CANopen Library Toolset

Development Support SW – Software User Manual

CAN-N7S-CDSDP-SUM rev. 1.2

N7 SPACE SP. Z O.O.

Prepared by	Date and Signature
Konrad Grochowski	[illegible] Konrad Grochowski 2025.09.08 14:27:57 +02'00'
Verified by	
Mateusz Dyrdół	[illegible] Mateusz Dyrdół 2025.09.09 13:33:51 +02'00'
Approved by	
Seweryn Ścibior	Seweryn Ścibior Digitally signed by Seweryn Ścibior Date: 2025.09.09 15:59:36 +02'00'

Table of Contents

Change Record

Issue	Date	Change
1.0	2024-12-01	Initial release
1.1	2025-06-13	Update for TRR: • dcfnetlint description updated • New CLI tools described (dcfnetmon, dcfnetsim, dcfnetlintd) • GUI manual added
1.2	2025-09-08	Update for CDR/QR: • Reference documents updated • 10.4.1 – CLI commands documentation updated with new options • Tutorials for "typical tasks" added (11.3)

1 Introduction

This document provides Software User Manual for the CANopen Development Support SW deliverable of the CANopen Library Toolset project.

CANopen SW Library (CANSW) is an adaptation to space industry requirements of an existing and field-tested open-source CANopen library (*lely-core*). CANSW is compliant with space-specific CANopen extensions defined in ECSS-E-ST-50-15C and ECSS Criticality Category B software requirements. It was developed in the scope of previous ESA activity and validated on representative hardware platform (SAMV71). In the scope of this project its validation will be extended to include other ARM (SAMRH71 and SAMRH707) and LEON3 (GR712RC) platforms.

CANopen Library Test Environment (CTESW) defines the environment required to execute CANopen Library Test Suite (CTSSW) which is used to validate CANSW. CTSSW was developed in the scope of previous ESA activity and is available as open-source software. In the scope of this project CTESW will be extended to support new platforms and CTSSW will be executed on those.

CANopen Library Development Support Software (CDSSW) is a set of new tools developed in the scope of this project and aiming at supporting design of CANopen networks using CANSW. It will provide user with capabilities to verify semantic correctness of the multiple nodes building the CANopen network and offer support with editing, monitoring and instrumenting of the network.

The Software User Manual is produced as a standalone document and structured according to the SUM Document Requirements Definition (DRD) given in Annex H of ECSS-E-ST-40C [AD1].

2 Applicable and reference documents

2.1 Applicable documents

ID	Title	Reference	Rev.
AD1	ECSS – Space engineering Software	ECSS-E-ST-40C	6 March 2009
AD2	ECSS – CANbus extension protocol	ECSS-E-ST-50-15C	1 May 2015

2.2 Reference documents

ID	Title	Reference	Rev.
RD1	CANopen Library Toolset Development Support SW – Software Requirements Specification	CAN-N7S-CDSDP-SRS	1.3
RD2	CANopen Library Toolset Development Support SW – Software Design Document	CAN-N7S-CDSDP-SDD	1.4
RD3	CANopen Library Toolset Development Support SW – Software Configuration File	CAN-N7S-CDSDP-SCF	1.4
RD4	CAN in Automation – Electronic data sheet specification for CANopen	CiA 306	Version 1.3.0
RD5	CAN in Automation – CANopen electronic device description, Part 1: Electronic data sheet (EDS) and device configuration file (DCF)	CiA 306-1	Version 1.4.0
RD6	CAN in Automation – CANopen electronic device description, Part 3: Network variable handling and tool integration	CiA 306-3	Version 1.2.0
RD7	Language Server Protocol	https://microsoft.github.io/language-server-protocol/	

3 Terms, definitions and abbreviated terms

This document acronyms and abbreviations are listed here under.

CAN	Controller Area Network
CANDP	CANopen SW Library Data Package
CANSW	CANopen SW Library
CDSDP	CANopen Development Support Data Package
CDSSW	CANopen Development Support Software
CLI	Command Line Interface
CTESW	CANopen Test Environment Software
CTSDP	CANopen Test Suite Data Package
CTSSW	CANopen Test Suite Software
GUI	Graphical User Interface
IDE	Integrated Development Environment
N7S	N7 Space

4 Conventions

This Software User Manual describes a project that includes Command Line Interface (CLI) application, therefore it refers to various commands that can be executed in the terminal In order to make those special blocks more readable, numerous style conventions are used. This chapter summarizes said conventions.

Short commands and code fragments that are embedded inside normal text paragraphs use `this style with a monospace font`.

Commands that are a bit longer or span multiple lines follow the following style:

```
$ command
Output (optional)
```

All commands listed in this manual were prepared and validated on Ubuntu 24.04 system. Any similar Linux system should support all of the commands, it is recommended to use Ubuntu/Debian family.

Directory contents listings follow the same convention:

```
include/
└── subfolder/
    └── file
lib/
└── a generic comment about contents of lib/
share/
```

5 Purpose of the Software

CDSDP – Technical Data package for CANopen Development Support Software contains a single software item – CDSSW itself. Its aim is to provide a set of helpful tool for CANopen network engineering process. The complexity of CANopen Object Dictionary, especially in its "raw" version, as stored in CANopen standard DCF files, requires a lot of attention and manual checks by the network engineer. The DCF file itself provides only key-value pair. The interpretation of this data (roles of various entries of the Object Dictionary) is provided in the CANopen standard itself ([RD4], extended in draft [RD5]). CDSSW provides support on both levels – raw data stored in the Object Dictionary and its interpretation in the context of the whole network. It provides user with capabilities to verify semantic correctness of the multiple nodes building the CANopen network and offers support for editing of the network. CDSSW can also use network description to provide capabilities related to monitoring and instrumenting of the network.

The CDSSW includes features dedicated for ECSS specific CANopen protocol extensions, as defined in ECSS-E-ST-50-15C [AD2].

The CDSSW consists of two application sets:

- Command Line Interface (CLI) tools – standalone command line applications:
  - *dcfnetlint* – performs various syntactical and semantical checks of the CANopen network consisting of multiple nodes, defined by DCF and CPJ [RD6] files.
  - *dcfnetmon* – interprets captured CAN bus traffic using CANopen network definition based on CPJ [RD6] file, generates report with description of all frames.
  - *dcfnetsim* – simulates selected node from CANopen defined by CPJ [RD6] file, allows for sending various messages on CAN bus.
  - *dcfnetlintd* – Language Server Protocol [RD7] application for integrating CDSSW features with any Integrated Development Environments (IDE) supporting LSP.
- *dcfneteditor* – Graphical User Interface (GUI) application – a plugin to Visual Studio Code IDE – providing editing and monitoring capabilities for engineer working with such CANopen network. It embeds features of all CLI tools and integrates them with GUI.

The CLI applications are designed to easily integrate with automation tools.

The GUI application aims to be convenient and intuitive editor for human end user.

Detailed software overview can be found in SRS [RD1] and SDD [RD2].

6 External view of the software

For convenience separate archives are provided for GUI and CLI applications.

Details on the composition of the software items, versions etc. can be found in data-pack software configuration file SCF [RD3].

