC1 " numerrors="0">
        <NLE name="Intervention_t" sc="typedef" filename="example.c" lineno="2">
                <TLE type="enum" align="4" reptype="sint32" size="4">
                        <NLE name="ABS" value="0" sc="enum" filename="example.c" lineno="2">
                        </NLE>
                        <NLE name="AYC" value="1" sc="enum" filename="example.c" lineno="2">
                        </NLE>
                        <NLE name="TCS" value="2" sc="enum" filename="example.c" lineno="2">
                        </NLE>
                </TLE>
                <MTE filename="example.c" lineno="1">
                        <COMMENT>
                                Possible vehicle stabilization interventions
                        </COMMENT>
                </MTE>
        </NLE>
        <NLE name="BrakePressure_t" sc="typedef" filename="example.c" lineno="7">
                <TLE type="unsigned short" align="2" reptype="uint16" size="2"/>
                <MTE unit="Bar" factor="0.01" min="0.8" max="500" filename="example.c" lineno="4">
                        <COMMENT>
                                Brake pressure unsigned representation type
                        </COMMENT>
                </MTE>
        </NLE>
        <NLE name="struct IntvChainTag" sc="tagname" filename="example.c" lineno="9">
                <TLE type="struct" align="4" size="16">
                        <NLE name="IntV" filename="example.c" lineno="10" addroffs="0x0">
                                <TLE type="typeref" def_type="Intervention_t"/>
                        </NLE>
                        <NLE name="Time" filename="example.c" lineno="11" addroffs="0x4">
                                <TLE type="float" align="4" reptype="float32" size="4"/>
                                <MTE unit="s" min="0.01" max="2" filename="example.c" lineno="11">
                                        <COMMENT>
                                                Intervention length
                                        </COMMENT>
                                </MTE>
                        </NLE>
                        <NLE name="FrontBrake" filename="example.c" lineno="12" addroffs="0x8">
                                <TLE type="typeref" def_type="BrakePressure_t"/>
                        </NLE>
                        <NLE name="RearBrake" filename="example.c" lineno="13" addroffs="0xA">
                                <TLE type="typeref" def_type="BrakePressure_t"/>
                        </NLE>
                        <NLE name="pNext" filename="example.c" lineno="14" addroffs="0xC">
                                <TLE type="pointer" align="4" reptype="uint32" size="4">
                                        <TLE type="const typeref" def_type="struct IntvChainTag"/>
                                </TLE>
                        </NLE>
                </TLE>
        </NLE>
        <NLE name="InterventionChain_t" sc="typedef" filename="example.c" lineno="15">
                <TLE type="typeref" def_type="struct IntvChainTag"/>
        </NLE>
        <NLE name="IntvChainHead" sc="static" filename="example.c" lineno="17">
                <TLE type="typeref" def_type="InterventionChain_t"/>
        </NLE>
        <NLE name="ProcessIntv" sc="extern" filename="example.c" lineno="19">
                <TLE type="function" align="4">
                        <TLE type="void"/>
                        <NLE name="pIntv" filename="example.c" lineno="19">
                                <TLE type="pointer" align="4" reptype="uint32" size="4">
                                        <TLE type="const typeref" def_type="InterventionChain_t"/>
                                </TLE>
                        </NLE>
                </TLE>
        </NLE>
</PDO_DATA>
```

3 PDO_TOOL

Pdo_tool.exe primarily acts as a linker/output generator: it reads the files generated by pdo_scan.exe optionally merges the information with address information read from a standardized map file and outputs one of three possible output formats:

- XML Database files using the same output format as pdo_scan.exe does
- ASAP2 (A2L) files
- SDL 2.0 Files

The recommended use to first merge all XML Databases generated by pdo_scan.exe and the standard map file information into one large XML database, and then use that merged database as input in following calls to pdo_tool.exe for generating A2L/SDL2.0 information[29].

3.1 Command line

In general everything not prefixed by '-' or part of a multi-word option is treated as an input file. The program returns zero to the OS if no errors occurred, and non-zero on error.

Option	Description
-f [format]	Specifies the output file format, currently valid formats: - xmldat1.0 : output PDO XML database - sdl2.0 : output SDL 2.0 file - asap2 : output ASAP 2 (A2L) file - textdat : obsolete textual PDO database file Example: *-f sdl2.0*
-h	Displays help information and then exits Example: *-h*
-d [num]	Set debug level to given number. Number has to be specified decimal. Note: the higher the number the more verbose the output log is. Example: *-d 9*
-c [mapfile]	Load map file 'mapfile' to get physical addresses of variables. Note: that the load takes place at the position in the command line where it was specified, i.e.: it will only influence symbols loaded from DAT files before the –c option was specified. Example: *-c ../out/ars300_master_appl_release.std.map*
-o [outputfile]	Save output in 'outputfile'. Example: *-o ../out/ars300_master_appl_release.a2l*
--config [configfilename]	Load pdo_tool configuration file [configfilename].

