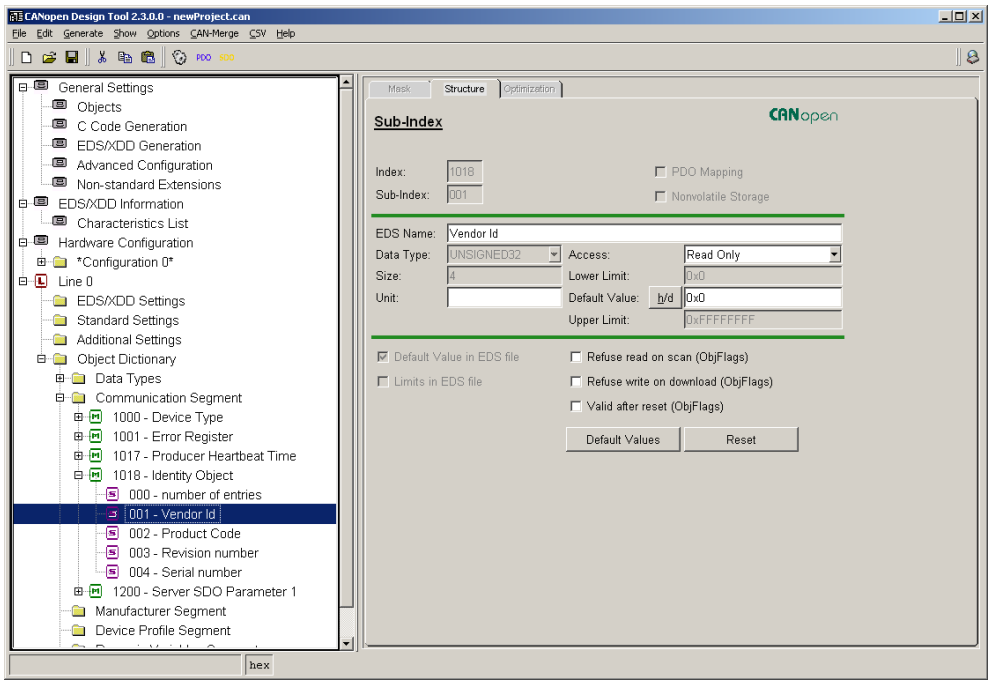
CANopen **Design Tool**

User Manual



Disclaimer
All rights reserved

The programs, boards and documentations supplied by *port* GmbH are created with due diligence, checked carefully and tested on several applications.

Nevertheless, *port* GmbH can not take over no guarantee and no assume del credere liability that the program, the hardware board and the documentation are error-free respective are suitable to serve the special purpose.

In particular performance characteristics and technical data given in this document may not be constituted to be guaranteed product features in any legal sense.

For consequential damages, which are emerged on the strength of use the program and the hardware boards therefore, every legal responsibility or liability is excluded.

port has the right to modify the products described or their documentation at any time without prior warning, as long as these changes are made for reasons of reliability or technical improvement.

We are thankful for hints of possible errors and may ask around for an information.

We will go all the way to verify such hints fastest

Copyright

© 2019 *port* GmbH
Regensburger Straße 7
D-06132 Halle
Tel. +49 345 - 777 55 0
Fax. +49 345 - 777 55 20
E-Mail service@port.de
Internet http://www.port.de

Table of Contents

1. Abbreviations . . . 7
2. References . . . 9
3. Introduction . . . 11
3.1. Product overview . . . 11
3.2. Product structure . . . 11
3.3. System requirements . . . 13
3.4. Installation . . . 13
3.5. Support by *port* . . . 14
4. Device structure . . . 15
5. File structure . . . 17
5.1. DT project file . . . 17
5.2. Profile databases . . . 17
5.2.1. profile304_july_2010.pro . . . 18
5.2.2. profile443_v2_1_0.pro . . . 18
5.3. Generated files . . . 20
6. Graphical user interface . . . 21
6.1. Menu . . . 21
6.1.1. File . . . 21
6.1.2. Edit . . . 21
6.1.3. Generate . . . 21
6.1.4. Show . . . 21
6.1.5. Options . . . 21
6.1.5.1. View Options . . . 21
6.1.5.2. Generation Options . . . 22
6.1.5.3. Import Options . . . 23
6.1.6. Help . . . 23
6.2. Toolbar . . . 23
6.3. Project tree . . . 24
6.3.1. General settings . . . 24
6.3.1.1. Objects . . . 25
6.3.1.2. C Code Generation . . . 25
6.3.1.3. EDS/XDD Generation . . . 25
6.3.1.4. Advanced Configuration . . . 25
6.3.1.5. Non-standard Extensions . . . 25

6.3.2. EDS/XDD Information . . . 25
6.3.3. Hardware Configuration . . . 25
6.3.4. Line . . . 26
6.3.4.1. Object Dictionary . . . 26
6.3.4.1.1. Sub-segmentation . . . 26
6.3.4.2. Standard Settings . . . 28
6.3.4.3. Additional Settings . . . 28
6.3.4.4. EDS/XDD Settings . . . 28
7. Project editing . . . 29
7.1. Beginning a project . . . 29
7.2. Hardware configuration . . . 29
7.3. Application variables . . . 30
7.4. Parameterization . . . 31
7.5. Generation of source code . . . 31
7.6. %-Variables . . . 31
7.7. {}-Expressions . . . 32
8. Communication services . . . 33
9. Special cases . . . 35
9.1. Range checking for variables . . . 35
9.2. Grouping of indices in sub-segments . . . 35
9.3. Predefined symbols . . . 35
9.4. Limitations . . . 36
10. Complex data types . . . 37
11. Object-specific callback functions . . . 39
12. Manufacturer-specific object descriptions . . . 41
12.1. Format descriptors . . . 41
12.2. Example for html-documentation . . . 44
12.3. Example for tcl-scripts . . . 45
12.4. Example for rtf documents . . . 45
12.5. Example for csv-files . . . 46
13. CAN-Merge PlugIn . . . 49
13.1. Project View . . . 49
13.2. Detail View . . . 50
14. Extension module: csv-import . . . 51
14.1. Structure of the csv-file . . . 51
14.2. Input parameters for the csv-import . . . 52

14.2.1. Line 52
14.2.2. Index offset 52
14.2.3. csv-file 53
14.2.4. Header in csv-file 53
14.2.5. csv-specifier line 53
14.3. csv-specifiers 53
14.4. csv-separator 64
14.5. Limitations 65
15. Non-standard extensions 67
15.1. Application-specific cob-IDs 67
15.2. Application-specific handling of NMT messages 68
15.3. Complete profile-specific data type index range 68
15.4. Complex data types in EDS 68
Index 70

1. Abbreviations

CAN	Controller Area Network
CiA	CAN in Automation international users and manufacturers group e.V.
COB	Communication Object (CAN Message)
COB-ID	Communication Object Identifier
CSDO	Client SDO
csv	comma separated value files
DT	Design Tool
EDS	Electronic Data Sheet
EMCY	Emergency Object
ESC	EtherCAT Slave Controller
ESI	EtherCAT Slave Information
ETG	EtherCAT Technology Group
GUI	graphical user interface
MPDO	Multiplex PDO
NMT	Network Management
PDO	Process Data Object
SDO	Service Data Object
SSDO	Server SDO
SYNC	Sychronization Object
TIME	Time Stamp Object
XDD	XML device description

XML	eXtensible Markup Language

2. References

/CiA-301/ CiA standard 301 "Communication profile", V4.2

/CiA-306-1/ CiA standard 306-1 "Electronic device description, EDS and DCF", V1.3.5

/CiA-311/ CiA standard 311 "CANopen device description, XML", V1.1.0

/RFC 4180/ Network Working Group, Request for Comments: 4180
Y. Shafranovich, SolidMatrix Technologies Inc., October 2005

3. Introduction

3.1. Product overview

The CANopen Design Tool of *port* is a software tool for the development of CANopen devices. It manages properties, hardware configurations and the object dictionary of the CANopen device. In the result the CANopen Design Tool generates:

- C code for the configuration, optimization and initialization of the Library,
- the object dictionary as C code,
- electronic device descriptions and
- various documentations.

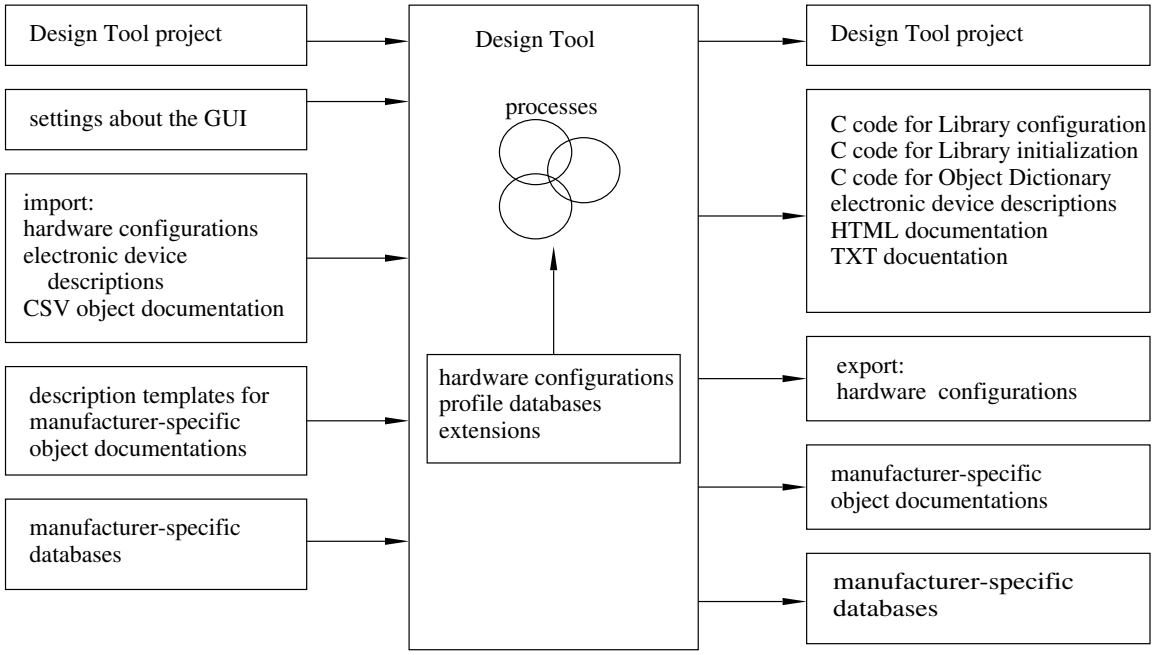


figure 1: Design Tool

With the CANopen Design Tool an instrument is available which frees the developer of error-prone activities repeating itself. It ensures consistency of implemented functionality, electronic device descriptions and device documentations.

The created object dictionary supports numerous options of the CANopen Library from *port*. Such as several CAN-lines (multiple lines) and segment structuring. A tree representation of all implemented parameters and data eases the maintenance of device software. With the CANopen Design Tool the beginning with the CANopen protocol is less difficult and the development of a device is accelerated.

The CANopen Design Tool is available in two variations:

- **full edition and**
- **demo edition.**

3.2. Product structure

functionality	DT demo version	DT full version
availability	free	licensed
maximal number of main-indices	15	all
maximal number of sub-indices	65	all
generation of Library configuration	x	x
generation of C-code	x	x
generation of EDS	x	x
generation of XDD	x	x
generation of documentation in html-format	x	x
generation of documentation in txt-format	x	x
generation of manufacturer-specific object descriptions	x	x
profile database 301	x	x
profile database 302	x	x
profile database 304	—	licensed
profile database 401	—	licensed
profile database 402	—	licensed
profile database 404	—	licensed
profile database 405	—	licensed
profile database 406	—	licensed
profile database 410	—	licensed
profile database 417	—	licensed
profile database 418	—	licensed
profile database 419	—	licensed
profile database 443	—	licensed
profile database 447	—	licensed
profile database 452	—	licensed
extension csv-import	—	licensed
extension CAN-merge	—	licensed
extension no global objects	—	licensed

table 1: product structure

The profile databases include the object specifications of the suitable CiA standard for import. The scope of delivery of the CANopen Design Tool comprises:

- software of the CANopen Design Tool
- user manual
- profile databases
- extensions

The CANopen Design Tool is available on the web-site of *port* for download under
Products / CANopen / Tools / CANopen Design Tool.
All licensed parts of the CANopen Design Tool must be activated by a valid license before usage.

3.3. System requirements

The CANopen Design Tool runs on PC's with Microsoft Windows™ or Linux with:

RAM:	512 MByte
Hard-disk Space:	45 MByte

3.4. Installation

For installation do the following steps:

1. Unpack the archive in zip-format.
2. for Windows™: Execute the setup program **setup.exe.**
   for Linux: Unpack the archive in tgz-format.