GUI (*dcfneteditor*) is distributed as single VSIX file – an archive in Visual Studio Code format, portable to any machine supported by IDE itself.

CLI is distributed as a single archive (ZIP and BZIP2, the latter being recommended for Linux) containing all tools binaries built for Linux operating system.

7 Operations environment

7.1 General

CDSSW is designed to be executed on common end users personal computers – x86 machines.

CDSSW CLI are a standalone applications and requires only operating system console (can be executed on “headless” system – one without any graphical user interface).

CDSSW GUI is a plugin for Visual Studio Code and requires it to be present on the target machine (must be provided by the user), with matching version.

7.2 Hardware configuration

The minimum hardware requirements are:

- Intel Core 8th generation/AMD Ryzen 1st generation (or later) with 16 GB of available system RAM.

Those requirements are driven by CDSSW GUI, the CLI application will run with less memory, but its exact usage depends on the size of the checked network and processed traffic.

7.3 Software configuration

CDSSW GUI is a plugin to an existing IDE – Visual Studio Code. It must be preinstalled by the user in compatible version (minimum 1.100). For GUI to operate with full capabilities, the *dcfnetlintd* – a Language Server Protocol application from CDSSW CLI package – must be available on machine and Visual Studio Code instance configured with its location.

7.4 Operational constraints

CDSSW does not provide any operational modes.

User should take care when working with CDSSW GUI – it offers both editing (static view of the network) and monitoring (dynamic view). Dynamic views rely on the snapshot of static data definitions – this helps to avoid data races and inconsistencies, but requires user to explicitly reload dynamic views after applying modifications in static views.

8 Operations basics

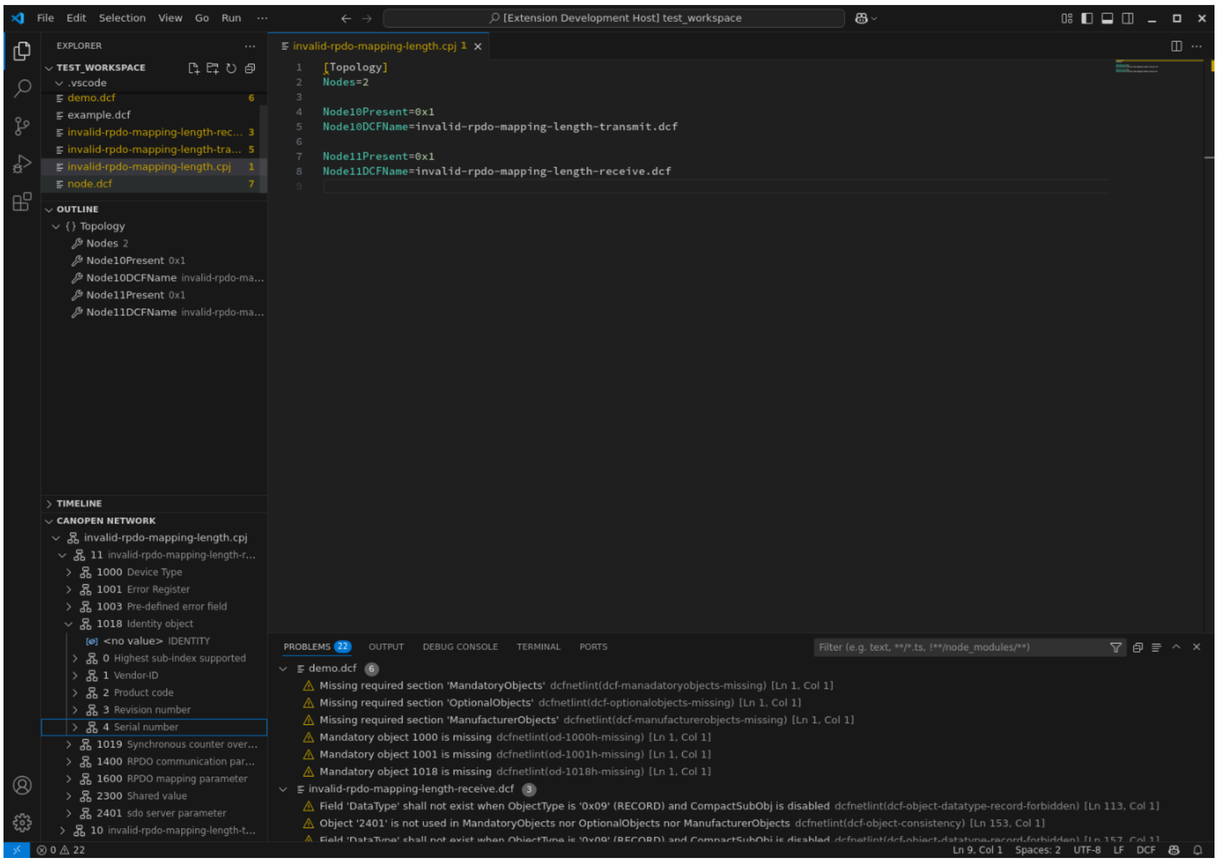
CDSSW CLI Linter (*dcfnetlint*) and Monitor (*dcfnetmon*) operates as a *single shot* command line applications in batch mode – user provides input data, application is called, analyses the data without any user interaction, and provides results as messages to standard output. User can customize the level of analysis or output format using arguments to the application call.

CDSSW CLI Simulator (*dcfnetsim*) can also work in batch mode, but it depends on *simulation plan* provided by the user. Plan can assume indefinite work or finish after performing some operations.

CSSSW CLI Daemon (*dcfnetlintd*) is a LSP server, operating as long as user requires it to work. It uses standard input and output to exchange LSP messages with client.

CDSSW GUI (*dcfneteditor*) is a extension to Visual Studio Code – a modern IDE (Integrated Development Environment) offering user multiple ways of data presentation and manipulation. Various views might be used at once to edit data.

Figure 1 presents overview of the IDE with *dcfneteditor* plugin installed.