[29] The merged XML database is usually a fraction of the size of the individual XML files, as through includes the same declarations can occur in several XML files. This leads to a noticeable speed improvement when having to read the merged XML file only, not having to reprocess the individual XML inputs

-l [logfile]	Log error/warning and debug messages in ‘logfile’ Example: *-l ../out/ars300_master_appl_release_a2l.log*
-a [allowid1[,allowid2…]]	Set list of allow IDs to use when generating ASAP2 or SDL2.0 output. Note: the identifiers comprising the allow list shall be separated by comas with no whitespaces in the list. Example: *-a user_bmw,user_audi,conti_intern*
-i [ident1[,ident2…]]	Set list of identifiers to filter for when generating ASAP2 or SDL2.0 output. Note: the identifiers in the list shall be separated by comas with no whitespaces in the list. Example: *-i TPObjectList,SIParams*
-r [range1[,range2…]]	Set physical address ranges to generate output for when outputting ASAP2, SDL2.0 format files. The ranges themselves have to specified in start address – end address format, without whitespaces in between. The addresses themselves can be specified in decimal or hexadecimal (prefixed by 0x or 0X). Example: *-r 0x804000-0x82FFFF,0x2000-0x2FFF*
-v [range1[,range2…]]	Set virtual address ranges to generate output for when outputting ASAP2 or SDL2.0 format files. The ranges themselves have to be specified in the same format as for the –r option. Example: -v 0x20000000-0x7FFFFFFF
--asapnamelimit [n:m]	Set default ASAP name length limits. Option only has an effect when generating ASAP2 output. The first decimal number specifies the total length, the second number specifies the maximum partial name length. The ASAP2 standard (v 1.51) specifies that the total name may be 255 characters at most, while the partial name length may be 32 characters at most. Example (for using maximal length): *--asapnamelimit 255:32*
--asapprojname [name]	Set ASAP project name when generating ASAP2 output. The project name shall contain no whitespaces and shall follow the ASAP identifier rules. Example: *--asapprojname bmw_f10*
--asapmodulename [name]	Set ASAP module name when generating ASAP2 output. The module name shall contain no whitespaces and shall follow the ASAP identifier rules. Example: *--asapmodulename ars350*
--asapformatspec [n.m]	Set default ASAP format string: the ‘n’ and ‘m’ options have to be decimal numbers specifying total and fractional precision for display. Example: *--asapformatspec 16.3* Displays total 16 digits, with 3 after the decimal point.
--asapmaxmatrixdim [n]	Set maximum matrix dimensions for ASAP2 output. The option shall be passed with a decimal number [n] between 0 (no matrix/array output) and 3 (maximal allowed matrix

	dimensions in ASAP). Note that most tools (including CANape) only support matrix dimensions up to 2. Example: *--asapmaxmatrixdim 2*
--cycleinfo [name1[,name2]] [viewname] [viewed]	Set SDL2.0 output file cycle information. The list of names has to be a coma separated list without whitespaces of identifiers use in @cycleid comment tags. Additionally the name 'default' may be used to indicate those objects that do not have any @cycleid specified. The viewname is the name of the view element in the SDL2.0 output file. The viewid is the numeric ID of the view output. To specify multiple output views, use multiple –cycleinfo options. Example: *--cycleinfo default,FCT_ENV,EM_ENV ARSMainCycle 60*
--checkrange [range1[,range2…]]	Specify physical address ranges to check for overlapping (unplausible) objects. The ranges have to be specified in the same format (without whitespaces) as for the *–r* option. For object's whose memory areas overlap error messages are output. Using this option is recommended when generating map file information merged XML databases. Note that specifying this option implicitly does *–enable overlap_check* Example: *--checkrange 0x0-0x100000,0x804000-0x82FFFF*
--checkvirtrange [range1[,range2…]]	Specify virtual address ranges to check for overlapping (erroneous) objects. . The ranges have to be specified in the same format (without whitespaces) as for the *–r* option. For object's whose memory areas overlap error messages are output. Using this option is recommended when generating merged XML databases. Note that specifying this option implicitly does *–enable overlap_check* Example: *--checkvirtrange 0x0-0x8FFFFF,0x902000-0xFFFFFFFF*
--enable [opt1[,[opt2…]]	Enable switchable options: the opt1,opt2 has to be a coma separated list without whitespaces consisting of one or more of the following options: • *normalobj* : enable output of objects physically present • *virtualobj* : enable output of objects declared via comments (virtual objects) • *virtualwriteable* : output virtual objects as writeable entities (option only effective for ASAP2) • *sdlpadding* : pad SDL objects to fit alignment constraints of objects (workaround for bugs in older SDLcompiler versions) • *a2l_module_only* : enable output of partial ASAP2 files consisting of module declaration only. This allows usage of ASAP2 include statements in a