After installation you will find the directory structure shown in figure 2 with:

- profiles: contains all profile databases for import
- examples: contains sample files
- manual: contains the user manual
- help: only for private usage of the CANopen Design Tool

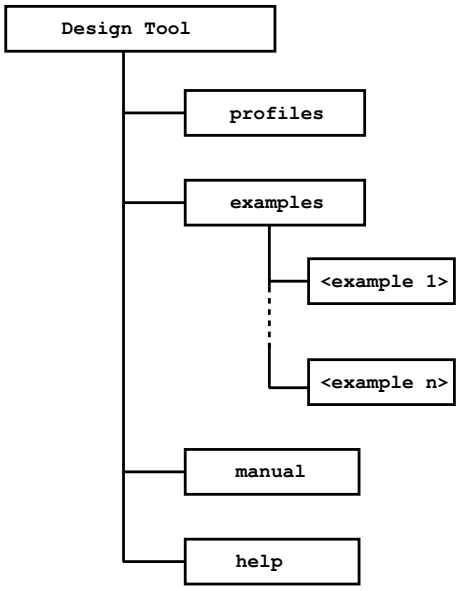


figure 2: directory structure of the CANopen Design Tool

On Windows™ systems the CANopen Design Tool can be started by the icon on the desktop or via the start menu.

On Linux systems the CANopen Design Tool can be started by calling the program **designtool.**

3.5. Support by *port*

Please contact us for sales via

e-mail:	**service@port.de**
phone:	**+49 345 777 55 - 0**
fax:	**+49 345 777 55 - 20**

Please contact us for technical support via

e-mail:	**support@port.de**

The engineers at *port* will give you some assistance as soon as possible.

port provides consultations in the whole field of CAN e.g. network planning, network configuration, message distribution, selection of devices and CANopen Profile implementations.

4. Device structure

Each DT project specifies a field device with 1 - 127 CANopen devices /CiA-301/. Each CANopen device is connected with one CANopen network. That means each CANopen device is assigned to another CAN line.
If the field device only supports 1 CANopen device, the field device is a single-line device. In the other case the field device must be a multi-line device.

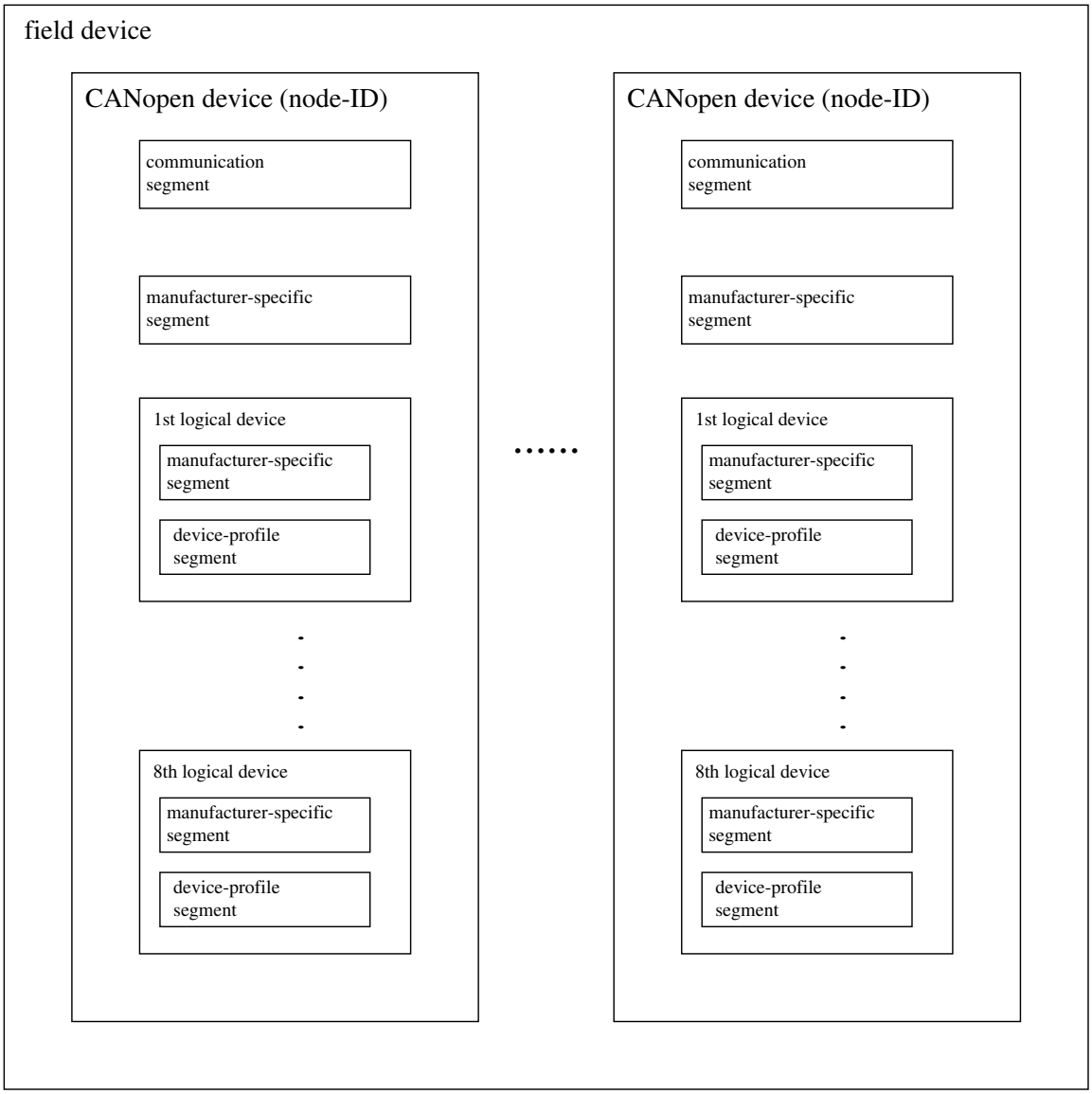


figure 3: device structure

Each CANopen device can have 1-8 logical devices.

5. File structure

5.1. DT project file

All information of a DT project is stored in the DT project file. Nevertheless it is useful to have a working directory for each DT project, because the generated files will be stored in the working directory of the DT. The file extension for the CANopen Design Tool is ".can".

5.2. Profile databases

Profile databases contain standardized objects with their attributes and allow implementations in shortest time. The CANopen Design Tool is delivered with profile databases for communication objects according to /CiA-301/ and /CiA-302/ and for safety-related communication according to /CiA-304/.

If communication objects are set up with the CANopen Design Tool, the objects will be loaded automatically from the profile databases. Additionally various profile databases to profile device standards are available at *port*, see table 1 and table 2.

Objects within segments in the project tree can be exported to own profile databases. The objects from all profile databases can be imported. The generation with objects from profile databases, created by *port*, fails without a valid license for the suitable profile database.

profile database file	**reference to standard**	
	version	**content**
profile301.pro	CiA-301, V4.2.0	communication profile
profile302.pro	CiA-302, part 2-6, V4.1.0	additional application layer
profile304.pro	CiA-304, V1.0.1	safety-related communication
profile304_july_2010.pro	EN 50325-5, July 2010	
profile401.pro	CiA-401, V2.1	generic I/O modules
profile402.pro	IEC 61800-7, December 2007 (CiA-402-2, V3.0.0)	drives and motion control
profile402_2_v4_1_0.pro	CiA-402-2, V4.1.0	
profile404.pro	CiA-404, V1.2	measuring devices and closed-loop controllers

profile database file	reference to standard	
	version	content
profile405.pro	CiA-405, V1.0	IEC 61131-3 programmable devices
profile406.pro	CiA-406, V3.2 + V3.1	encoder
profile410.pro	CiA-410, V1.1	inclinometer
profile417-3.pro	CiA-417-3, V2.0	lift
profile417-4.pro	CiA-417-4, V2.0	
profile418.pro	CiA-418, V1.0.1	battery modules
profile419.pro	CiA-419, V1.0.1	battery chargers
profile443_v2_1_0.pro	CiA-443, V2.1.0	SIIS level-2 devices
profile447_v2_0_0.pro	CiA-447, V2.0.0	car add-on devices
profile452_v1_0_0.pro	CiA-452, V1.0.0	PLCopen motion control

table 2: overview about profile databases

The following sub-chapters give additional information to some profile database files.

5.2.1. profile304_july_2010.pro

The profile database for the CiA-304 provides the profile-specific objects for

- the SRDO service and
- the GFC service.

Some objects uses different default values dependig on the information direction. The default values of these objects are set to invalid, most on value 0.

5.2.2. profile443_v2_1_0.pro

The profile database for the CiA-443 provides the profile-specific data types and objects with except for object 6010h (p443_cpu_clock_time_and_date).

The profile-specific data types are dependent on the number of channels or valves which can be different for various objects. The Design Tool generates special C structures for the sub-indices which are depending on channels or valves to provide an effective access on the objects in C code. All structure definitions are stored in the generated file objects.h.

```
Example: data type 0082h (12-CHANNEL METAL LOSS)

The data type is compound by general and channel-dependent sub-indices.
The following C structure for the channel-dependent sub-indices is generated:
```

```
typedef struct {
    UNSIGNED8  status;                  /**< channel status */
    REAL32     averageMetalLoss;        /**< channel average metal loss */
    REAL32     averageTemperature;      /**< channel average temperature */
    REAL32     sector_1_metalLoss;      /**< channel sector 1 metal loss */
    REAL32     sector_1_temperature;    /**< channel sector 1 temperature */
    REAL32     sector_2_metalLoss;      /**< channel sector 2 metal loss */
    REAL32     sector_2_temperature;    /**< channel sector 2 temperature */
    REAL32     sector_3_metalLoss;      /**< channel sector 3 metal loss */
    REAL32     sector_3_temperature;    /**< channel sector 3 temperature */
    REAL32     sector_4_metalLoss;      /**< channel sector 4 metal loss */
    REAL32     sector_4_temperature;    /**< channel sector 4 temperature */
    REAL32     sector_5_metalLoss;      /**< channel sector 5 metal loss */
    REAL32     sector_5_temperature;    /**< channel sector 5 temperature */
    REAL32     sector_6_metalLoss;      /**< channel sector 6 metal loss */
    REAL32     sector_6_temperature;    /**< channel sector 6 temperature */
    REAL32     sector_7_metalLoss;      /**< channel sector 7 metal loss */
    REAL32     sector_7_temperature;    /**< channel sector 7 temperature */
    REAL32     sector_8_metalLoss;      /**< channel sector 8 metal loss */
    REAL32     sector_8_temperature;    /**< channel sector 8 temperature */
} P443_CHANNEL_METAL_LOSS_T;
```

```
The C structure for data type 0082 with 2 channels is:
```

```
typedef struct {
    UNSIGNED8       numOfEntries;       /**< number of entries */
    UNSIGNED32      siUnitMetalLoss;    /**< SI unit for metal loss */
    UNSIGNED32      siUnitTemperature;  /**< SI unit for temperature */
    P443_CHANNEL_METAL_LOSS_T chan[2];  /**< channels */
} P443_12_CHANNEL_METAL_LOSS_2_T;
```

The condition for the generation of channel-dependent data types is the setting of the profile number in object 1000h (p301_device_type) or in object 67FFh (single_device_type) for multiple logical devices.

The names of the C structure elements are fixed.

The C names of the P443 objects on tab *Structure* for the main-index have to be modified by a repeated import without usage of sub-segmentation.

Limitations:

- object 6010h (p443_cpu_clock_time_and_date) is not supported because the DT does not provide the data type TIME_OF_DAY
- the profile database P443 is not available for non-global variables

Please contact *port* if you need any of the limited functions, see chapter 3.5.

5.3. Generated files

An overview about the generated output files gives table 3.

file	description
cal_conf.h	configuration file for the Library in C code
objects.h	Object dictionary implementation in C code
objects.c	Object dictionary implementation in C code (optional)
co_init.c	initialization file for the Library in C code
<edsFileName>.eds	EDS file for a CAN line
<edsFileName>.xdd	XDD file for a CAN line in xml-format
<edsFileName>.txt	description file for the CANopen Device Monitor
<projectName>_docu.txt	documentation of implemented objects in txt-format
<projectName>.html	documentation of the DT project in html-format
<manufacturer-specific>	manufacturer-specific object descriptions (optional)
generate.err	information file with error and warning messages

table 3: generated files

6. Graphical user interface

The CANopen Design Tool is controlled via

- menu,
- toolbar and
- project tree with masks on tabs.

Information in html-format are indicated in the standard browser.

6.1. Menu

6.1.1. File

The menu *File* controls the DT project files and terminates the DT.

6.1.2. Edit

The menu *Edit* allows to copy, cut, paste, delete and duplicate objects in the project tree.

6.1.3. Generate

The menu *Generate* provides the generation process of the output files depending on the setting in the menu *Generate* > *Generate Documentation* and the settings in the menu *Options* > *Generation Options*. The content of the output files can be specified about the settings in the project tree, see chapter 6.3.

It is possible to execute application-specific programs before and after generation. The commands can be entered about the menu *Generate* > *Pre-Generation Command* and *Generate* > *Post-Generation Command*. The command can be a shell-script or a batch file, or you can call an executable file, e.g. the EDS checker, make all. The output of the command is written into the information file "generate.err".

6.1.4. Show

The menu *Show* allows to indicate all generated files.

6.1.5. Options

The menu *Options* provides configuration settings related to the DT itself.

6.1.5.1. View Options

EDS names in tree:
This option determines the indication of the object names.

on: The EDS names of the objects are indicated in the project tree.

off: The C names of the objects are indicated in the project tree.

Use mask view:
This option determines the active tab after selection of an object in the project tree.

on: Tab Mask is indicated first.

off: Tab Structure is indicated first.