File
Edit
Selection
View
Go
Run
[Extension Development Host] test_workspace
EXPLORER
TEST_WORKSPACE
.vscode
demo.dcf 6
example.dcf
invalid-rpdo-mapping-length-rec... 3
invalid-rpdo-mapping-length-tra... 5
invalid-rpdo-mapping-length.cpj 1
node.dcf 7
OUTLINE
{} Topology
Nodes 2
Node10Present 0x1
Node10DCFName invalid-rpdo-ma...
Node11Present 0x1
Node11DCFName invalid-rpdo-ma...
TIMELINE
CANOPEN NETWORK
invalid-rpdo-mapping-length.cpj
11 invalid-rpdo-mapping-length-r...
1000 Device Type
1001 Error Register
1003 Pre-defined error field
1018 Identity object
<no value> IDENTITY
0 Highest sub-index supported
1 Vendor-ID
2 Product code
3 Revision number
4 Serial number
1019 Synchronous counter over...
1400 RPDO communication par...
1600 RPDO mapping parameter
2300 Shared value
2401 sdo server parameter
10 invalid-rpdo-mapping-length-t...
invalid-rpdo-mapping-length.cpj 1
PROBLEMS 22
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
0 22
Ln 9, Col 1
Spaces: 2
UTF-8
LF
DCF

```
[Topology]
Nodes=2

Node10Present=0x1
Node10DCFName=invalid-rpdo-mapping-length-transmit.dcf

Node11Present=0x1
Node11DCFName=invalid-rpdo-mapping-length-receive.dcf
```

```
demo.dcf 6
  Missing required section 'MandatoryObjects' dcfnetlint(dcf-manadatoryobjects-missing) [Ln 1, Col 1]
  Missing required section 'OptionalObjects' dcfnetlint(dcf-optionalobjects-missing) [Ln 1, Col 1]
  Missing required section 'ManufacturerObjects' dcfnetlint(dcf-manufacturerobjects-missing) [Ln 1, Col 1]
  Mandatory object 1000 is missing dcfnetlint(od-1000h-missing) [Ln 1, Col 1]
  Mandatory object 1001 is missing dcfnetlint(od-1001h-missing) [Ln 1, Col 1]
  Mandatory object 1018 is missing dcfnetlint(od-1018h-missing) [Ln 1, Col 1]
invalid-rpdo-mapping-length-receive.dcf 3
  Field 'DataType' shall not exist when ObjectType is '0x09' (RECORD) and CompactSubObj is disabled dcfnetlint(dcf-object-datatype-record-forbidden) [Ln 113, Col 1]
  Object '2401' is not used in MandatoryObjects nor OptionalObjects nor ManufacturerObjects dcfnetlint(dcf-object-consistency) [Ln 153, Col 1]
  Field 'DataType' shall not exist when ObjectType is '0x09' (RECORD) and CompactSubObj is disabled dcfnetlint(dcf-object-datatype-record-forbidden) [Ln 157, Col 1]
```

Figure 1 – CDSSW GUI overview

9 Operations manual

9.1 General

CDSSW consist of two types of applications:

- CLI (Command Line Interface), referenced as CDSSW CLI and including *dcfnetlint*, *dcfnetlintd*, *dcfnetmon,* and *dcfnetsim;*
- GUI (Graphical User Interface) – *dcfneteditor*, referenced as CDSSW GUI.

Although CDSSW GUI embeds features of CDSSW CLI, those two offer different user experience (command line versus graphical), hence each following chapter describes them separately.

9.2 Set-up and initialization

9.2.1 CLI

CDSSW CLI setup requires unpacking the archive containing its binary distribution – see Listing 1. User might want to add the directory to system search path (e.g. environment variable PATH). Following examples will assume that the *dcfnetlint* application is available in search path, if not, user should prefix all calls with path to the folder holding the executable itself.

Listing 1 – Unpacking CDSSW CLI from TAR BZIP2 file (recommended for Linux).

```
$ tar -xvf unzip CAN-N7S-CDSDP-CDSSW-cli-v1_0_0.tar.bz2  # assuming version 1.0.0
```

To ensure proper setup it is recommended to try to execute the application as show on Listing 2. Output other than the one shown on the listing might indicate system incompatibility with the binary. All CLI applications support `--version` argument and should produce similar output.

Listing 2 – Checking CDSSW CLI version.

```
$ dcfnetlint --version
dcfnetlint (CANopen Network Linter) v1.0.0
Copyright (C) 2024-2025 N7 Space sp. z o.o.
```

9.2.2 GUI

CDSSW GUI – *dcfneteditor* – is a plugin to Visual Studio Code IDE. The easiest way to install it is to obtain it from Microsoft's Marketplace: go to *Extensions* and then search for *dcfneteditor* on the Marketplace – see Figure 2.

If necessary *dcfneteditor* can be installed "manually" from distributed VSIX file. Go to *Extensions* then select *Install from VSIX…* from menu (see Figure 3) and choose proper CDSSW distribution file (e.g. `CAN-N7S-CDSDP-CDSSW-gui-v1_0_0.vsix`).

CDSSW GUI requires CDSSW CLI to provide all its features (CDSSW GUI will execute without CLI, but some features will not be available). Install CLI as described in 9.2.1 and then configure *dcfneteditor* with paths to the CDSSW CLI tools using Visual Studio Code *Settings* window (see Figure 4).

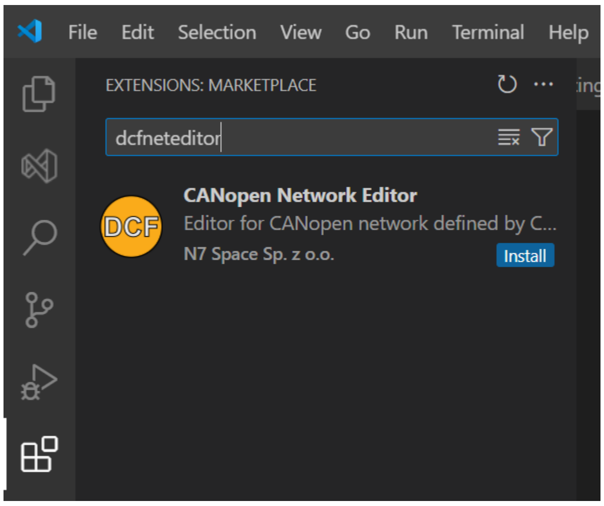


Figure 2 – *dcfneteditor* installation from Marketplace.

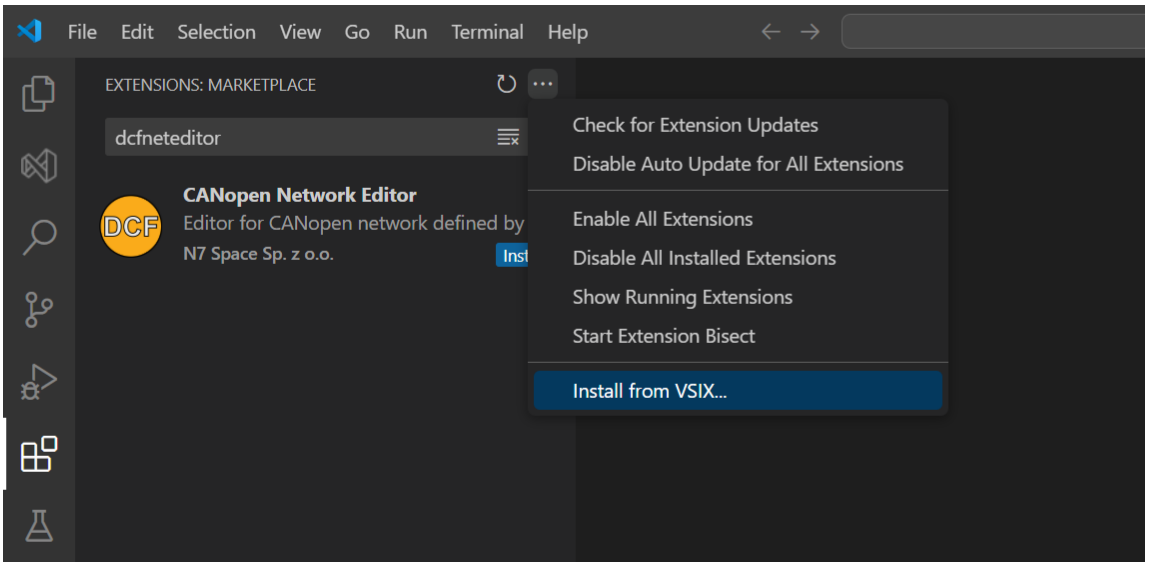
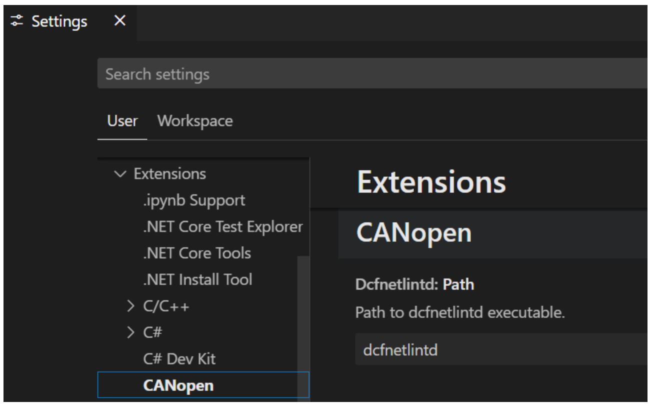