	large project-wide A2L • *a2l_utf-8* : enable output of UTF-8 encoding ASAP2 files (only effective when ASAP2 output is generated) • *a2l_group_by_dir* : Enable generation of groups in ASAP2 output based on directories of objects • *a2l_group_by_file* : Enable generation of groups in ASAP2 output based on file names of objects • *a2l_vtab_prefer_comment* : Prefer using enumerator comments when generating value tables in ASAP2 output. When disabled, the enumerators themselves are output. • *overlap_check* : Enable checking address range(s) for overlaps in objects • *ordered_output* : Enable ordered XML output. Option only effective when outputting XML Databases: it then sorts output to allow easier comparison between different tool versions which might change default ordering of symbols. • *vsmsg* : Enable Visual Studio style line number information in output messages (enable only option) • *tevesmsg* : Enable Continental Teves style line number information in output messages (enable only option) • *gccmsg* : Enable GCC style line number information in output messages (enable only option) • *cc32rmsg* : Enable Renesas CC32R style line number information in output messages (enable only option) Example: *--enable normalobj,a2l_goup_by_dir*
--disable [opt1[,opt2…]]	Disable switchable options: the opt1,opt2 has to be a coma separated list without whitespaces consisting of the options listed for the –enable option. Example: *--disable virtualobj,a2l_utf-8*
--valuesoutput [none|both|only]	The –valuesoutput option influences how comment tags with @values are handled when generating ASAP2 or SDL2.0 output. The option has to be followed by one of the words none, both or only. When 'none' is used, then @values are ignored for output generation. When 'only' is used, then wherever @values declares an alternative layout declaration, then only that is used for output (i.e.: the original C declaration is ignored). When 'both' is used, then as far as possible both the @values declaration and the normal C declaration is output. Note: 'both' is not recommended as it can lead to strange effects due to limitations inherent in both ASAP2 and SDL2.0.

	Example: --*valuesoutput only*

3.1.1 Example command line: merging DAT files with map information

Typically the DAT files created with pdo_scan.exe have to be merged together with map file information into a single large output DAT file for further output generation. A typically command line for doing so (provided we have three DAT files: abs.dat, ayc.dat and tcs.dat, one map file mysw.map and we wish to save output to myout.dat, and also perform a complete overlap check for 32 bit address space):

pdo_tool.exe abs.dat ayc.dat tcs.dat -c mysw.map -f xmldat1.0 --checkrange 0x0-0xFFFFFFFF -o myout.dat

The important thing to note here is that you have to load all DAT files before specifying the map file! When processing the *-c mysw.map* option, all addresses read from the map file will be assigned to the already loaded symbols.

3.1.2 Example command line: generating ASAP2 output

Once a merged single DAT file (for example myout.dat) has been created which contains the necessary address information (i.e.: it has been merged with address information from a map file) one can generate ASAP2 output. First if one wishes to generate all variables without any filtering and default settings:

pdo_tool.exe myout.dat -f asap2 -o myout.a2l

If one only wishes to generate output for those objects that have @allow comments specifying either foo or bar:

pdo_tool.exe myout.dat -f asap2 -a foo,bar -o myout_foobar.a2l

Finally the previous example combined with some fancy grouping according to directory and file name of sources, non-default format specification (10 total digits, 4 fractional digits) and limited ASAP name length (total length 32 at most, with partial names maximal 10 characters long):

pdo_tool.exe myout.dat -f asap2 -a foo,bar --enable a2l_group_by_dir,a2l_group_by_file --asapformatspec 10:4 --asapnamelimit 32:10 -o myout_fancy.a2l

3.1.3 Example command line: generating SDL2.0 output

Using a merged DAT file created (myout.dat) SDL2.0 files can be created. Normally only virtual objects need to be output into SDL files (since physical addresses are not used over MTS systems where SDL2.0 files are used). Also specifying views via the –cycleinfo option is mandatory for SDL2.0. Based on this a typical command line for generating SDL output is:

pdo_tool.exe myout.dat --enable virtualobj --disable normalobj --cycleinfo default,FCT_ENV,EM_ENV ARSMainCycle 60 --cycleinfo FCT_VEH,EM_VEH ARS20Cycle 20 -f sdl2.0 -o myout.sdl

This command will disable output of all normal objects (--*disable normalobj*), while enabling output of all virtual objects (--*enable virtualobj*) into two SDL2.0 views: one

called *ARSMainCycle* with cycle-id *60*, where all objects without @cycleid tags (specified by *default*) and those with @cycleid specified to either *FCT_ENV* or *EM_ENV* are output. The second called *ARS20Cycle* with cycle-id *20*, where all objects with @cycleid set to either *FCT_VEH* or *EM_VEH* are output.