Hide EDS flags on tab Structure:
This option controls the indication of the EDS settings:

- *Default Value in EDS file*
- *Limits in EDS file*
- *Refuse read on scan*
- *Refuse write on download*
- *Valid after reset*

on tab *Structure* when the sub-index of an object is selected in the project tree.

on: The settings are not indicated and can not be changed. But the configured values of the settings are used for generation.

off: The settings are indicated and can be changed.

Font settings:
The font of views can be configured.

Expert Mode:
Properties of objects in the *Communication Segment* of the project tree are restricted to avoid settings which violates CiA standards. This constraint can be removed about this option.
Modifications made in the Expert Mode are not reset if the Expert Mode is switched off. It is possible to load the default settings for the object from the profile database about tab *Structure* > button *Default Values*.

on: The Expert Mode is active. Object properties can be adapted to special application requirements. **ATTENTION! It is possible that the modified object properties violates standards and the CiA Conformance Test is not passed.**

off: The Expert Mode is deactive.

6.1.5.2. Generation Options

Check objects:
This options allow to disable the checking of objects before generation.

on: The object dictionary is checked before generation. This setting is

recommended.

off: The object dictionary is NOT checked before generation. Errors in the object dictionary can cause a crash of the DT. This setting shall only be used for large and approved object dictionaries.

Generate objects.c:
This option determines the location of the generated object dictionary.

on: The object dictionary is stored in the file objects.c. Extern declarations are stored in objects.h.

off: The object dictionary and the extern declarations are stored in objects.h.

Manufacturer-specific object description:
About this option the user can configure up to 3 description templates and the desired documentation files. For more details see chapter 12.

6.1.5.3. Import Options

Delete HW Configurations for EDS-import:
This option configures the deletion of all hardware configurations in the object tree during the EDS-import initiated about the menu File > Import EDS/XDD file.

on: All Hardware Configurations are deleted (by default).

off: No Hardware Configuration is deleted.

6.1.6. Help

The menu *Help* provides:

- context help to the selected view,
- DT user manual
- overview about available compiler defines,
- routine for installing license,
- actual version of the DT and
- information to the newest available DT version.

6.2. Toolbar

The toolbar below the menu provides a fast access to:

- control functions for DT project files
- editing functions for the project tree
- generation process
- addition of the communication services PDO and SDO

- query of the newest available DT version

figure 5: Toolbar

6.3. Project tree

The project tree represents all information to the field device. About button <F1> the online help for each view is available.

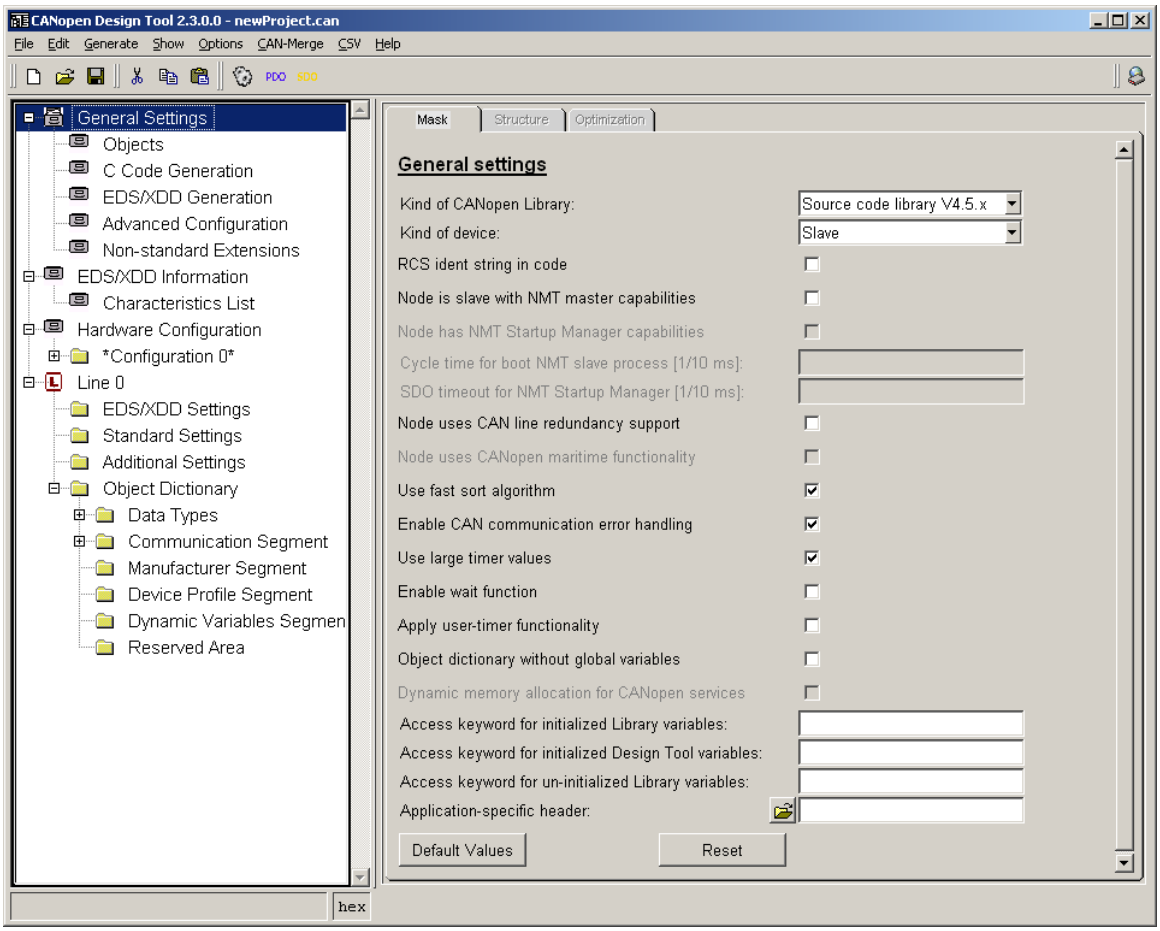


figure 6: project tree

6.3.1. General settings

The general settings makes common configurations related to the field device.

Kind of device:

This setting implies the configuration of the Library which shall be used for this field device related to the NMT capabilities.

6.3.1.1. Objects

The usage of object-specific callback functions is described in chapter 11.

6.3.1.2. C Code Generation

These options determine code-specific properties to allow the integration of the Library into application code.

6.3.1.3. EDS/XDD Generation

These options allow the usage of optional information in the EDS/XDD file according to /CiA-306-1/ and /CiA-311/.

6.3.1.4. Advanced Configuration

This mask allows to make device-specific definitions. These definitions have to be written in C syntax and are taken over into the configuration file for the Library.

6.3.1.5. Non-standard Extensions

In some less cases it can be necessary to violate definitions in the CiA standards to realize special application-specific requirements. This mask makes some exceptions available, see chapter 15.

6.3.2. EDS/XDD Information

Information about the manufacturer and the device are necessary for the header of the EDS and XDD file.

6.3.3. Hardware Configuration

The DT provides the possibility to configure more than one hardware configuration for the field device. This can be useful when different hardware environments are used during development, test and production for instance. But only one hardware configuration can be active.

The hardware configuration is generated to the configuration file cal_conf.h and is used by the CANopen driver.

It is configurable to generate the C code settings for all hardware configurations or only for the active hardware configuration about the menu *General Settings* > *C Code Generation* > *Generate only active hardware configuration*. In the case that the C code settings for all hardware configurations are generated only the settings for the active hardware configuration is valid during compilation of the application code.

A field device can work in different CANopen networks at the same time. Such multi-line devices can be gateways for instance. Each network line needs their own CAN configuration while the CPU and compiler settings are valid for the complete field device. For multi-line devices each *Hardware Configuration* has to include CAN settings for each line. *CAN Settings 0* refers to *Line 0* etc.

6.3.4. Line

The *Line* item in the project tree represents a network and includes line-dependent settings and the object dictionary of the device for the line.

6.3.4.1. Object Dictionary

The object dictionary specifies all needed objects and their properties and is grouped in the following segments:

- Data Types: contains all data types in the index range 0001h - 0FFFh
- Communication Segment: contains all objects for communication services in the index range 1000h - 1FFFh
- Manufacturer Segment: contains manufacturer-specific objects in the index range 2000h - 5FFFh
- Device Profile Segment: contains device profile objects in the index range 6000h - 9FFFh
- Dynamic Variables Segment: contains standard network variables in the index range A000h - AFFFh
- Reserved Area: the index range B000h - FFFFh are reserved, system variables are not supported

6.3.4.1.1. Sub-segmentation

Inside the Manufacturer Segment and the DeviceProfile Segment it is possible to create virtual or real sub-segments.

Virtual sub-segments are used to structure the graphical user interface and for the creation of own sub-segment-specific profile data bases.

Real sub-segments are provide the access about named array variables additionally. This access to variables in C code can be useful for devices with identical logical devices:

```
device[0].control = 0x1234;
device[1].control = 0x5678;
```

The indexing on the named array variables starts always with 0 in C code style. The data type of the named array variables is a generated C structure representing all objects inside the sub-segment:

```
typedef struct {
    UNSIGNED16 control;
} CO_OD_LINE0_devSub_T;

CO_OD_LINE0_devSub_T device[2];
```

All sub-segments are specified by the following properties:

property	description
Name	logical name used in the object tree
Variable	name of the sub-segment in C code `devSub` in the example above
Start	first object index inside the sub-segment
Length	number of object indices inside the sub-segment
Constant description structure	The descriptions of all objects in the sub-segment are stored in the program memory.
Virtual segment	virtual sub-segments are only indicated in the object tree and are excluded by C code generation

Add a sub-segment: A new sub-segment can be created about tab *Mask* > *Add new Sub-Segment* after selection of the Manufacturer Segment or the DeviceProfile Segment.

Configure a sub-segment: An existing sub-segment can be configured about tab *Mask* > *Configure Sub-Segment* after selection of the sub-segment.

Delete a sub-segment: An existing sub-segment can be removed about menu *Edit* > *Delete* after selection of the sub-segment.
ATTENTION! - All objects inside the sub-segment are also deleted!

The usage of sub-segments bases on the following rules:

1.rule:
A sub-segment must be located completely inside the Manufacturer Segment or the DeviceProfile Segment.

2.rule:
The index range of a sub-segment must not overlaps the index range of an other sub-segment.

3.rule:
The Variable name of sub-segments inside the Manufacturer Segment must not be used for sub-segments in the DeviceProfile Segment and vice versa.

4.rule:
All sub-segments with the same Variable name must be real or virtual and have to include the same objects shifted by an index offset.

5.rule:
All sub-segments with the same Variable name must be constant or non-constant.

An example is in the DT installation directory > examples > sub_segmentation.

6.3.4.2. Standard Settings

Some object properties can be changed about *Standard Settings / Global Settings for Object <tab>* for all objects of the line. The configuration about the *Standard Settings* has the higher priority against the object-specific setting, i.e. any change of these standard settings overwrites the object-specific setting for all objects. Object-specific settings can only be made if the change is allowed by activation of *Standard Settings / <property> / Local Changeable*. The object-specific settings are located on tab *Optimization* for main-indices and on tab *Structure* for sub-indices.

Kind of Node specifies if the line shall have NMT master or NMT slave capabilities.

6.3.4.3. Additional Settings

These settings provide the configuration of line-dependent special functions of the Library and/or extra packages to the Library.

6.3.4.4. EDS/XDD Settings

This item includes line-dependent information for the EDS and XDD file.

7. Project editing

This chapter describes the working flow for creating and editing projects. The order of the flow is not mandatory, but very useful. Basic parameter like the number of CAN lines and the kind of the device (Slave or Master) shall be clear at the beginning of the development. The necessary steps are the following:

- configuration of global parameter
- configuration of global ESI resp. EDS parameter
- configuration of the hardware settings
- configuration of CAN line specific EDS parameter
- configuration of standard and additional settings
- definition of application variables
- parameterization of application variables
- definition of communication variables
- parameterization of communication variables
- optimization of each object if needed
- generation of outputs

7.1. Beginning a project

A project is created by the menu *File* > *New Project*.
Existing projects are opened by the button *Open Project* or the menu *File*.
Each CAN line can be preset with data from a profile or an EDS file.

7.2. Hardware configuration

At first the target hardware has to be configured. The most important decision is to choose a CPU resp. an operating system.
Default configurations for the supported platforms can be found at the s1-example of the CANopen Library. These configuration files (`conf_xxxx.h`) can be imported via "Import Configuration".

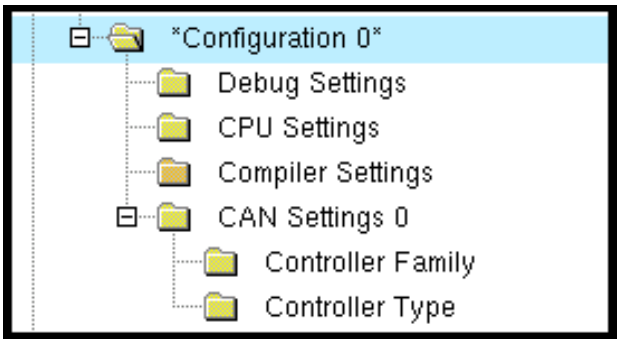


Figure 7, hardware configuration part of the project tree

If the CPU is set, the other CPU settings are set to CPU-specific default values. These default values are suitable in the majority of cases. At *Compiler Settings* the used compiler can be selected. For each line the CAN-controller or the PC-CAN-Interface must be configured. This can be done at *CAN Settings*. There the CAN controller family must be chosen at first and then the particular CAN controller type can be set. Detailed information about each attribute at these form can be found at the context help.