Figure 3 – *dcfnetlint* manual installation.



Dcfnetlintd: Path
Path to dcfnetlintd executable.
dcfnetlintd

Figure 4 – *dcfneteditor* settings inside Visual Studio Code.

9.3 Getting started

After performing the actions described in the previous chapter no additional operations are required and user can start using the CDSSW features. For tutorial refer to chapter 11.

9.4 Mode selection and control

N/A

9.5 Normal operations

9.5.1 CLI

9.5.1.1 dcfnetlint

Application works in batch mode – after execution it produces list of the issues found in the checked network and exits. See Listing 3.

Listing 3 – Example *dcfnetlint* call.

```
$ dcfnetlint duplicated-sections-fields-names.dcf
duplicated-sections-fields-names.dcf:6:0: warning: Field 'Description' already defined in
line 3 [dcf-sections-fields-duplicates]
```

Each issue will be produced in a separate line. When no issues are found no output is produced.

Exit code of the application depends on the number of issues found – *zero* (interpreted by operating system as success) when no issues were found, *non-zero* when any issue was found. This allows for convenient integration with automation tools – non-zero exit code will indicate error in the network and require use intervention.

9.5.1.2 dcfnetmon

Application works in batch mode – after execution it produces the traffic analysis report and exits. See Listing 4. The default format of the report is a standalone HTML that can be opened in web browser.

Listing 4 – Example *dcfnetmon* call.

```
$ dcfnetmon --network=generic-network.cpj \
            --output=report.html \
            --format=file
            --timestamp=absolute
            candump.log
```

The report can be written to requested file or produced on the standard output. This allows for *piping* the call to the monitor directly with *candump* call. Note that *candump* must exit via timeout in order to generate the report. See Listing 5

Listing 5 – Example *dcfnetmon* call with *candump*.

```
$ sudo candump -T 5000 vcan-cdssw | \
       dcfnetmon --network=generic-network.cpj --output=report.html
```

Exit code of the application depends only on the correctness of the call – *non-zero* is reported only when invalid arguments or files are passed to the application.

9.5.1.3 dcfnetsim

Application works by executing the *simulation plan* provided by the user in form of a YAML file. During execution application reports current simulation state on standard output in human readable form. See Listing 6.

Listing 6 – Example *dcfnetsim* call.

```
$ dcfnetsim --plan=sdo-upload.yml --interface=vcan-cdssw --slave-simulation
Found slave node 2 with DCF file generic-slave.dcf
Found master node 1 with DCF file generic-master.dcf
Starting simulator on CAN interface vcan-cdssw based on plan sdo-upload.yml
Slave simulator enabled!
Master node (ID: 1) created
Slave node (ID: 2) created
Resetting slave 2...
Resetting master...
SDO upload from node 2 @ 2000:00 started
Idling for 200ms...
SDO upload from node 2 @ 2000:00 completed, read value: 0x1234567890ABCDEF
Stopping the simulator!
Slave node (ID: 2) destroyed
MasterNode destroyed
```

The simulation length is user-defined – it can consist of a single message to be sent or involve configuring emulated CANopen service operating as long as needed. Simulation consists of *steps*. Each such step can be of any type supported by the simulation, and all steps are executed sequentially. Note that step execution can change the state of the simulated node (e.g. some CANopen service becomes enabled), so multiple long-running operation can be set-up simultaneously, even with linear execution of the steps. Simulation will run indefinitely unless a “stop” step is used..

Exit code of the application depends on the correctness of the call, simulation plan and simulated nodes configuration – *non-zero* is reported when invalid arguments or files are passed to the application, or simulation encounters an error due to node configuration or plan’s fault (for example, trying to write a value to non-existent object in node’s object dictionary may result in immediate stop and *non-zero* exit code).

9.5.1.4 dcfnetlintd

Application is a *daemon* – it operates as LSP server and responds to LSP client requests with processed data of the CANopen network definition. It can be used with any LSP supporting IDE to provide:

- linter (issue reporting)
- outline
- code autocompletion

Additionally, for custom clients (like the *dcfneteditor*) it can provide complete CANopen Network overview (tree view of all nodes and object dictionaries).

The application communicates using standard input and standard output. It supports only a single client, but the client can open as many files as needed. Server finishes upon request by client.

9.5.2 GUI

CDSSW GUI is a plugin for Visual Studio Code. After installation, it will automatically activate when a workspace with DCF/CPJ files is opened. When DCF or CPJ file is opened, it provides data for built-in outline view, syntax highlighting and real-time linting. Detailed overview of the available views and features can be found in 10.3.

9.6 Normal termination

9.6.1 CLI

9.6.1.1 dcfnetlint

Application always terminates after reporting the issues. The execution time depends on the complexity of the analysed network and end-user machine, but should not exceed dozens of seconds.

Important note: *non-zero* exit code is to be expected when any issues are detected. It is still considered nominal termination of the application itself, it points to the need of network inspection. Non-nominal termination will also use *non-zero* exit code but accompanied by error message instead of issue list.

9.6.1.2 dcfnetmon

Application always terminates after producing the traffic analysis report. The execution time depends on the number of frames to process, complexity of the network, and end-user machine, but for thousands of frames should not exceed dozens of seconds.

Monitor exits with *zero* exit code (success) upon generating the report. Non-nominal termination will result in *non-zero* exit code accompanied by error message provided on standard output.

9.6.1.3 dcfnetsim

The way application finishes nominally depends on the *simulation plan* provided as input to the application. If the plan contains explicit *stop simulation* step, then the application finishes after processing whole plan up to the “*stop*” step. Otherwise, the application will run and continue to execute planned simulation, until user sends SIGINT (interrupt signal – Ctrl-C on standard Linux terminal). After reception of the signal application will finish in nominal way.

The execution time of the application depends on the simulation plan provided by the user (and user decision if plan has no explicit finish).