In the second example we only generate output for two C identifiers (*MyVar1* and *MyVar2*):

pdo_tool.exe myout.dat --enable virtualobj --disable normalobj --cycleinfo default,FCT_ENV,EM_ENV ARSMainCycle 60 --cycleinfo FCT_VEH,EM_VEH ARS20Cycle 20 -f sdl2.0 -i MyVar1,MyVar2 -o myout_myvars.sdl

3.1.4 Obsolete textual output

The output format *–f textdat* is only supported for backwards compatibility with 1.x versions of PDO tools. While the current version of the toolset can read these files, ASAP2 or SDL2.0 output generation with the 2.x version of pdo_tool.exe is no longer possible (since the textual DAT file does not store all the necessary allocation information).

3.2 Map file

The map file describing address information loadable via pdo_tool.exe's –c option needs to be a line based text file. Typically map files output by various compilers need to be pre-processed using some scripting to convert them to a format accepted by pdo_tool.exe.

The accepted format needs to be a line-oriented text file, where each line specifies the address and/or type of exactly one object. Thus there shall be no empty lines[30], and each line has to be terminated by a newline (including the last line, which has to be then followed by the end of file). Three fundamental types of lines are accepted:

```
object-identifier address
object-identifier address size
object-identifier address type-information
```

Where `object-identifier` if the name of the symbol for which the address/type information is provided. It is recommended[31] to use directly C identifiers without any compiler decorations, i.e.: underscore or dollar-sign prefixes added by some. The object identifier may either be a top level identifier or that corresponding to a field/element within a struct/union/array. Field/array element specification parts have to be separated by '.' (dots). Field names have to conform to C identifier rules (i.e.: begin with letter or underscore followed by letters/underscores/digits. Array element designators have to consist of a decimal number prefixed and postfixed by an underscore character. For example:

```
ClashTest.SubArray._0_.DOUBLE
```

[30] Since PDO version 2.6 stray empty lines (lines only containing whitespaces) are silently ignored
[31] The lexer tries to automatically guess if underscore or dollar prefixes are used in the map file. For this automatic guessing to work the underscore/dollar prefixing has to be used consistently (i.e.: either used for all or for none).

Specifies the for the global variable `ClashTest`'s field `SubArray`'s 1st (0-th) element's field named `DOUBLE`.

The `object-identifier` needs to be followed by the `address` which will be set for the given object. If the `object-identifier` belongs to a complex object (i.e.: struct, union or array), than the address is taken to be the starting address of the given complex object. The `address` can be specified in hexadecimal with `0x`/`0X` prefixes or a 'h' suffix. If no hexadecimal prefix/suffix is found, then the number is taken to be provided in decimal. For example:

```
0x804000
804000h
8404992
```

All three specify the same address, the first two are interpreted as hex, while the last one is the decimal representation of the address.

If a `size` specification follows the `address` specification, than the same rules apply to it as for the `address` field. Note that the size only acts as a verification check to assure that the internal allocator of PDO determined the same object size as the target compiler did – thus it is not necessarily required, but if the compiler provides the information than it can be used for an additional check.

The most verbose form of map-file can use the `type-information` to specify the exact memory representation of a scalar type[32]. It needs to have one of the below forms:

```
BYTE 0
UBYTE 0
WORD 0
UWORD 0
ENUM
DWORD 0
UDWORD 0
BIT width offset
UBIT width offset
FLOAT 0
DOUBLE 0
```

These specify the layout of the type: if prefixed by '`U`' then unsigned, otherwise signed. '`BYTE`' means byte type, `WORD` means 16 bit integer type, `DWORD` means 32 bit integer type, `FLOAT` means 32 bit IEEE float and `DOUBLE` means 64 bit IEEE float. The types BIT and UBIT indicate a signed/respectively unsigned bit fields which are `width` wide and start at `offset` bit[33].

[32] Complex types can not have this form of type-information as the type-information can only represent basic arithmetic types

[33] Sadly without a reference point this definition is ambiguous: for example bit offset 0, width 1 in byte has mask 0x80 on certain platforms, while it has mask 0x01 on others. The recommended practice is to have bit offset 0 always mean the least significant bit. For backwards compatibility PDO will try to guess which specification it is reading as input.