If the application shall be used on different hardware plattforms, more than one hardware configuration can be created. If no configuration is marked as active, the define `CONFIG_USE_TARGET_x` must be set to 1 in the Makefile or in the compiler project.

7.3. Application variables

Definition of application variables is the creation of objects with indices in the range of $2000_h - 5FFF_h$ for manufacturer-specific objects or in the range of $6000_h - 9FFF_h$ for standardized device-profile objects. There are two ways for the definition:

- loading from a profile database and
- creation by hand

Certain objects can be selected by means of their index from a profile database. Databases for the standardized CANopen device profiles are available from *port*. If a database does not yet exist, please contact *port* (see chapter 3.5). All non-standardized objects and not provided objects by a database can be created by hand.
The main-index defines the structure and properties for all sub-indices of the object.

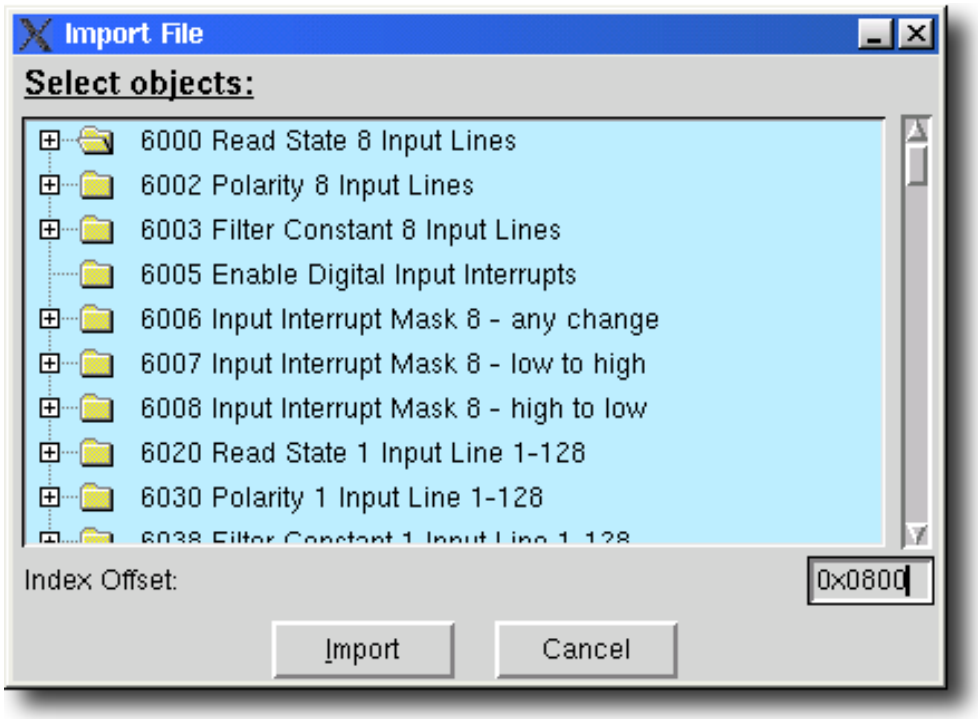


Figure 8, import mask for profiles

7.4. Parameterization

For parameterization of complex communication variables
the mask view can be used.

7.5. Generation of source code

About the button *Generate* all files listed in chapter 5.3 are generated.

This method garantees that all C code and documentation files are consistent to each other.

The file **generate.err** contains warnings and errors. It is recommended to check this file after each generation.

7.6. %-Variables

%-variables can be used in EDS Names and C Names of objects. In the file **objects.c**, in the EDS files, in the documentation and also in the object tree the %-variables are replaced by their current values.

%-variable	description
%i	number of the index
%s	number of the sub-index
%l	number of the line
%f	number of the service (SSDO, CSDO, RPDO, RPDO-Mapping, TPDO-Mapping, SRDO, SRDO-Mapping) starting at 1
%a	number of the service starting at 0
%p	(index - start of segment) starting at 0
%q	(index - start of segment + 1) starting at 1
%u	(index - start of sub segment) starting at 0
%v	(index - start of sub segment + 1) starting at 1
%t	short name of data type
%k	number of sub-segment within a segment starting at 1

Table 5, %-variables

For numerical object settings the %-variable written in lower case is replaced by the decimal value, otherwise the %-variable is replaced by the hexadecimal value without leading "0x". %-Variables that are undefined in the current context return an empty string, e.g. %s at an main-index.

Examples for the usage of these %-variables can be found at the SDO or PDO objects in the communication profile.

7.7. {}-Expressions

{}-expressions allow the usage of %-variables and mathematical operators in EDS Names and C Names of objects. Inside of the braces %-variables, constants (decimal/hexadecimal) and the operators + - * and / can be used.
Examples for valid object names with {}-expressions:

- output {%i - 0x100}
- state {%p + 100} at device %l

8. Communication services

The CANopen protocol specifies various communication services. The most communication services are optional. The CANopen Library can be optimized, so that only the code for the required CANopen services is implemented.

Communication services shall be implemented about the project tree by Line / Object Dictionary / Communication Segment / tab Mask / Add Communication Service.

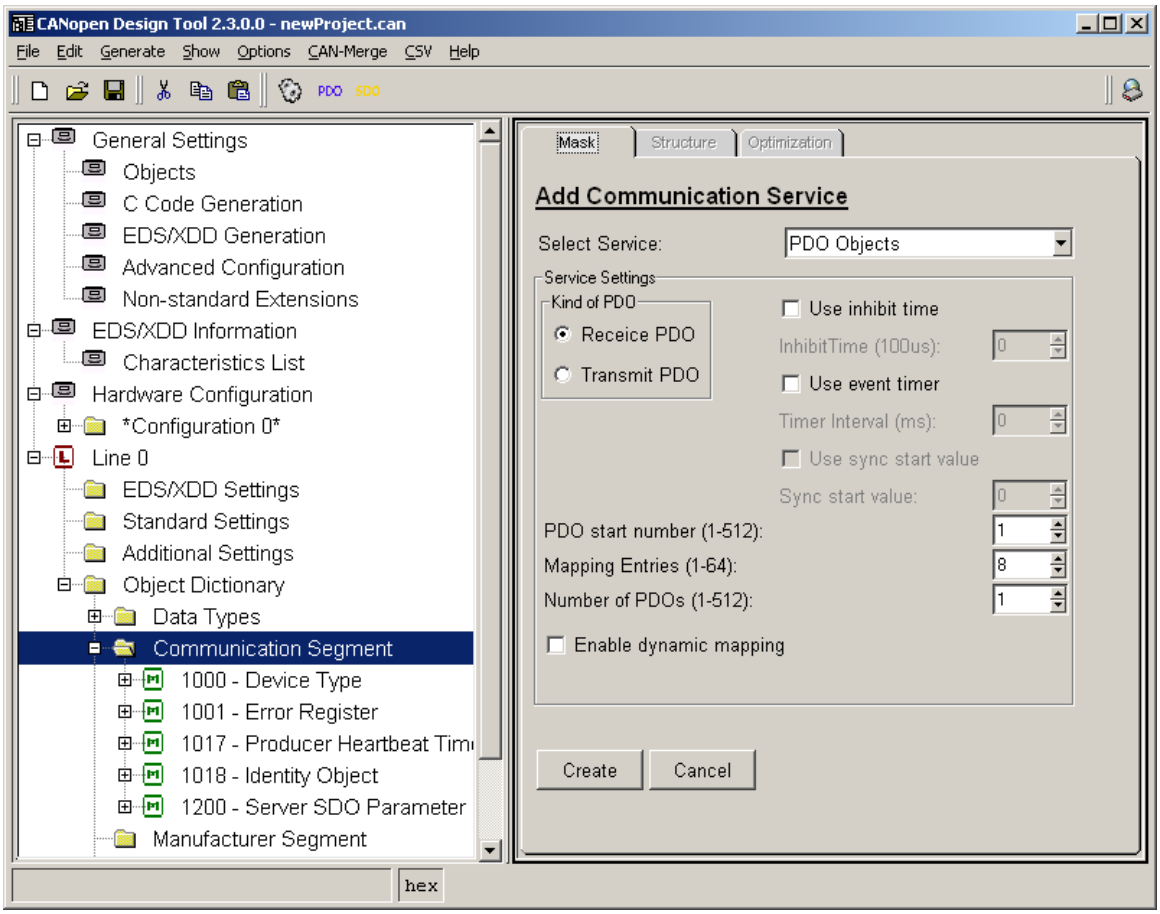


figure 9: dialog "Add new communication service"

The dialog allows to configure the selected communication service among other things the implementation of optional communication objects. During creation of the communication service the CANopen Design Tool makes all necessary settings and loads all necessary communication objects from the profile database of the newest available version.

Communication objects can be also added:

- by loading from profile databases about the project tree by
  Line / Object Dictionary / Communication Segment / tab Mask / Import Data from File or
  Line / Object Dictionary / Communication Segment / tab Mask / Add new Index,

- by import from csv-file (see chapter 14) or
- by import from EDS/XDD file.

Please note that the CANopen Design Tool only creates the objects in these cases and does not create the communication service, i.e. if the communication service was not created before it is possible that compiler defines for the CANopen Library are not set correctly.

Objects according to older standard versions can be loaded from the profile databases suitable to the desired standard version. If there is no suitable profile database available, please contact *port* (see chapter 3.5).

9. Special cases

This chapter describes some special cases for object dictionary generation. The CANopen Design Tool supports the generation of object dictionary implementations for the 'single'-line release and the 'multiple'-line release of the CANopen library of *port*. The following special cases are used for the optimization of the implementation.

9.1. Range checking for variables

Entries are set up as a default during generation of the object dictionary for minimum and maximum values. These limit values are used of the CANopen library by port during SDO transfer for limit supervision. The limit monitoring can be switched on/off via object 1200h / tab Mask / checkbox "Check Object Limits with SDO Transfer". This setting activates or deactivates the compiler define **CONFIG_LIMITS_CHECK**.

9.2. Grouping of indices in sub-segments

A further characteristic is the organization of device variables within structures. Such an organization is possible for the manufacturer data and for data of the standardized device profiles. This is convenient if identical entries in the individual sub-segments are stored. If the data of two sub-segments should be filled in a structure, the sub-segments should share the same variable name and the same length.

Useful steps for this are:

- Creation of the sub-segment (Add new Sub Segment)
- Filling with objects
- Duplication of the sub-segment (Duplicate)

Result of this grouping is a structure and a C-array of the type of this structure. The array has as many elements as group members (sub-segments with same variable name) are available. The advantage of such an organization is that the indexed addressing of the C-arrays can be used. E.g. with multi axis drives for fast access with pointer switching in firmware between the individual axes.

The same principle is pursued for manufacturer data. However, it is possible to select a starting index for the segment and to declare the segment size freely. That means that the data is stored normally up to the user defined index. From this index on the tree becomes segmented in the manufacturer specific area.

9.3. Predefined symbols

Predefined preproccesor symbols like __DATE__, __TIME__, __FILE__ etc. can be used within strings, but you have to put quotes around.

9.4. Limitations

The optimization configured at the optimization tab of an index can only be used outside of sub-segments. The reason for this is, that sub-segments are stored in C-structures and it is not possible to assign a storage class to a single member of a structure.

10. Complex data types

The standard /CiA_301/ allows the definition of manufacturer-specific complex data types within the index range 0040h-005Fh and device profile specific standard and complex data types within the index range 0060h-025Fh. The DT only supports the device profile specific complex data types with the object code DEFSTRUCT.

Before objects of such data types can be specified, the data type must be created in the DT. Data types are considered line-dependent, i.e. in order to specify objects of complex data types the complex data type must be available on the same line.
The following steps are necessary to create a new data type:

1. select *Line* > *Object Dictionary* > *Data Types* in the project tree
2. press *Add new Data Type* on tab *Mask*
   → the input dialog *Add new Index* is opened
3. enter the desired index into the dialog *Add new Index* in hexadecimal format without special identifier, example: 0040, and press <Enter>
   → an entry for the new data type is in the project tree, see *Line* > *Object Dictionary* > *Data Types* > <*new index*>
4. specify the new data type about button *Add new Sub-Index*, tab *Structure* and tab *Optimization*

If the data type is created, objects of the data type can be defined, when the objects have the *Object Code* RECORD.

It is recommended to use the same name for *C Name* and *EDS Name* for the data type.

During the generation process the DT generates a struct definition for the data type and stores the struct definition in objects.h. In multi-line projects identical data type definitions on various lines are summed up to one struct definition.

11. Object-specific callback functions

Object-specific callback functions allow to execute application-specific operations during reading and/or writing of objects via SDO, PDO and/or MPDO.