Simulator exits with *zero* exit code (success) upon finishing the simulation. Non-nominal termination will result in *non-zero* exit code accompanied by error message provided on standard output.

9.6.1.4 dcfnetlintd

The *dcfnetlintd* is a daemon, meaning that it by design run indefinitely, serving responses to LSP client requests. To finish it has receive SIGINT (interrupt signal) from the parent application (in LSP it is usually the client that starts the server and maintains it as long as it needs it).

The application exits with *non-zero* exit code if the client did not requested closeup using dedicated LSP messages before server was closed.

9.6.2 GUI

CDSSW GUI is a plugin working inside the Visual Studio Code. The plugin terminates when the IDE closes – upon request by the user.

9.7 Error conditions

9.7.1 CLI

Applications might fail if user provides it with invalid arguments. There are two common errors:

- arguments are not matching available list of arguments (misspelled, missing required argument, mutually exclusive arguments used together etc.),
- files passed to the application cannot be accessed by it.

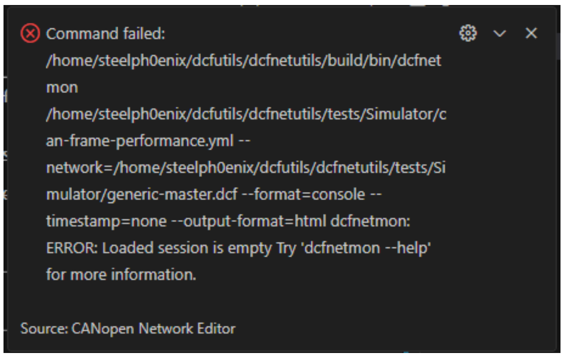
In both cases the CLI will finish with *non-zero* exit code and produce detailed message about the encountered error, prefixed with `dcfnetlint: ERROR:` (the application name in the message depends on the called tool). See example on Listing 7.

Listing 7 – Example of invalid CDSSW CLI call.

```
$ dcfnetlint a.dcf b.dcf
dcfnetlint: ERROR: Multiple files specified
Try 'dcfnetlint --help' for more information.
```

9.7.2 GUI

If invalid arguments are provided to one of CDSSW binaries via VSCode, it will display appropriate error message with output from executed binary either in message box, or console output.


Command failed: /home/steelph0enix/dcfutils/dcfnetutils/build/bin/dcfnetmon /home/steelph0enix/dcfutils/dcfnetutils/tests/Simulator/can-frame-performance.yml --network=/home/steelph0enix/dcfutils/dcfnetutils/tests/Simulator/generic-master.dcf --format=console --timestamp=none --output-format=html dcfnetmon: ERROR: Loaded session is empty Try 'dcfnetmon --help' for more information.

Source: CANopen Network Editor

Figure 5 – Message with CDSSW tool call error

9.8 Recover runs

No dedicated actions are required – after resolving the issue the CLI application needs to be executed again, while GUI might only need re-executing the requested task.

10 Reference manual

10.1 Introduction

Basic operations can be found in chapter 9. This chapter provides detailed lists of available features.

10.2 Help method

Calling the CDSSW CLI with `-h` or `--help` argument will produce help message (and finish the application execution). See example on Listing 8.

Listing 8 – Example of CDSSW CLI help call.

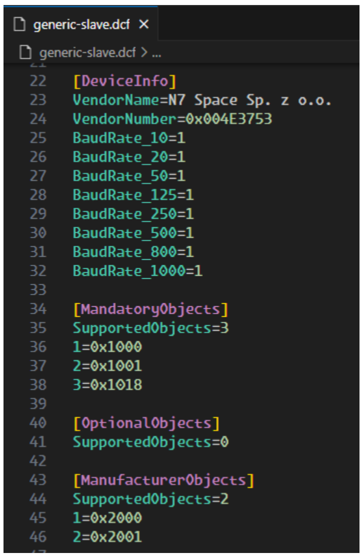
```
$ dcfnetlint --help
Usage: dcfnetlint [OPTIONS] [FILE]
Performs verification of the CANopen network configuration (CANopen Network Linter).
FILE must be a DCF file (single-node linter) or CPJ file (whole network linter).
FILE type is assumed based on .dcf or .cpj extension.

Available options:
    --help, -h       Displays this help and exits
    --version, -v    Displays version information and exits
    --checks=CHECKS  Selects the subset of checks to perform
           This argument may appear multiple times
           Checks can be provided as a list, separated by `,`
           Checks can be disabled by prepending them with `-`
           Wildcards are accepted (e.g.: `dcf-*`)
    --list-all-checks  Prints list of all supported checks and exits
    --list-enabled-checks   Prints list of enabled checks and exits
    --node-id=NODEID Sets the integer value for $NODEID
    --dump-objdict   Dumps the contents of object dictionary as DCF sections on standard
output and exits.
    --disable-non-std-time-types    Disable the support of non-standard time-related types
(SCET/SUTC)

N7 Space sp. z o.o. <https://n7space.com>
```

10.3 Screen definitions and operations

This chapter applies only to the CDSSW GUI application.

10.3.1 Syntax highlighting

Figure 6: DCF and CPJ files code will be highlighted in the editor window (*dcfnetlintd* not required).


generic-slave.dcf
generic-slave.dcf > ...

```
[DeviceInfo]
VendorName=N7 Space Sp. z o.o.
VendorNumber=0x004E3753
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1

[MandatoryObjects]
SupportedObjects=3
1=0x1000
2=0x1001
3=0x1018

[OptionalObjects]
SupportedObjects=0

[ManufacturerObjects]
SupportedObjects=2
1=0x2000
2=0x2001
```

Figure 6 – Syntax highlighting of DCF file in Visual Studio Code

10.3.2 Outline

When DCF or CPJ file is opened and *dcfnetlintd* is connected to *dcfneteditor*, the Visual Studio Code *Outline* window will be filled with tree view of all sections and entries in the file – see Figure 7.

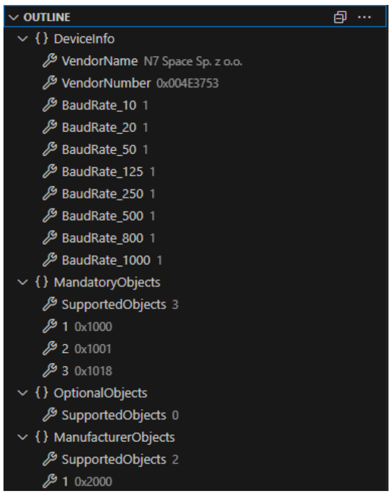

```
OUTLINE
{} DeviceInfo
    VendorName N7 Space Sp. z o.o.
    VendorNumber 0x004E3753
    BaudRate_10 1
    BaudRate_20 1
    BaudRate_50 1
    BaudRate_125 1
    BaudRate_250 1
    BaudRate_500 1
    BaudRate_800 1
    BaudRate_1000 1
{} MandatoryObjects
    SupportedObjects 3
    1 0x1000
    2 0x1001
    3 0x1018
{} OptionalObjects
    SupportedObjects 0
{} ManufacturerObjects
    SupportedObjects 2
    1 0x2000
```

Figure 7 – Outline view of DCF file in Visual Studio Code

10.3.3 Network view

If CPJ file is detected in current workspace, CANopen network view will be populated with data from DCF files of network nodes – see Figure 8. Feature requires working *dcfnetlind*.