3.3 Configuration file

Starting from version 2.8 of the PDO-toolchain, the pdo_tool.exe also supports a configuration file. This file has a different syntax from the configuration file for the pdo_scan executable. The file is an ASCII text file, where C and C++ comments can be used. Non-comments shall contain special tool-configuration declarations, which mostly resemble C assignment expressions with the form *option_name* = *value ;*
Note that each such statement has to be terminated by a semicolon.

```
/*
       Configuration file for PDO-Tool
       Template for A2L (ASAP) output

       Note: C and C++ comments are supported
       Each statement has to be terminated by a semicolon ';'
*/

output_format = "ASAP2";      // Supported values: "ASAP2", "XmlDat1.0",
                              // "TextDat", "SDL2.0"
output_encoding = "ISO8859-1";       // Supported values: "ISO8859-1", "CP437",
                                     // "CP852", "CP1252", "UTF-8"
//output_file_name = "testcfg.a2l";  // Output file name setting (only use in special
cases)

default_byte_order = "auto";  // Default byte order in ASAP header
                              // ("motorola"/"intel"/"auto")

asap_proj_name = "BMW_F20";   // ASAP project name string to use for ASAP2 output
asap_module_name = "ARS352";  // ASAP module name string to use for ASAP2 output
asap_module_description = "FRR-02 Full range radar ECU";   // Module description

asap_max_total_name_length = 255;    // The maximum total name length to allow
                                     // in ASAP2 output ([5..1023])
asap_max_partial_name_length = 32;   // The maximum partial name length to allow
                                     // in ASAP2 output ([1..255])
asap_max_matrix_dim = 2;             // ASAP maximum matrix dimensions
asap_virtuals_writeable = disable;   // If virtual objects in ASAP2 output shall be
                                     // writeable (enable/disable)
asap_module_only = disable;   // Output ASAP module only for inclusion in other A2L
asap_group_by_dir = enable;   // Group ASAP2 output by source directories
asap_group_by_file = enable;  // Group ASAP2 output by source file name
asap_vtab_use_comment = enable;      // ASAP value tables should use enumerator
                                     // comments instead of enumerators
asap_name_comment_root = disable;    // Names specified shall be used at root level
                                     // or merely replace field names (enable/disable)
asap_default_format_spec = "%6.3";   // ASAP default format specifier, or "auto"
asap_version = "1.51";               // ASAP version number for output
values_output = "only";       // Values output option for comment @values decalrations.
                              // Supported values: "none", "only", "both"

address_filter = 0x0-0xFFFFF, 0x804000-0x82FFFF;    // Coma seperated address range
                                                    // specifications for addresses
virtual_address_filter = 0x90000-0x901FFF;  // Coma seperated address range
                                            //specifications for virtual addresses
allow_filter = "oem_bmw";     // Coma seperated list of allow IDs to use
identifier_filter = "ANecLatKeep", "Dynamic";       // Come seperated list of
                                                    // identifier filters to use

view_cfg("ARSMainCycle", 60, "default", "cycleid_60");
// View configuration, in the form view_cfg(name, number, coma seperated cycle-ids)
```

Figure 32 Example pdo_tool configuration file

3.3.1 Supported keywords

Keyword	Output File Format(s)	Description
address_filter	A2L, SDL, ECUEXTRACT	Filter the output objects according to their normal addresses (not virtual addresses). Specify a coma separated list of address ranges after this keyword. Only the objects in the ranges specified will be output. Example: *address_filter = 0x0-0x7FFF,0x100000-0x1AB00;*
allow_filter	A2L, SDL, ECUEXTRACT	Filter the output objects according to the identifiers specified after the comment @allow tags. Specify a coma separated list of identifiers after this keyword. If any of the specified identifiers is found along the type tree of an object, then it will be output. Example: *allow_filter = "FCT_developer", "Conti";*
asap_addr_range_objtype	A2L	Keyword used in statements to introduce the ASAP2 specific declaration of which ASAP2 object to use for a given address range. This keyword has to be followed by a parenthesized expression of which ASAP2 measurement object type shall be output (possible values: *characteristic, measurement, readonly characteristic, writeable measurement*). This has to be followed by a coma separated address range list. Example: *asap_addr_range_objtype("readonly characteristic") = 0x0-0xFFFFF,0x900000-0x900FFF;*
asap_default_const_objtype	A2L	Keyword can be used to specify the default ASAP2 measurement object type for constant objects. (Note: asap_addr_range_objtype takes precedence over this keyword, when specified). The keyword has to be followed by *characteristic, measurement, readonly characteristic or writeable measurement*. Example: *asap_default_const_objtype = "measurement";*