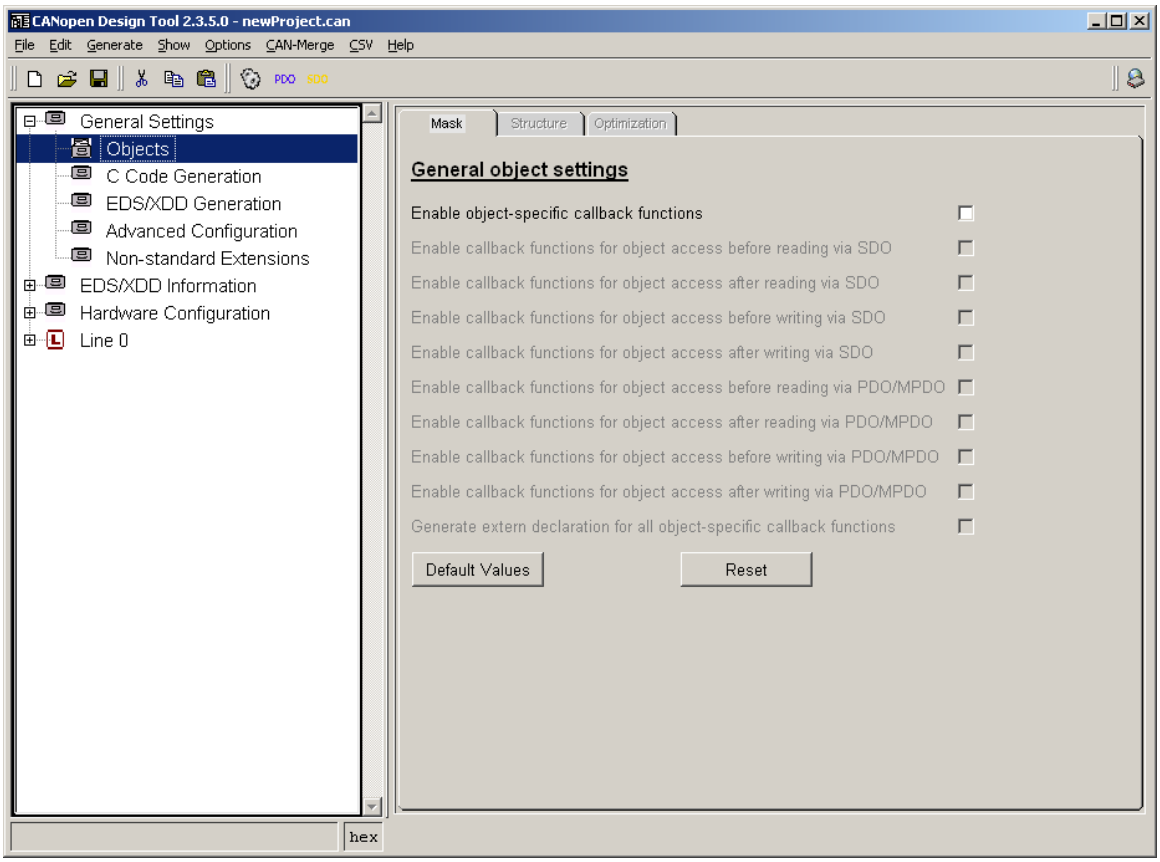


figure 10: dialog for *General Settings / Objects*

The usage of the object-specific callback functions requires the following steps:

- Use CANopen Library V4.5 and higher and configure the CANopen Library version about *General Settings / Kind of CANopen Library / Source code library V4.5x*. This makes the object-specific callback functions available.
- The object-specific callback functions can be activated or deactivated about *General Settings / Objects / Enable object-specific callback functions*.
- If object-specific callback functions are activated generally, it is possible to switch on/off the object-specific callback functions service related according to the following criteria, see figure 10:
  - access to the object via SDO, PDO and/or MPDO,
  - read and/or write access to the object and
  - before and/or after changing the value of the object in the object dictionary.
- The extern declaration for the object-specific callback functions can be located in the generated or application software. This can be configured about *General*

Settings / Objects / Generate extern declaration for all object-specific functions.

- For each object one object-specific callback function can be specified about object *Main-Index* / tab *Structure / C Callback*, see figure 11. This function is called for all accesses to the object configured in *General Settings / Objects*. The name of the object-specific callback function is application-specific. But the function prototype for the object-specific callback function is fixed as follow:

```
RET_T <callback_name> (UNSIGNED16 index, UNSIGNED8 sub,
          CO_OBJ_CB_TYPE_T reason CO_COMMA_LINE_DECL);
```

  It is allowed to specify the same object-specific callback function for different objects.

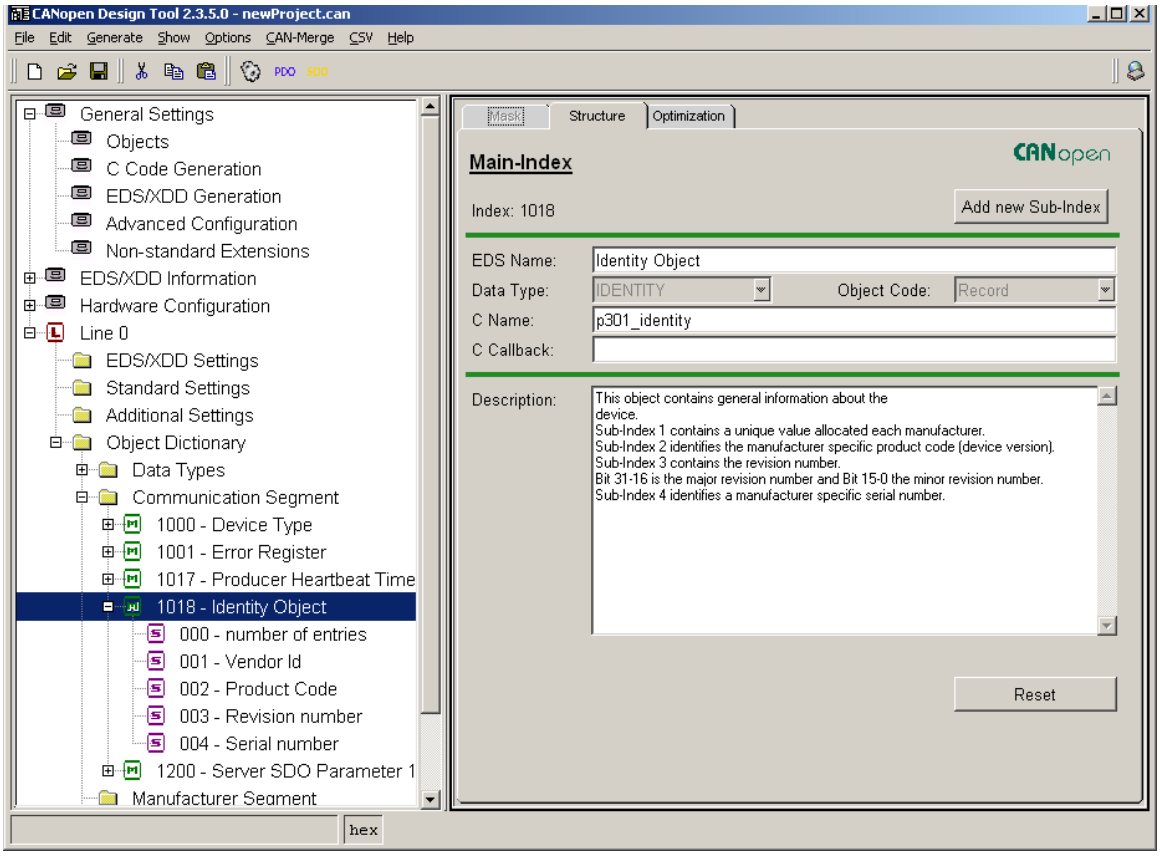


figure 11: dialog for object *Main-Index* / tab *Structure / C Callback*

Each setting in the DT is described in the online help about <F1>. The flow of calling of the object-specific callback functions is documented in the user manual of the CANopen Library.

12. Manufacturer-specific object descriptions

The DT provides format descriptors for the generation of manufacturer-specific object descriptions in ASCII-format for different document types.

The description template is a text in ASCII-Format including format descriptors for object-specific information and is stored in a file. During the generation of all files the description template is applied to the specified objects. In the result an object description for the specified objects from the specified lines is created and stored in the object description file.

Depending on the format of the object description file it can be necessary to add a file header and/or an end-of-file with special format settings to get a complete object description file. The description header is a text in ASCII-Format which is loaded at the beginning of the object description file. The description end is a text in ASCII-Format which is loaded at the end of the object description file.

The DT is able to generate up to 3 different manufacturer-specific object descriptions during generation process. The line, the object range and the files with the descriptions and the location of the object descriptions are configurable about the menu *Options* > *Generation Options* > *Manufacturer-specific object descriptions*, see figure 12.

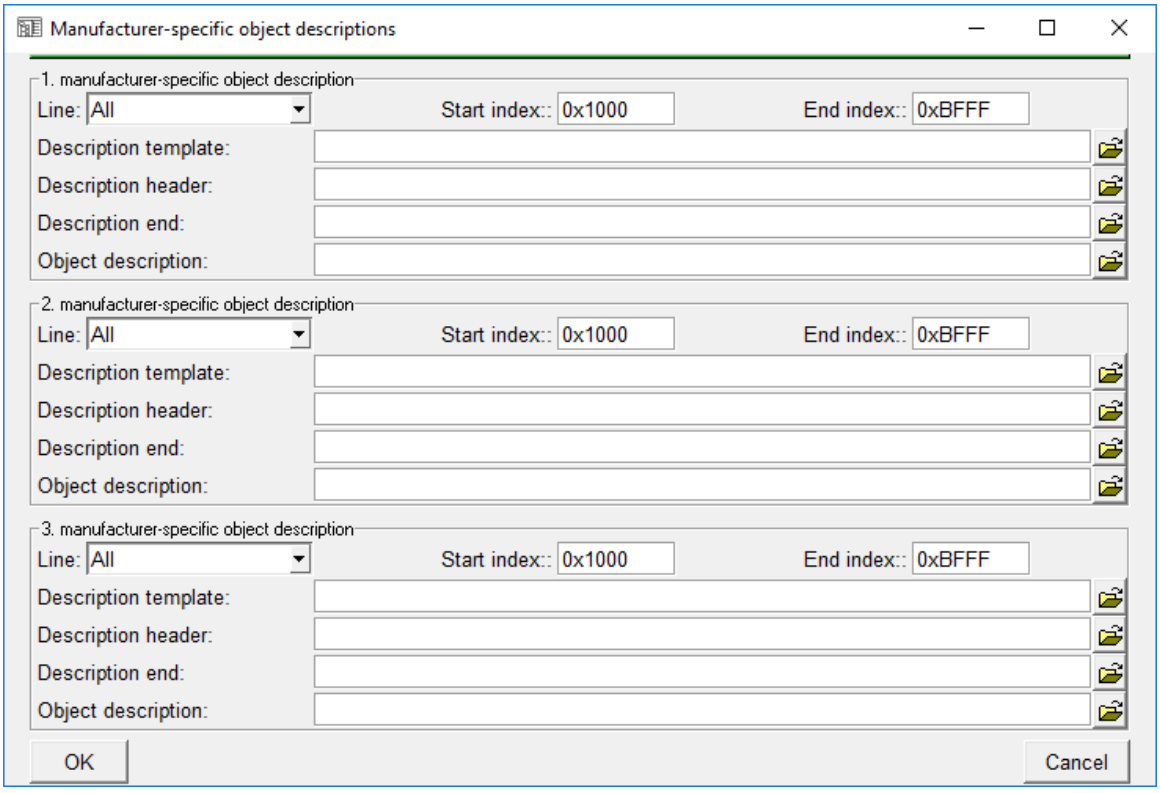


figure 12: dialog *Options* > *Generation Options* > *Manufacturer-specific object descriptions*

12.1. Format descriptors

The DT supports the following format descriptors:

format descriptor	description
%a	access type of the sub-index: RO, WO, RW, RWW, RWR, CONST
%B	object code of the main-index in hexadecimal format without prefix (example: 07)
%C	name of the C callback function
%c	nonvolatile storage: 0 - object is not stored nonvolatile, 1 - object is stored nonvolatile
%D	refuses write on download for the sub-index: 0 - not valid, 1 - valid
%d	data type of the sub-index according to CiA-309-3 (example: u32)
%e	EDS name of the main-index
%F	default value in EDS valid for the sub-index: 0 - not valid, 1 - valid
%G	C name of the data type of the main-index as string (example: DATATYPE_T)
%g	size in bytes of the sub-index in decimal format (example: 12)
%H	data type number of the main-index in hexadecimal format without prefix (example: 004A)
%h	data type number of the sub-index in hexadecimal format without prefix (example: 004A)
%I	main-index in hexadecimal format without prefix (example: 1A00)
%i	main-index in decimal format (example: 4096)
%L	line number in hexadecimal format without prefix (example: A)
%l	line number in decimal format (example: 12)
%M	limits in EDS valid for the sub-index: 0 - not valid, 1 - valid
%m	lower (minimum) limit of the sub-index • for numerical objects: in hexadecimal format with prefix "0x" (example: 0xFFFFFFFF) • for string objects: <empty string>
%N	EDS/C name of the sub-index as string, for objects of object code RECORD it is in C syntax (example: myRecordElement)
%n	name of the main-index in C code
%p	PDO mapping allowed for the sub-index: 0 - not valid, 1 - valid
%O	refuses Read on scan for the sub-index: 0 - not valid, 1 - valid

format descriptor	description
%R	valid after reset for the sub-index: 0 - not valid, 1 - valid
%S	sub-index in hexadecimal format without prefix (example: A)
%s	sub-index in decimal format (example: 12)
%T	description of the object Take note that the Design Tool can not convert special characters because it does not know the desired document type.
%U	upper limit of the sub-index • for numerical objects: in hexadecimal format with prefix "0x" (example: 0xFFFFFFFF) • for string objects: <empty string>
%u	unit of the sub-index (example: 100 ms)
%V	default value of the sub-index • for numerical objects: in hexadecimal format with prefix "0x" and with the keyword "$NODEID" for SDO-, EMCY- and PDO-cobIDs (example: 0xFFFFFFFF or $NODEID+0x200) • for string objects: without quotation tags (example: Test string)
%v	default value of the sub-index • for numerical objects: in decimal format with the keyword "$NODEID" for SDO-, EMCY- and PDO-cobIDs (example: 1614872592 or $NODEID+512) • for string objects: without quotation tags (example: Test string)
%W	default value of the sub-index • for numerical objects: in hexadecimal format with prefix "0x" without the keyword "$NODEID" for SDO-, EMCY- and PDO-cobIDs (example: 0xFFFFFFFF) • for string objects: without quotation tags (example: Test string)
%w	default value of the sub-index • for numerical objects: in decimal format without the keyword "$NODEID" for SDO-, EMCY- and PDO-cobIDs (example: 1614872592) • for string objects: without quotation tags (example: Test string)

format descriptor	description
%z	enumeration counter of the object in decimal format, counting starts with 0 (example: 12, i.e. it is the 12th object)

table 6: format descriptors

12.2. Example for html-documentation

Each specified object in the object dictionary is listed with index, sub-index in bold text style and the object description. The following example only refers to object 1000h and 1001h.