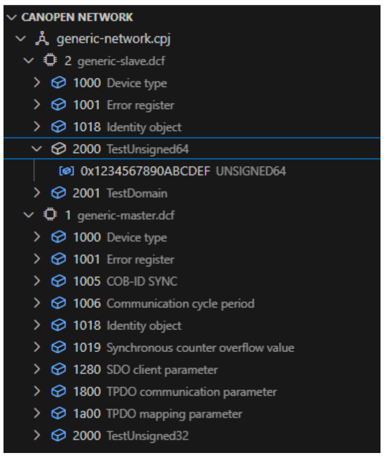


Figure 8 – Network view in Visual Studio Code

Using network view, new objects can be added to nodes in the network by right clicking on the network node – see Figure 9

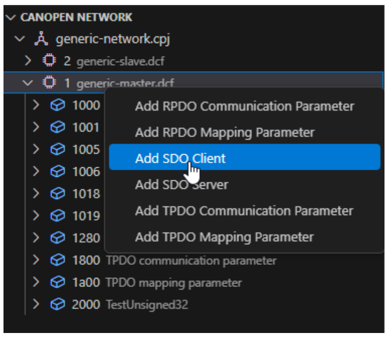
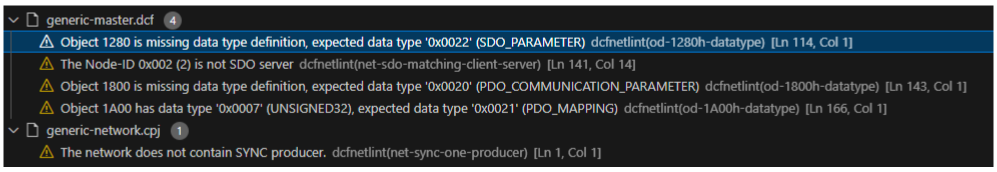

CANOPEN NETWORK
generic-network.cpj
2 generic-slave.dcf
1 generic-master.dcf
1000
1001
1005
1006
1018
1019
1280
Add RPDO Communication Parameter
Add RPDO Mapping Parameter
Add SDO Client
Add SDO Server
Add TPDO Communication Parameter
Add TPDO Mapping Parameter
1800 TPDO communication parameter
1a00 TPDO mapping parameter
2000 TestUnsigned32

Figure 9 – Adding new objects to node in network via VSCode

10.3.4 Linter

When *dcfnetlintd* is properly configured in Visual Studio Code, then any issue detected while editing single DCF file, or while working in workspace containing CPJ network definition, will be reported and shown in *Problems* window – see Figure 10.

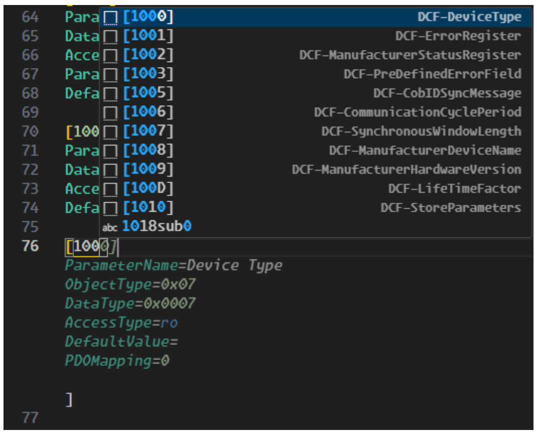


```
generic-master.dcf 4
  Object 1280 is missing data type definition, expected data type '0x0022' (SDO_PARAMETER) dcfnetlint(od-1280h-datatype) [Ln 114, Col 1]
  The Node-ID 0x002 (2) is not SDO server dcfnetlint(net-sdo-matching-client-server) [Ln 141, Col 14]
  Object 1800 is missing data type definition, expected data type '0x0020' (PDO_COMMUNICATION_PARAMETER) dcfnetlint(od-1800h-datatype) [Ln 143, Col 1]
  Object 1A00 has data type '0x0007' (UNSIGNED32), expected data type '0x0021' (PDO_MAPPING) dcfnetlint(od-1A00h-datatype) [Ln 166, Col 1]
generic-network.cpj 1
  The network does not contain SYNC producer. dcfnetlint(net-sync-one-producer) [Ln 1, Col 1]
```

Figure 10 – List of issues reported by *dcfnetlintd* in Visual Studio Code

10.3.5 Snippets

While editing the DCF file, autocompletion can be invoked to generate a pre-defined object as show on Figure 11.



```
Para [1000]                       DCF-DeviceType
Data [1001]                       DCF-ErrorRegister
Acce [1002]          DCF-ManufacturerStatusRegister
Para [1003]              DCF-PreDefinedErrorField
Defa [1005]                   DCF-CobIDSyncMessage
     [1006]           DCF-CommunicationCyclePeriod
[100 [1007]         DCF-SynchronousWindowLength
Para [1008]              DCF-ManufacturerDeviceName
Data [1009]       DCF-ManufacturerHardwareVersion
Acce [100D]                    DCF-LifeTimeFactor
Defa [1010]                    DCF-StoreParameters
     abc 1018sub0
[1000]
ParameterName=Device Type
ObjectType=0x07
DataType=0x0007
AccessType=ro
DefaultValue=
PDOMapping=0

]
```

Figure 11 – Autocompletion of DCF file in Visual Studio Code

10.3.6 Simulator integration

Plugin provides commands for running *dcfnetsim* and *dcfnetmon* inside IDE (Figure 12).

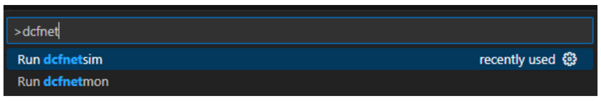
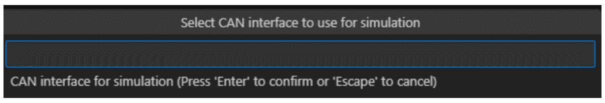


Figure 12 – Visual Studio Code commands for running monitor and simulator

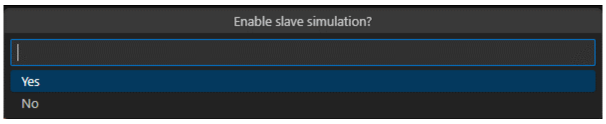
To run network simulation, trigger “Run dcfnetsim” command. If the editor is currently focused on a simulation plan file, it will try to execute it – otherwise it will ask for plan file. Additionally, it will require a CAN interface name to run simulation on (Figure 13), and simulation mode (slave simulation can be either enabled – to simulate whole network, or disabled – to simulate only master node – Figure 14.