asap_default_format_spec	A2L	Used to specify the default formatting string to use for measurement objects. This is used for formatting output for display in the ASAP2 measurement system (see ASAP2 spec for details) Example: *asap_default_format_spec = "%6.3";*
asap_default_non_const_objtype	A2L	Keyword can be used to specify the default ASAP2 measurement object type for non-constant objects. (Note: asap_addr_range_objtype takes precedence over this keyword, when

		specified). The keyword has to be followed by *characteristic, measurement, readonly characteristic or writeable measurement*. Example: *asap_default_non_const_objtype = "characteristic";*
asap_group_by_dir	A2L	Keyword can be used to enable generation of ASAP2 groups based on the directory structure of the parsed input files. Example: *asap_group_by_dir = enable;*
asap_group_by_file	A2L	Keyword can be used to enable generation of ASAP2 groups based on file names of parsed input files. Example: *asap_group_by_file = enable;*
asap_max_matrix_dim	A2L	Can be used to specify the maximum matrix dimensions to use in ASAP2 output. This is useful for certain measurement systems with broken/partial support for ASAP2 matrices. A setting of zero disables ASAP matrix generation. Example: *asap_max_matrix_dim = 2;*
asap_max_partial_name_length	A2L	Specifies the maximum partial name length of identifiers output in the ASAP2 file. ASAP2 version 1.51 and prior maximized this to 32 characters. Certain measurement systems have even stricter limitations. Example: *asap_max_partial_name_length = 20;*
asap_max_total_name_length	A2L	Specifies the maximum total name length of names output in the ASAP2 file. ASAP2 version 1.51 and prior maximized this to 255 characters. Example: *asap_max_total_name_length = 64;*
asap_module_description	A2L	Can be used to specify the module description string for the ASAP2 output. Example: *asap_module_description = “FRR-01 Full Range Radar”;*
asap_module_name	A2L	Can be used to specify the module name for the ASAP2 output. Example: *asap_module_name = “ARS350”;*
asap_module_only	A2L	Option can be used to output the module section of the ASAP2 file only. This by itself is an incomplete ASAP2 can file that however can be included via ASAP2 include mechanisms in a larger project. Example: *asap_module_only = enable;*
asap_name_comment_root	A2L	When enabling this keyword option, then objects with @name tags in their comments will be output at ASAP2 root level. When disabled (the default), then the @name tags only influence the partial identifier at the given hierarchy level. Example: *asap_name_comment_root = disable;*
asap_proj_name	A2L	Option can be used to specify the project

		name of the output ASAP2 file. Example: *asap_proj_name = “F10 Mue”;*
asap_version	A2L	The ASAP2 version number to use for output. Note that this option also influences several features, as certain features (matrices etc.) are only available after a given version. Example: *asap_version = “1.51”;*
asap_virtuals_writeable	A2L	Option for setting if virtual objects output to ASAP2 shall be treated as writeable. The default is off (treated as constants). Note that asap_addr_range_objtype takes precedence over this option. Example: *asap_virtuals_writeable = enable;*
asap_vtab_use_comment	A2L	Option enabling the specification of the brief comment strings of enumerators instead of the raw enumerator identifiers for value tables in ASAP2. Example: *asap_vtab_use_comment = enable;*
check_address_range	A2L, SDL, DAT, ECUEXTRACT	Option for checking a given address range for overlaps between different symbols. Specify a coma separated list of ranges after this option. Example: *check_address_range = 0x0-0xFFFFF, 0x804000-0x82FFFF;*
check_virtual_address_range	A2L, SDL, DAT, ECUEXTRACT	Option for checking a given virtual address range for overlaps between different virtual instances. Specify a coma separated list of ranges after this option. Example: *check_virtual_address_range = 0x20000000-0x2FFFFFFF, 0x80000-0x8FFFF;*
comment_desc_clash	n/a	Option for specifying tool behavior when merging multiple comments description for the same object with different contents. Possible values: ignore, warning, error. Example: *comment_desc_clash = ignore;*
comment_factor_clash	n/a	Option for specifying tool behavior when merging multiple comments scaling factors for the same object with different contents. Possible values: ignore, warning, error. Example: *comment_factor_clash = error;*
comment_logical_error	n/a	Option for specifying tool behavior for inconsistent or logical errors in object comments. (For example, when the minimum value is greater than the maximum value) Example: *comment_logical_error = error;*
comment_max_clash	n/a	Option for specifying tool behavior when merging comments with different maximums for one object. Example: *comment_max_clash = error;*
comment_min_clash	n/a	Option for specifying tool behavior when merging comments with different

		minimums for one object. Example: *comment_min_clash = error;*