description template:

```
<p>
  <b> object %Ih/%S </b>: %T
</p>
```

description header:

```
<h1> Object dictionary </h1>
```

description end:

```
<hr>
```

object description:

```
<h1> Object dictionary </h1>
<p>
<b> object 1000h/0 </b>: The device type specifies the kind of device. The lower 16 bit contain the device profile number and the upper 16 bit an additional information.
</p>
<p> <b> object 1001h/0 </b>: The error register is a field of 8 bits, each for a certain error type. If an error occurs the bit has to be set: Bit 0 generic error, Bit 1 current, Bit 2 voltage, Bit 3 temperature, Bit 4 communication error (overrun, error state), Bit 5 device profile specific, Bit 6 reserved, Bit 7 manufacturer specific
</p>
<hr>
```

indication in the web-browser:

Object dictionary

object 1000h/0: The device type specifies the kind of device. The lower 16 bit contain the device profile number and the upper 16 bit an additional information.

object 1001h/0: The error register is a field of 8 bits, each for a certain error type. If an error occurs the bit has to be set: Bit 0 generic error, Bit 1 current, Bit 2 voltage, Bit 3 temperature, Bit 4 communication error (overrun, error state), Bit 5 device profile specific, Bit 6 reserved, Bit 7 manufacturer specific

12.3. Example for tcl-scripts

A list of all specified objects is created. The entry for each object starts with a comment including the enumeration counter. The enumeration counter is also used as list index for the list objTab. The index, sub-index and the default value are listed for each object. The following example only refers to object 1000h and 1001h.

description template:

```
#object %z
set objNum %z
set objTab($objNum,indexHex)      0x%I
set objTab($objNum,subHex)        %S
set objTab($objNum,defValHex)     "%V"
```

description header:

```
################################################
# list of objects
```

description end:

<does not exist>

object description:

```
################################################
# list of objects

#object 0
set objNum 0
set objTab($objNum,indexHex)      0x1000
set objTab($objNum,subHex)        0
set objTab($objNum,defValHex)     "0x00000000"
#object 1
set objNum 1
set objTab($objNum,indexHex)      0x1001
set objTab($objNum,subHex)        0
set objTab($objNum,defValHex)     "0x00"
```

The generated object list can be used after sourcing in a tcl-scripts - especially in the frame for testing in the Console of the CANopen Device Monitor.

12.4. Example for rtf documents

Each specified object in the object dictionary is listed with index, sub-index and the name. The following example only refers to object 1000h and 1001h.

description template:

```
\par
{\b Index:} 0x%I \par
```

{\b Sub-index:} %s \par
{\b Name:} %e \par
\par

description header:

{\rtf1\ansi\deff0

description end:

}

object description:

{\rtf1\ansi\deff0
\par
{\b Index:} 0x1000 \par
{\b Sub-index:} 0 \par
{\b Name:} Device Type \par
\par
\par
{\b Index:} 0x1001 \par
{\b Sub-index:} 0 \par
{\b Name:} Error Register \par
\par
}

after the import into a WORD document:

Object index: 0x1000
Sub-index: 0
Name: Device Type

Object index: 0x1001
Sub-index: 0
Name: Error Register

12.5. Example for csv-files

The generation of manufacturer-specific object descriptions can be used for csv-export.

csv-specifier line (see chapter 14.2.5):

```
index;sub;mObjCode;mDtIndex;sDtIndex;size;acc;min;max;val;
```

description template:

```
0x%I;%s;0x%B;0x%H;0x%h;%g;%a;%m;%U;%V;
```

description header:

<does not exist>

description end:

<does not exist>

object description:

```
0x2000;0;0x07;0x0006;0x0006;2;RO;0x0000;0xFFFF;0x1234;
0x3000;0;0x07;0x0007;0x0007;4;RW;0x00000000;0xFFFFFFFF;0x12345678;
```

13. CAN-Merge PlugIn

CAN-Merge provides functions to visualize and modify differences between different CANopen Design Tool projects. The CAN-Merge extension module is an optional Plug-In which is activated by the license file of the CANopen Design Tool. If a valid license is present, CAN-Merge can be started by the CAN-Merge menu in the menu bar of the CANopen Design Tool.

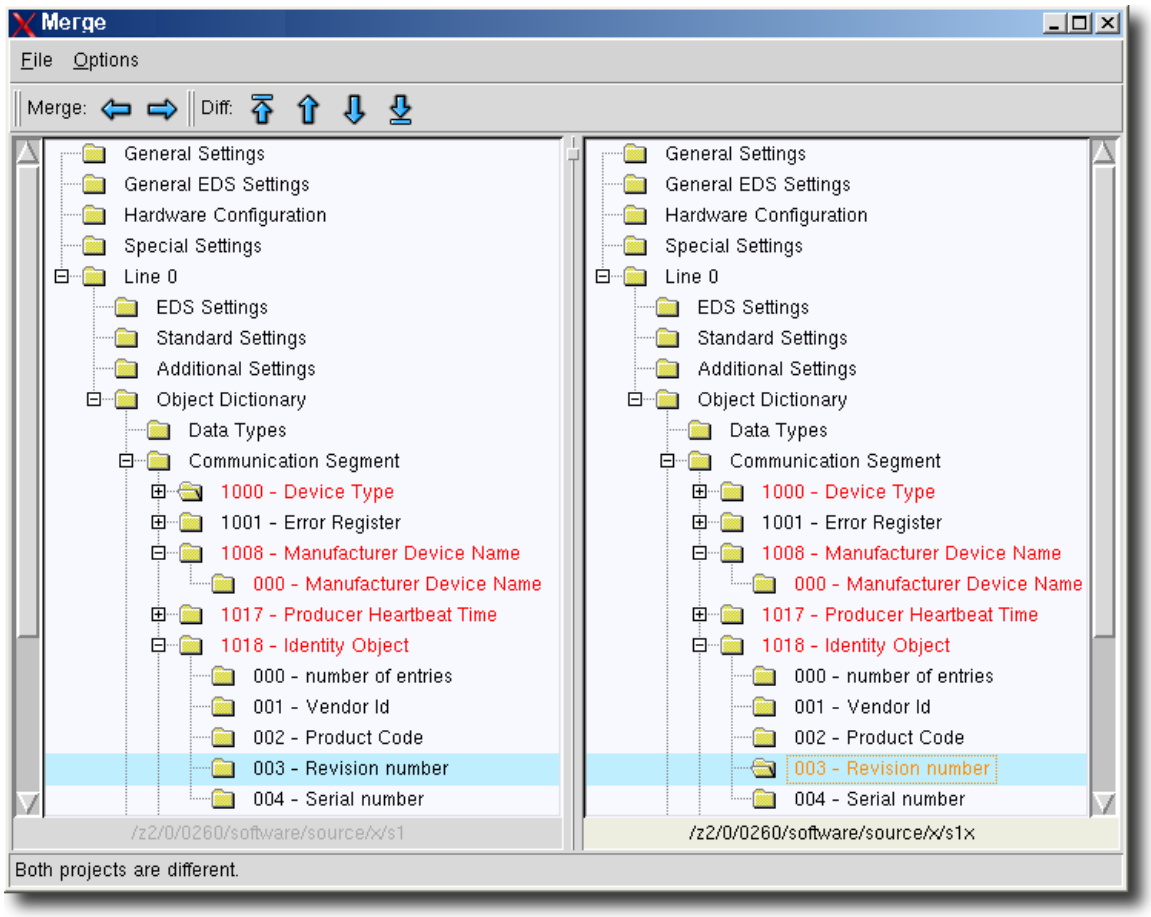


Figure 13, CAN-Merge main window displaying 2 projects

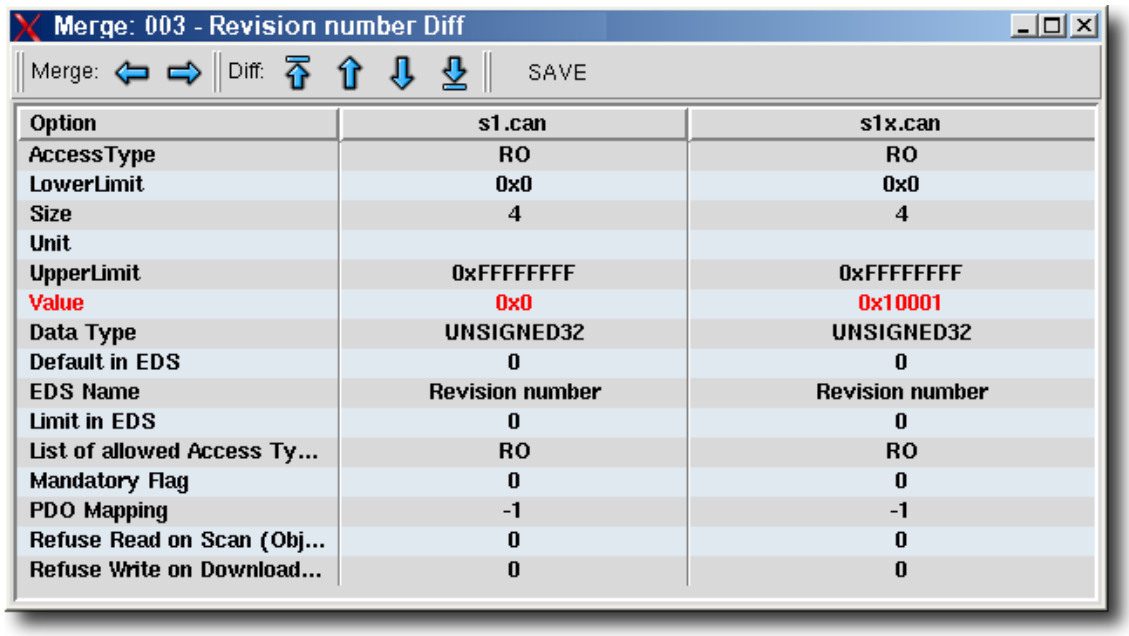
13.1. Project View

The project view shows different settings, sub segments, objects and sub indices with different colors. By using the *Merge* function in the tool bar these elements can be copied from one project into the other one.

Via the menu *File* the modified projects can be saved to disk or imported into the CANopen Design Tool. The menu entries like *Save* resp. *Open* refer to the active project tree. Via *Generate Diff Report* an overview about the differences between the projects can be generated. This diff report is an XML file containing all differences.

Via *Options* several settings can be configured. These include the attributes to be compared or merged and further options.

13.2. Detail View



Option	s1.can	s1x.can
AccessType	RO	RO
LowerLimit	0x0	0x0
Size	4	4
Unit		
UpperLimit	0xFFFFFFFF	0xFFFFFFFF
Value	0x0	0x10001
Data Type	UNSIGNED32	UNSIGNED32
Default in EDS	0	0
EDS Name	Revision number	Revision number
Limit in EDS	0	0
List of allowed Access Ty...	RO	RO
Mandatory Flag	0	0
PDO Mapping	-1	-1
Refuse Read on Scan (Obj...	0	0
Refuse Write on Download...	0	0

Figure 14, CAN-Merge diff window displaying differences of 2 sub indices

The detail view displays differences of global settings, indices and sub indices. Attributes of indices and sub indices can be modified directly within this view. Modifications have to be confirmed by the button *SAVE* in the tool bar. A plausibility check is performed when saving these settings.

14. Extension module: csv-import

The DT provides the import of object specifications in the csv-format for:

- manufacturer-specific complex data types (index range: 0040h - 005Fh)
- profile-specific complex data types (index range: 0080h - 025Fh)
- objects for communication, manufacturer-specific, device profile and dynamic variables (index range: 1000h - BFFFh)
- object-specific options for the generation of the EDS/XDD file
- object-specific options for the code generation

All data from the csv-file are imported into the specified line of the actual DT project.

The specification of an object refers to:

- main-index, i.e. these specifications are valid for all sub-indices of the object, and
- sub-index, i.e. these specifications are only valid for the specific sub-index of the object.