Select CAN interface to use for simulation

CAN interface for simulation (Press 'Enter' to confirm or 'Escape' to cancel)

Figure 13 – Simulation interface selection dialog

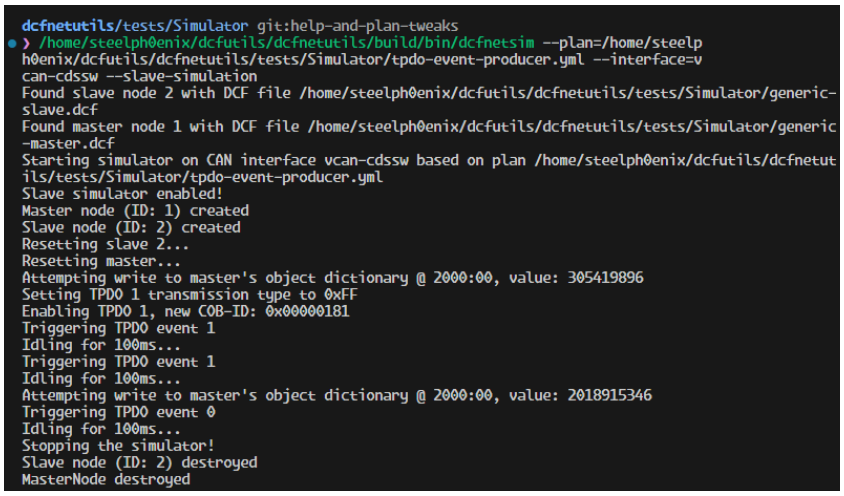


Enable slave simulation?

Yes
No

Figure 14 – Simulation mode selection dialog

After providing all required information, *dcfnetsim* will be executed with appropriate arguments in the integrated terminal (Figure 15).



```
dcfnetutils/tests/Simulator git:help-and-plan-tweaks
● ❯ /home/steelph0enix/dcfutils/dcfnetutils/build/bin/dcfnetsim --plan=/home/steelp
h0enix/dcfutils/dcfnetutils/tests/Simulator/tpdo-event-producer.yml --interface=v
can-cdssw --slave-simulation
Found slave node 2 with DCF file /home/steelph0enix/dcfutils/dcfnetutils/tests/Simulator/generic-
slave.dcf
Found master node 1 with DCF file /home/steelph0enix/dcfutils/dcfnetutils/tests/Simulator/generic
-master.dcf
Starting simulator on CAN interface vcan-cdssw based on plan /home/steelph0enix/dcfutils/dcfnetut
ils/tests/Simulator/tpdo-event-producer.yml
Slave simulator enabled!
Master node (ID: 1) created
Slave node (ID: 2) created
Resetting slave 2...
Resetting master...
Attempting write to master's object dictionary @ 2000:00, value: 305419896
Setting TPDO 1 transmission type to 0xFF
Enabling TPDO 1, new COB-ID: 0x00000181
Triggering TPDO event 1
Idling for 100ms...
Triggering TPDO event 1
Idling for 100ms...
Attempting write to master's object dictionary @ 2000:00, value: 2018915346
Triggering TPDO event 0
Idling for 100ms...
Stopping the simulator!
Slave node (ID: 2) destroyed
MasterNode destroyed
```

Figure 15 – *dcfnetsim* output in integrated Visual Studio Code terminal

10.3.7 Network monitoring

Plugin provides commands for running *dcfnetsim* and *dcfnetmon* inside IDE (Figure 12).

Network monitor can be launched in similar way, by specifying the path to traffic log in *candump* format (Figure 16), path to CPJ file describing the network (Figure 17), and format of traffic log (Figure 18 and Figure 19) – *candump* has two formats of output (to file and to console) and couple of timestamp formats.

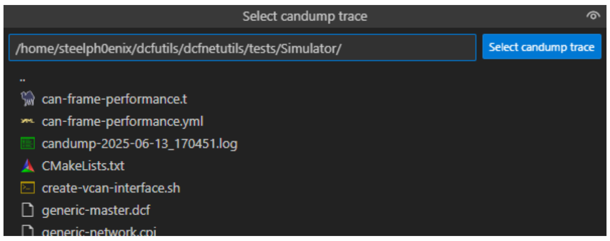


Figure 16 – *candump* trace selection for *dcfnetmon*

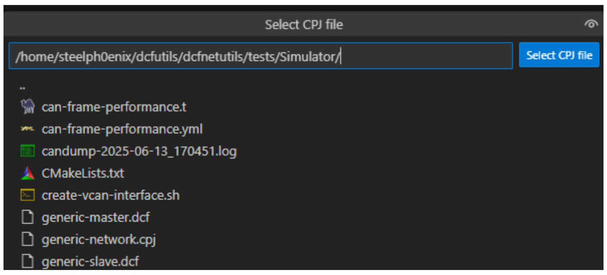


Figure 17 – CPJ file selection for *dcfnetmon*

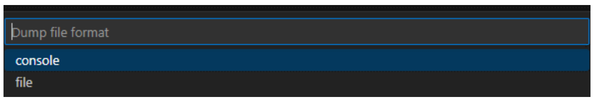


Figure 18 – *candump* log format selection for *dcfnetmon*

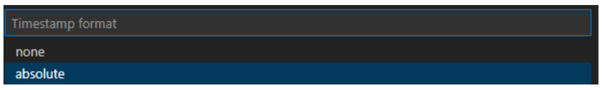
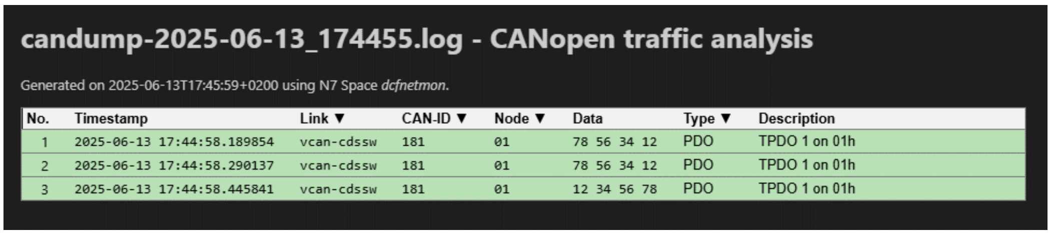


Figure 19 – *candump* timestamp format selection for *dcfnetmon*

After providing all required information, `dcfnetmon` will be used to generate HTML report of the traffic and it will be displayed in Visual Studio Code window (Figure 20).



candump-2025-06-13_174455.log - CANopen traffic analysis

Generated on 2025-06-13T17:45:59+0200 using N7 Space *dcfnetmon*.

No.	Timestamp	Link ▼	CAN-ID ▼	Node ▼	Data	Type ▼	Description
1	2025-06-13 17:44:58.189854	vcan-cdssw	181	01	78 56 34 12	PDO	TPDO 1 on 01h
2	2025-06-13 17:44:58.290137	vcan-cdssw	181	01	78 56 34 12	PDO	TPDO 1 on 01h
3	2025-06-13 17:44:58.445841	vcan-cdssw	181	01	12 34 56 78	PDO	TPDO 1 on 01h

Figure 20 – Generated report from *dcfnetmon* in Visual Studio Code

10.4 Commands and operations

10.4.1 CLI

10.4.1.1 dcfnetlint

CDSSW *dcfnetlint* CLI call consist of two groups of command line arguments: *options* and *files*.