comment_name_clash	n/a	Option for specifying tool behavior when merging comments with different names specified for one object. Example: *comment_name_clash = error;*
comment_offset_clash	n/a	Option for specifying tool behavior when merging comments with different scaling offsets for one object. Example: *comment_offset_clash = error;*
comment_unit_clash	n/a	Option for specifying tool behavior when merging comments with different unit strings for one object. Example: *comment_unit_clash = error;*
comment_values_clash	n/a	Option for specifying tool behavior when merging comments with different values expressions for one object. Example: *comment_values_clash = error;*
comment_virtual_clash	n/a	Option for specifying tool behavior when merging comments with different virtual instances/addresses for one object. Example: *comment_virtual_clash = error;*
default_byte_order	DAT	Option controlling the default byte order assumed when reading map files. Possible values: motorola or intel. Example: *default_byte_order = motorola;*
ecuextract_allow_leading_underscore	ECUEXTRACT	Option controlling if leading underscores may be output into EcuExtract files. If disabled, leading underscores are removed. (Needed since some Autosar tools do not support underscores) Example: *ecuextract_allow_leading_underscore = off;*
ecuextract_force_typedefs	ECUEXTRACT	Option controlling how type trees are handled when outputting EcuExtract files. If this option is set to ‘on’, then each alias type along a type tree is generated, potentially generating multiple types with the same content. If set to ‘off’, then only genuinely different types are output to the EcuExtract. Example: *ecuextract_force_typedefs = on;*
ecuextract_max_short_name_len	ECUEXTRACT	Specifies the maximum short name length for outputting EcuExtract files. Different Autosar toolchains have different limitations on this. Example: *ecuextract_max_short_name_len = 32;*
ecuextract_process_bit_fields	ECUEXTRACT	Autosar EcuExtract Files have no proper mechanism for describing bit-fields. When this option is set to ‘off’, then errors will be output when bit-fields are exported to EcuExtract files. When set to ‘on’, then bit-fields will be output as separate record fields, meaning that the generated structs are not binary

		compatible only field-level compatible (meaning fields with the same name occur in the same order, but not exactly at the same offset/bit-mask). Example: *ecuextract_process_bit_fields = on;*
ecuextract_process_pointers	ECUEXTRACT	Autosar EcuExtract files lack the capability to describe pointers. When this option is set to 'off', then errors will be output when pointers are exported to EcuExtract files. When this option is set to 'on', then pointers are exported into EcuExtract as simple integer types. Example: *ecuextract_process_pointers = on;*
ecuextract_process_tagged_obj	ECUEXTRACT	Option controlling if tag-named objects shall be output into EcuExtract files. Example: *ecuextract_process_tagged_obj = off;*
ecuextract_process_unions	ECUEXTRACT	Autosar EcuExtract files lack the capability to describe union-type objects. When this option is set to 'on', then unions are exported as normal records. When this option is set to 'off', then errors will be generated when attempting to export a union. Example: *ecuextract_process_unions = off;*
ecuextract_root_package_desc	ECUEXTRACT	Can be used to specify the root-level package description in the output EcuExtract file. Example: *ecuextract_root_package_desc = "My Autosar package generated";*
ecuextract_root_package_path	ECUEXTRACT	Can be used to specify the root package name in the output EcuExtract. Please use quoted valid identifier. Example: *ecuextract_root_package_path = "MyPackage";*
ecuextract_use_raw_ranges	ECUEXTRACT	Specifies how numeric ranges shall be handled when outputting EcuExtract files. When this option is set to 'on', then the raw binary ranges are set for all fields, maintaining binary compatibility (in this respect). When this option is set to 'off', then the ranges specified by comments and/or the ranges of enumerators are used. Example: *ecuextract_use_raw_ranges = on;*
extended_type_compat	n/a	Option most useful when merging DAT files. Specifies if extended type compatibility rules shall be used (meaning that incompatibilities along type trees are not treated as distinct types). When set to 'off' only completely equal types/symbols are treated as the same (strict checking). Example: *extended_type_compat = on;*
fun_par_name_redef	n/a	Specifies the behavior to exhibit when function parameter names are redefined. Normally this is a benign error, which

		however causes sever problems on certain compilers/architectures. Possible values are 'ignore', 'warning', 'error'. Example: *fun_param_name_redef = error;*
identifier_filter	A2L, SDL, ECUEXTRACT	When outputting ASAP2, SDL2.x or EcuExtract files, the output can be filtered to only contain certain symbols (an their hierarchical sub-structures). Example: *identifier_filter = "AYC_ControlVar", "WheelPulseCounters", "InputVoltages";*