The differentiation between main-index and sub-index is relevant for objects of object code ARRAY or RECORD especially.

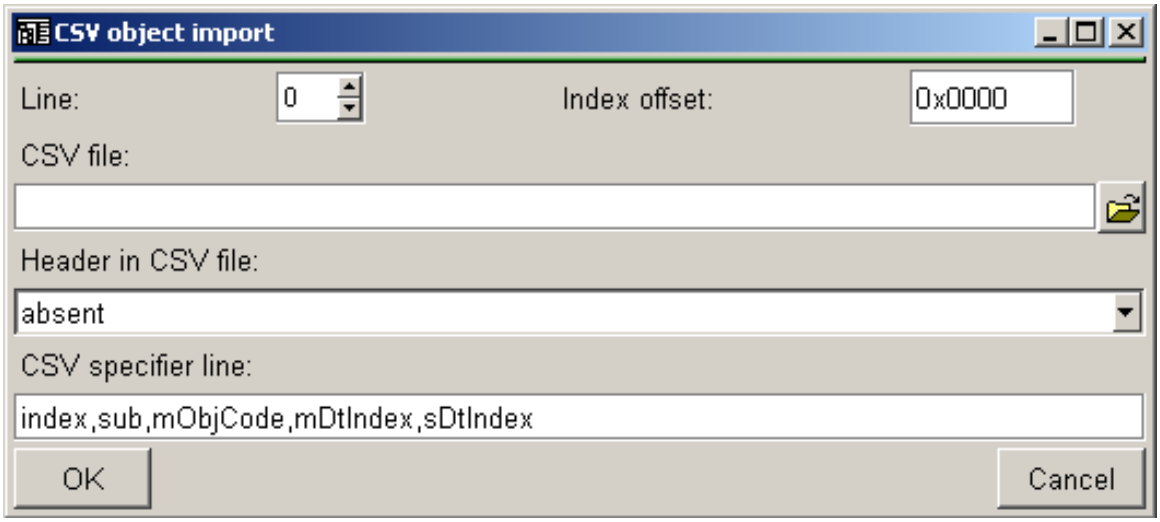


figure 15: dialog *CSV object import*

If the object already exist in the DT project it is completely substituted by the imported object specification, i.e. the object is deleted with all sub-indices and then the new object is imported. Sub-index 0 must be imported before higher sub-indices.

Errors during csv-import are reported about the GUI.

Numerical values in hexadecimal format has to be written with the prefix "0x" in C Syntax, example: 0x123A.

14.1. Structure of the csv-file

The csv-import has taken the recommendations from /RFC 4180/ in consideration and expects csv-files with the following structure:

line number in csv-file	content of the csv-data line
1	• header with a comment OR • header with the csv-specifier line OR • csv-data
2 - n	csv-data

table 8: structure of csv-file

Empty lines between the csv-data are allowed. Empty lines means lines without any char or lines containing only the csv-separators in the correct number. These lines are ignored during the csv-import. Additional comment lines are not supported because they are not recommended by /RFC 4180/.

Each csv-data line has to contain the specification of one object identified by the main-index and sub-index. This specification is a list of object properties, which can be arranged application-specific. The specification is defined by the csv-specifier line. The csv-specifier line includes special keywords for each object property separated by the csv-separator sign. All csv-data lines has to use the same csv-specifier line.

The csv-specifier line can be enter about the GUI or can be loaded from the 1st line of the csv-file.

14.2. Input parameters for the csv-import

The following input parameters can be specified in the dialog *CSV object import* about the menu *CSV > Object import*.

14.2.1. Line

description:	The csv-import works line-related, i.e. the csv-data are only imported into one line. This parameter specifies the number of the desired line.
value range:	0 - 99
condition:	The line numbering must be consecutively.
description:	The csv-import works line-related, i.e. the csv-data are only imported into one line. This parameter specifies the number of the desired line.
value range:	0

14.2.2. Index offset

description: The csv-data can be imported into different index ranges within a line. This parameter specifies an offset for shifting the main-index from the csv-data to the object in the DT project according to: obj_main_index = csv_main_index + index_offset

value range: 0x0000 - 0xBFFF

14.2.3. csv-file

description: The directory and the name of the csv-file can be specified by this input parameter.

14.2.4. Header in csv-file

description: This setting determines the interpretation of the 1st line in the csv-file.

value range:
- absent: The csv-file does not contain a header in the 1st line. csv-data start in the 1st line.
- present, used as csv-specifier line: The 1st line in the csv-file defines the csv-specifier line.
- present, ignored as csv-specifier line: The 1st line in the csv-file contains any comment. This line is ignored during csv-import.

14.2.5. csv-specifier line

description: The csv-specifier line specifies the meaning of the csv-data by special csv-specifier keywords. The csv-specifiers are described in chapter 14.3 and have to be separated by the csv-separator sign.

14.3. csv-specifiers

The csv-import supports the following csv-specifiers:

csv-specifier name	category	description
index	description:	This csv-specifier defines the numerical number of the main-index.
	category:	mandatory

csv-specifier name	category	description
	value range:	0x0040 - 0xBFFF
sub	description:	This csv-specifier defines the numerical number of the sub-index.
	category:	mandatory
	value range:	0 - 255
mObjCode	description:	This csv-specifier defines the numerical value of the object code according to /CiA-301/.
	category:	mandatory
	related to:	main-index
	value range:	0x06 for DEFSTRUCT 0x07 for VAR 0x08 for ARRAY 0x09 for RECORD
	GUI reference:	*object main-index* > *tab Structure* > *Object Code*
mDtIndex	description:	This csv-specifier defines the numerical value of the data type according to /CiA-301/.
	category:	mandatory
	related to:	main-index
	value range:	0x0001 - 0x025F
	GUI reference:	*object main-index* > *tab Structure* > *Data Type*

csv-specifier name	category	description
mCName	description:	This csv-specifier defines the name of the complex data type or of the variable in the generated C code. If no name is specified the name is generated by the Design Tool as follow: 1. The EDS name (mEdsName) is used if it is specified with adjustments to C syntax. 2. If no name is specified the Design Tool builds own names: • for data types: DATATYPE_<index>_T, example: DATATYPE_0040_T • for variables: obj_<index>, example: obj_2000
	category:	optional
	related to:	main-index
	GUI reference:	*object main-index* > *tab Structure* > *C Name*
mEdsName	description:	This csv-specifier defines the name of the object for the documentation in the EDS/XDD file. If no name is specified the Design Tool uses the C name, see csv-specifier mCName.
	category:	optional
	related to:	main-index
	GUI reference:	*object main-index* > *tab Structure* > *EDS Name*
mCb	description:	This csv-specifier defines the name of the C callback function.
	category:	optional
	related to:	main-index

csv-specifier name	category	description
	condition:	During csv-import object-specific callback functions are enabled if the csv-specifier cb is used, i.e. the checkbutton "Enable object-specific callback functions" in General Settings / Objects is set. But all other configurations in General Settings / Objects have to be done manually.
	GUI reference:	*object main-index* > *tab Structure* > *C Callback*
desc	description:	This csv-specifier defines the documentation of the object.
	category:	optional
	related to:	main-index
	GUI reference:	*object main-index* > *tab Structure* > *Description*
sCName	description:	This csv-specifier defines the name of the sub-index used in the generated C code for the element of complex data types and in the EDS/XDD files. If no name is specified the name is generated by the Design Tool as follow: 1. for object code VAR: The C name of the main-index is used, see mCName. 2. for object Code ARRAY or RECORD: The name is generated: sub_<sub>, example: sub_012. <sub> is a decimal number with 3 digits.
	category:	optional
	related to:	sub-index
	GUI reference:	*object sub-index* > *tab Structure* > *EDS Name*
sDtIndex	description:	This csv-specifier defines the numerical value of the data type of the sub-index according to /CiA-301/.

csv-specifier name	category	description
	category:	mandatory
	related to:	sub-index
	value range:	0x0001 - 0x001B
	GUI reference:	*object sub-index* > *tab Structure* > *Data Type*
size	description:	This csv-specifier defines the size of the sub-index. This csv-specifier is relevant for extended data types, e.g. VISIBLE_STRINGs.
	category:	optional
	related to:	sub-index
	default value:	• basic data types: size of the data type • VISIBLE_STRING: 125 bytes • OCTET_STRING: 125 bytes • DOMAIN: 1 byte
	GUI reference:	*object sub-index* > *tab Structure* > *Size*
unit	description:	This csv-specifier defines the unit of the object.
	category:	optional
	related to:	sub-index
	value range:	<any string>
	default value:	<empty>
	GUI reference:	*object sub-index* > *tab Structure* > *Unit*
acc	description:	This csv-specifier defines the access right of the sub-index.
	category:	optional
	related to:	sub-index
	value range:	RO, RW, WO, CONST, RWR, RWW
	default value:	RO
	GUI reference:	*object sub-index* > *tab Structure* > *Access*

csv-specifier name	category	description
max	description:	This csv-specifier defines the upper limit of numerical objects.
	category:	optional
	related to:	sub-index
	value range:	range of the numerical data type
	default value:	maximum of the data type
	GUI reference:	*object sub-index* > *tab Structure* > *Upper Limit*
min	description:	This csv-specifier defines the lower limit of numerical objects.
	category:	optional
	related to:	sub-index
	value range:	range of the numerical data type
	default value:	minimum of the data type
	GUI reference:	*object sub-index* > *tab Structure* > *Lower Limit*
val	description:	This csv-specifier defines the value of the object.
	category:	optional
	related to:	sub-index
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Default Value*
defVal	description:	This csv-specifier defines the default value. The Design Tool set the object on this value after pressing the button "Default Values".
	category:	optional
	related to:	sub-index
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *button Default Values*
mapPdo	description:	This csv-specifier defines the permission to map the object into a PDO.
	category:	optional

csv-specifier name	category	description
	related to:	sub-index
	value range:	0 - PDO mapping is not allowed 1 - PDO mapping is possible
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *PDO Mapping*
nvStorage	description:	This csv-specifier allows the usage of the mechanism for nonvolatile storage of the CANopen Library V4.5 and higher.
	category:	optional
	related to:	sub-index
	value range:	0 - nonvolatile storage not supported by Library 1 - nonvolatile storage supported by Library
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Nonvolatile Storage*
objFlagRefuseRd	description:	This csv-specifier defines the objFlag / Read (bit 1), see /CiA-306-1/ and /CiA-311/. This flag allows the reading on upload or not.
	category:	optional
	related to:	sub-index
	value range:	0 - read on upload is allowed 1 - read on upload is not allowed
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Refuse read on scan*
objFlagRefuseWr	description:	This csv-specifier defines the objFlag / Write (bit 0), see /CiA-306-1/ and /CiA-311/. This flag allows the writing on download or not.
	category:	optional

csv-specifier name	category	description
	related to:	sub-index
	value range:	0 - write on download is allowed 1 - write on download is not allowed
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Refuse write on download*
objFlagValidAfterReset	description:	This csv-specifier defines the objFlag / after reset (bit 2), see /CiA-311/. This flag determines when changed values are be activated.
	category:	optional
	related to:	sub-index
	value range:	0 - change values are valid immediately 1 - change values are valid after reset
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Valid after reset*
edsDefault	description:	This csv-specifier defines if the default values shall be documented in the EDS and XDD files.
	category:	optional
	related to:	sub-index
	value range:	0 - EDS/XDD without default value 1 - EDS/XDD with default value
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Default Value in EDS file*
edsLimit	description:	This csv-specifier defines if the limit values shall be documented in the EDS and XDD files.
	category:	optional
	related to:	sub-index
	value range:	0 - EDS/XDD without limits 1 - EDS/XDD with limits

csv-specifier name	category	description
	default value:	0
	GUI reference:	*object sub-index* > *tab Structure* > *Limits in EDS file*
optConstDefault	description:	This csv-specifier defines which memory area shall be used to store the default value of the object in C code.
	category:	optional
	related to:	main-index
	value range:	0 - default value is stored in RAM 1 - default value is stored in program memory
	default value:	1
	GUI reference:	*object main-index* > *tab Optimization* > *Constant default value*
optConstDesc	description:	This csv-specifier defines which memory area shall be used to store the description of the object in C code, see /Library manual/.
	category:	optional
	related to:	main-index
	value range:	0 - object description is stored in RAM 1 - object description is stored in program memory
	default value:	1
	condition:	The option must be local changeable, see GUI Line / Standard Settings / tab Mask / Global Settings for Object Optimization / Constant value description structure.
	GUI reference:	*object main-index* > *tab Optimization* > *Constant description structure*
optConstLimits	description:	This csv-specifier defines which memory area shall be used to store the limits of the object in C code.

csv-specifier name	category	description
	category:	optional
	related to:	main-index
	value range:	0 - object limits is stored in RAM 1 - object limits is stored in program memory
	default value:	1
	GUI reference:	*object main-index* > *tab Optimization* > *Constant limits*
optCreateExternal	description:	This csv-specifier defines if an external declaration shall be generated.
	category:	optional
	related to:	main-index
	value range:	0 - extern declaration is not generated 1 - extern declaration is generated
	default value:	1
	GUI reference:	*object main-index* > *tab Optimization* > *Create extern declaration*
optCreateTypeDef	description:	This csv-specifier defines if the C struct for an complex data type shall be generated in C code.
	category:	optional
	related to:	main-index
	value range:	for complex data types: 0 - data type struct is not generated 1 - data type struct is generated for variables: 0
	default value:	for complex data types: 1 for variables: 0
	GUI reference:	*object main-index* > *tab Optimization* > *Create type definition*
optCreateVariable	description:	This csv-specifier defines if the variable shall be generated in C code.
	category:	optional
	related to:	main-index
	value range:	for complex data types: 0

csv-specifier name	category	description
		for variables: 0 - variable is not generated 1 - variable is generated
	default value:	for complex data types: 0 for variables: 1
	condition:	The option must be local changeable, see GUI *Line* > *Standard Settings* > *tab Mask* > *Global Settings for Object Optimization* > *Create variable*.
	GUI reference:	*object main-index* > *tab Optimization* > *Create Variable*
optMemorySpecifier	description:	This csv-specifier defines a compiler specific memory specifier keyword, e.g. xdata.
	category:	optional
	related to:	main-index
	value range:	<compiler-specific string>
	default value:	<empty string>
	condition:	The setting must be local changeable, see GUI *Line* > *Standard Settings* > *tab Mask* > *Global Settings for Object Optimization* > *Storage class*.
	GUI reference:	*object main-index* > *tab Optimization* > *Storage class*
optVirtualObject	description:	This csv-specifier defines if the object shall be interpreted as virtual or normal object.
	category:	optional
	related to:	main-index
	value range:	0 - normal object 1 - virtual object
	default value:	0

csv-specifier name	category	description
	condition:	The option must be local changeable, see GUI *Line* > *Standard Settings* > *tab Mask* > *Global Settings for Object Optimization* > *Virtual object*.
	GUI reference:	*object main-index* > *tab Optimization* > *Virtual object*

table 9: csv-specifier

Mandatory csv-specifiers must be included in the csv-specifier line. Optional csv-specifier may be included in the csv-specifier line. If optional csv-specifiers are not used in the csv-specifier line the DT uses the default value for these object properties.

Settings related to the main-index are taken from sub-index 0. Changed object properties related to the main-index in sub-index 1..n are ignored.

Example: The entry for mDtIndex of the objects 3003h/1 and 3003h/2 is ignored. The data type related to the main-index is imported from object 3003h/0.

```
index;sub;mEdsName;mObjCode;mDtIndex;sCName;sDtIndex;acc;min;max;val;
0x3003;0;ARR_I16;0x08;0x0003;Highest Sub;0x0003;CONST;0x02;0x02;0x02;
0x3003;1;ARR_I16;0x08;0x0005;sub1;0x0003;RW;0x8000;0x7FFF;0x1234;
0x3003;2;ARR_I16;0x08;0x0005;sub2;0x0003;RW;0x8000;0x7FFF;0x5678;
```

The format of values is determined by the data type of the sub-index (sDtIndex). The specified values for max, min, val and/or defVal have to match this format.

```
index;sub;mCName;mObjCode;mDtIndex;sCName;sDtIndex;size;acc;min;max;val;
```

correct:

```
0x3001;0;arr_r32_2;0x08;0x0008;num;0x0008;4;CONST;2.0;2.0;2.0;
```

wrong:

```
0x3001;0;arr_r32_2;0x08;0x0008;num;0x0005;4;CONST;2.0;2.0;2.0;
```

Objects which shall be mapped into PDOs must be imported with the csv-specifier mapPdo. The generation process reports an PDO mapping error without this setting.

14.4. csv-separator

The csv-separator can be the comma, colon, semicolon or tabulator sign and is taken from the csv-specifier line according to the following rules:

1. If the lines in the csv-file are closed with comma, semicolon, colon or tabulator this sign is taken as csv-separator.
2. Otherwise the first occurrence of comma, colon, semicolon or tabulator is taken as separator.

The possible csv-separator signs are handled in the listed order.

14.5. Limitations

The extension csv-import in the DT has some limitations. The limitations result from the condition, that csv-files contains object properties and each line of the csv-file contains the same information. Therewith the following settings can not be imported by csv-files:

- general settings
- general EDS/XDD settings with except of derived value from object 1000h (p301_device_type)
- hardware configurations
- manufacturer-specific advanced configurations
- standard settings
- additional settings

The only exception is the dynamic PDO mapping. Depending on the access right of sub-index 0 of the PDO mapping parameter the CANopen Design Tool determines if static or dynamic PDO mapping is required.
For static PDO mapping the Granularity is always set to 0 bit and the option "Enable dynamic PDO Mapping" is deactivated.
For dynamic PDO mapping the Granularity is always set to 8 bit and the option "Enable dynamic PDO Mapping" is activated.

During csv-import only the format of the imported csv-data is checked. The content and dependencies are checked during generation, i.e. wrong imported csv-data can cause errors during generation. The imported csv-data are not checked against profile databases, so it is possible that settings break standards. The csv-file shall be generated carefully.

By usage of sub-segments it is necessary to change the C names of objects which are imported with an index offset in the same sub-segment.

15. Non-standard extensions

The non-standard extensions are configurable in the project tree about *General Settings / Non-standard Extensions*, see figure 16.

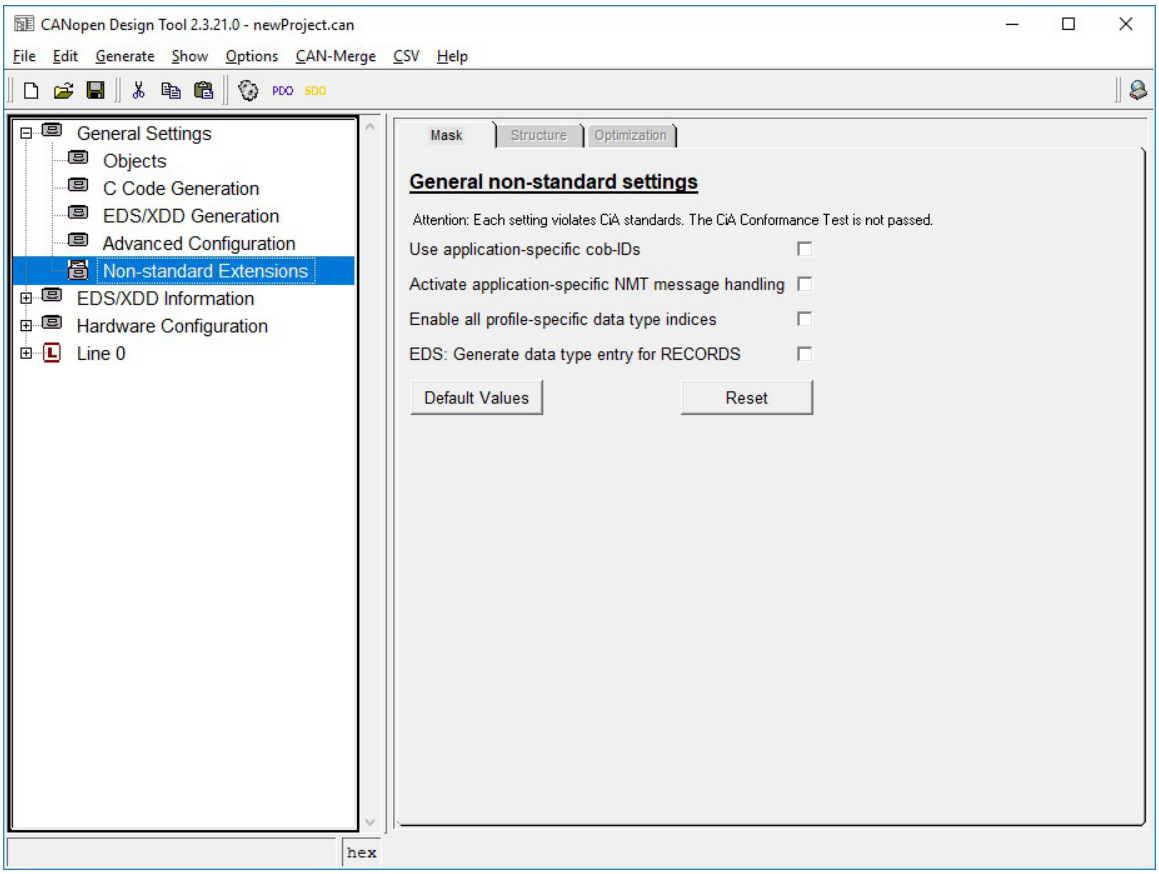


figure 16: dialog *Non-standard Extensions*

Object properties can be changed in a non-standard manner about the Expert Mode, see chapter 6.1.5.1.

15.1. Application-specific cob-IDs

The generation of the cob-IDs according to the Pre-defined Connection Set in /CiA-301/ is switched off about the project tree *General Settings / Non-standard Extensions / Use application-specific cob-IDs*.

After the activation of this setting the DT generates the documentation of the cob-IDs in the EDS and the manufacturer-specific object descriptions without the keyword "NODEID". In the C module co_init.c the DT provides the function

```
RET_T overwritePreDefConnSet (CO_GLOBVARS_PARA_DECL)
```

to set the cob-IDs on the default values configured in the DT without any modification,

i.e. the Design Tool does not change valid bits in the cob-IDs. The call of this function has to be added in the application after the execution of the function init_Library() and in nmtslave.c/resetCommInd().

The following Cob-IDs are controlled via this option:

- SYNC: object 1005h/0 (p301_cob_id_sync)
- TIME: object 1012h/0 (p301_cob_id_time)
- EMCY: object 1014h/0 (p301_cob_id_emcy)
- SDO: objects 1201h/1+2 - 12FFh/1+2 (p301_n.._ssdo_par)
- RPDO: objects 1400h/1 - 15FFh/1 (p301_n.._rpdo_para)
- TPDO: objects 1800h/1 - 19FFh/1 (p301_n.._tpdo_para)
- SRDO: objects 1301h/1+5+6 - 1340h/1+5+6 (p304_n.._srdo_para)

ATTENTION: The user is responsible to avoid cob-ID conflicts. The EDS checker of the CiA reports the violation of the Pre-defined Connection Set. The CiA Conformance Test is not passed.

15.2. Application-specific handling of NMT messages

In some less cases it is necessary to disable NMT states and to ignore NMT messages. After activation of *General Settings / Non-standard Extensions / Activate application-specific NMT message handling* in the project tree the Library passes the handling of the NMT massages on to the application.

15.3. Complete profile-specific data type index range

The activation of the setting *General Settings / Non-standard Extensions / Enable all profile-specific data type indices* enables the complete index range from 0040h to 025Fh for manufacturer-specific complex data types.

15.4. Complex data types in EDS

In some use cases the description of complex data types in the EDS file is insufficient. The index of the complex data type is documented in the EDS file after activation of *General Settings / Non-standard Extensions / EDS: Generate data type entry for RECORDS* in the project tree. The DT adds a DataType entry into the section of the main-index:

```
[1018]
SubNumber=5
ParameterName=Identity Object
ObjectType=0x09
DataType=0x0023
```

The setting *EDS: Generate data type entry for RECORDS* is applied on the main-indices 1000h - *h.

ATTENTION: The EDS checker of the CiA reports the violation of the /CiA-306-1/. The CiA Conformance Test is not passed.

During EDS-import the DT accepts such non-standard DataType entries independently from this setting. The index of the complex data type is loaded into the object properties. But the Design Tool does not generate the data type itself, i.e. the data type has to be already exist in the segment Data Types.

The DT indicates the imported complex data type in the object properties about the object *Main-Index / tab Structure / Data Type* if the data type is already defined in the DT project. If there is no suitable data type definition the indication about the object *Main-Index / tab Structure / Data Type* remains empty and the generation process fails.

It is not possible to handle the definition of complex data types itself about the EDS file because the index range of data types are not supported by EDS. Such an extension means a large violation of the /CiA-306-1/. For the import of complex data types itself the DT provides the csv-import.

Index

- -

__DATE__ 35
__FILE__ 35
__TIME__ 35

- C -

CAN line 15
CANopen device 15
CO_OBJ_CB_TYPE_T 40
csv-export 46
csv-import
  comments 52
  empty lines 52
  hardware configurations 65
  main-index properties 64
  mandatory csv-specifier 64
  optional csv-specifier 64
  PDO mapping 64
  settings 65
  sub-segmentation 65
  value format 64

- D -

data type, complex 68
data type
  complex 37
  creation 37
  DEFSTRUCT 37
  device profile-specific 37
  manufacturer-specific 37
DEFSTRUCT 37
description template 41
device
  CANopen 15
  field 15

- E -

EDS, hide options 22
EDS-import, delete Hardware Configurations 23
expert mode 22

- F -

field device 15
font settings 22

- G -

generation
  post command 21
  pre command 21

- K -

kind of
  device 24
  node 28

- L -

line 15
list.conf 41
local changeable 28

- M -

main-index 51
multi-line 15, 26

- N -

NMT messages 68
NODEID 67

- O -

object description 41
objects.c 23
optimization, sub-segmentation 36

- P -

post command 21
pre command 21
pre-defined connection set 67
profile304_july_2010.pro 18
profile443_v2_1_0.pro 18

- S -

single-line 15
sub-index 51
sub-segment
    add 27
    configure 27
    delete 27
    properties 26
    real 26
    rules 27
    virtual 26
sub-segmentation, optimization 36

- T -

type
    complex, data 37
    creation, data 37
    DEFSTRUCT, data 37
    device profile-specific, data 37
    manufacturer-specific, data 37