incomplete_fwd_ref	n/a	Specifies the behavior to exhibit when incomplete forward type references are read from DAT files. Possible values are 'ignore', 'warning', 'error'. Example: *incomplete_fwd_ref = error;*
map_file_auto_mirror_bitmasks	n/a	Enables automatic mirroring of bit-masks as work-around for certain (brain-dead) map files. Usually leave this on, and only disable it, when all else fails, as the tool tries to automatically determine if the given map file need mirroring or not. Example: *map_file_auto_mirror_bitmasks = on;*
map_file_auto_multibyte_bit_workaround	n/a	Enables multibyte-bitfield address workaround needed for some map files. Depending on map file generator version, it may store the address of the 'word' containing the given bit-mask, or the lowest address byte touched by the given bit-mask or the highest address byte touched by the bit-mask. It's usually safe to leave this enabled, as the tool tries to autodetect which map file it's dealing with. Example: *map_file_auto_multibyte_bit_workaround = on;*
map_file_extended_compat	n/a	Option controlling map file extended type compatibility. Some map files do not properly respect symbol types and minimize bit-fields based on their width. This is architecturally a major blunder (as most MCUs behave differently when accessing built-in registers with a different access width), but quite common among map-files. It's usually safe to leave this option on, only disable if you're sure your map file is valid in this respect. Example: *map_file_extended_compat = on;*
map_file_mirror_bitmasks	n/a	Option controlling if bit-masks are counted from least significant bit (off) or most significant bit position (on). Example: *map_file_mirror_bitmasks = on;*
map_file_multibyte_bit_workaround	n/a	Option controlling if highest address byte is specified for bit-fields (on) or lowest

		address byte (off). Example: *map_file_update_indices = off;*
map_file_update_indices	n/a	Option controlling if array sizes shall be updated when reading map file. This may be a useful feature if only incomplete declarations were parsed in the C files, which were completed in assembler or another language. Example: *map_file_update_indices = on;*
msg_format	n/a	Specifies which output message format to use. Possible values 'gcc', 'msvc', 'teves'. Each of these has a different line number/file name formatting scheme. Example: *msg_format = "gcc";*
output_encoding	A2L	Specify the output encoding to use when generating A2L files. Supported values: 'UTF-8', 'ISO8859-1'. Example: *output_encoding = "UTF-8";*
output_file_name	n/a	Can be used to specify the output file name. Note: that the directory specified already has to exist. Example: *output_file_name = "..\out\myfile.sdl";*
output_format	n/a	Can be used to specify the output file format. Possible values are 'ASAP2', 'XmlDat1.0', 'TextDat', 'SDL2.0', 'SDL2.1', 'EcuExtract'. Example: *output_format = "SDL2.1";*
scaling_for_floats	A2L, SDL, ECUEXRACT	Option can be used to disable scaling factors for floating point types. This is a workaround for erroneous specifications of scaling factors for floating point values. Example: *scaling_for_floats = on;*
sdl_header_alignment	SDL	Option can be used to specify the alignment value to write into SDL file header. Example: *sdl_header_alignment = 4;*
sdl_unit_enum_str_output	SDL	Output enumeration declrations with special encoding into 'unit' string of SDL file.
sdl_padding	SDL	Option that can be used to automatically add padding fields, where compiler pads data for alignment. Possible values are 'on', 'off'. Example: *sdl_padding = on;*
static_redefinition	n/a	Option controlling reaction to the same static symbol being used in multiple compilation units with different layouts. (Useful when merging databases). Example: *static_redefintion = error;*
symbol_redefinition	n/a	Option controlling reaction when a symbol is redefined with different contents. Possible values: 'ignore', 'warning' or 'error'. Example: *symbol_redefinition = error;*
values_output	A2L, SDL, ECUEXTRACT	Option controlling if sub-structures or enumerations specified in comments with the @values tag shall be output. Example: *values_output = enable;*

view_cfg	SDL	Specify a top-level SDL view element. This has to be followed by a parenthesized expression containing the name of the view, the numerical ID and the list of comment @cycleid tags that shall be associated with the view. Example: *view_cfg("ARSMainCycle", 60, "default", "cycleid_60", "FCT_ENV", "EM_ENV", "RSP_ENV", "ALN_ENV", "FPGA_ENV");* In this example the view's name is 'ARSMainCycle' has the numeric ID 60 and is assigned the virtual objects without any cycle id (special case for the 'default' string) and the cycleids 'FCT_ENV', 'EM_ENV', 'RSP_ENV', 'ALN_ENV' and 'FPGA_ENV'.
virtual_address_filter	A2L, SDL, ECUEXTRACT	Similar in operation to the address_filter keyword, only for virtual addresses. Example: *virtual_address_filter = 0x0-0x8FFFFF,0x902000-0xFFFFFFFF;*

Note that the special keywords 'enable'/'on' and 'disable'/'off' can be used for binary options (see examples in above table).