Table 1 lists available options.

Linter determines the type of the analysis it has to perform using the files extensions:

- `.dcf` – "single node" analysis (analyses correctness of the single DCF file),
- `.cpj` – "complete network" analysis (checks each node in the network and their consistency).

Those analyses differ by the set of default checks performed (full network checks cannot be performed on a single node) and applicable options.

Table 1 – *dcfnetlint* CLI options.

Option	**Description**
`-h,--help`	Prints help message and finishes CDSSW execution.
`-v,--version`	Prints CDSSW version and finishes its execution.
`--checks=<CHECKS>`	Selects the subset of checks to be performed (all are enabled by default). Checks are identified by their name as in Table 2. Checks can be provided as a list, separated by comma. Checks can be disabled by prepending them with `-`. Wildcards are accepted (e.g.: `dcf-*`)
`--list-all-checks`	Prints list of all supported checks and exits (see Table 2).
`--list-enabled-checks`	Prints list of enabled checks (evaluates `--checks` switch) and exits.
`--node-id=<NODE-ID>`	*DCF analysis only.* Determines NODEID to be used when resolving node depended values in the DCF.

Option	Description
	In network analysis, the NODEID is obtained from network definition(CPJ file.
`--dump-objdict`	Dumps the contents of object dictionary as DCF sections on standard output and exits
`--disable-non-std-time-types`	Disable the support of non-standard ECSS time-related types (SCET/SUTC)

Table 2 – Linter checks.

Check	Description
dcf-sections-names	All DCF sections' names are consistent with DCF format. Allowed names: - FileInfo - DeviceComissioning - DeviceInfo - DummyUsage - Comments - MandatoryObjects - OptionalObjects - ManufacturerObjects - object identifier (with optional Name or Value suffix) - sub-object identifier (object identifier, "sub", index)
dcf-sections-duplicates	All DCF sections' names are unique.
dcf-sections-fields-duplicates	All DCF fields' names are unique within their section.
dcf-fileinfo-names	All DCF fields' names for FileInfo section are consistent with DCF format. Allowed names: - FileName - FileVersion - FileRevision - EDSVersion - Description - CreationTime - CreationDate - CreatedBy - ModificationTime - ModificationDate - ModifiedBy
dcf-fileinfo-fileversion-range	FileVersion DCF field value is in accepted range (UNSIGNED8).
dcf-fileinfo-filerevision-range	FileRevision DCF field value is in accepted range (UNSIGNED8).
dcf-fileinfo-edsversion-value	EDSVersion DCF field value is equal to 4.0.
dcf-fileinfo-description-length	Description DCF field value is not longer that 243 characters.
dcf-fileinfo-creationtime-value	CreationTime DCF field value is consistent with DCF format ("hh:mm(AM|PM)").
dcf-fileinfo-creationdate-value	CreationDate DCF field value is consistent with DCF format ("mm-dd-yyyy").
dcf-fileinfo-createdby-length	CreatedBy DCF field value is not longer that 245 characters.
dcf-fileinfo-modificationtime-value	ModificationTime DCF field value is consistent with DCF format ("hh:mm(AM|PM)").

Check	Description
dcf-fileinfo-modificationdate-value	ModificationDate DCF field value is consistent with DCF format ("mm-dd-yyyy").
dcf-fileinfo-modifiedby-length	ModifiedBy DCF field value is not longer that 244 characters.
dcf-deviceinfo-names	All DCF fields' names for DeviceInfo section are consistent with DCF format. Allowed names: - VendorName - VendorNumber - ProductName - ProductNumber - RevisionNumber - OrderCode - BaudRate_10 - BaudRate_20 - BaudRate_50 - BaudRate_125 - BaudRate_250 - BaudRate_500 - BaudRate_800 - BaudRate_1000 - SimpleBootUpMaster - SimpleBootUpSlave - Granularity - DynamicChannelsSupported - GroupMessaging - NrOfRxPDO - NrOfTxPDO - LSS_Supported - CompactPDO
dcf-deviceinfo-missing	DeviceInfo DCF section is provided.
dcf-deviceinfo-vendorname-missing	VendorName DCF field is provided.
dcf-deviceinfo-vendorname-length	VendorName DCF field value is not longer that 244.
dcf-deviceinfo-vendornumber-missing	VendorNumber DCF field is provided.
dcf-deviceinfo-vendornumber-range	VendorNumber DCF field value is in accepted range (UNSIGNED32).
dcf-deviceinfo-vendornumber-consistency	VendorNumber DCF field value and identity object (1018h) value at index 01h are equal.
dcf-deviceinfo-productname-length	ProductName DCF field value is not longer that 243.
dcf-deviceinfo-productnumber-consistency	ProductNumber DCF field value and identity object (1018h) value at index 02h are equal.
dcf-deviceinfo-productnumber-range	ProductNumber DCF field value is in accepted range (UNSIGNED32).
dcf-deviceinfo-revisionumber-consistency	RevisionNumber DCF field value and identity object (1018h) value at index 03h are equal.
dcf-deviceinfo-revisionumber-range	RevisionNumber DCF field value is in accepted range (UNSIGNED32).
dcf-deviceinfo-ordercode-length	OrderCode DCF field value is not longer that 245.

Check	Description
dcf-deviceinfo-baudrate10-range dcf-deviceinfo-baudrate20-range dcf-deviceinfo-baudrate50-range dcf-deviceinfo-baudrate125-range dcf-deviceinfo-baudrate250-range dcf-deviceinfo-baudrate500-range dcf-deviceinfo-baudrate800-range dcf-deviceinfo-baudrate1000-range	BaudRate_*N* DCF field value is in accepted range (BOOLEAN).
dcf-deviceinfo-simplebootupmaster-range	SimpleBootUpMaster DCF field value is in accepted range (BOOLEAN).
dcf-deviceinfo-simplebootupslave-range	SimpleBootUpSlave DCF field value is in accepted range (BOOLEAN).
dcf-deviceinfo-granularity-range	Granularity DCF field value is in accepted range [0,64]..
dcf-deviceinfo-groupmessaging-range	GroupMessaging DCF field value is in accepted range (BOOLEAN).
dcf-deviceinfo-nrofrxpdo-range	NrOfRXPDO DCF field value is in accepted range (UNSIGNED16).
dcf-deviceinfo-nroftxpdo-range	NrOfTXPDO DCF field value is in accepted range (UNSIGNED16).
dcf-deviceinfo-lsssupported-range	LSS_Supported DCF field value is in accepted range (BOOLEAN).
dcf-deviceinfo-compactpdo-range	CompactPDO DCF field value is in accepted range (UNSIGNED32).
dcf-devicecomissioning-names	All DCF fields' names for DeviceComissioning section are consistent with DCF format. Allowed names: - NodeID - NodeName - NodeRefd - Baudrate - NetNumber - NetworkName - NetRefd - CANopenManager - LSS_SerialNumber
dcf-devicecomissioning-nodeid-range	NodeID DCF field value is in accepted range (UNSIGNED8).
dcf-devicecomissioning-nodeid-missing	NodeID DCF field is provided.
dcf-devicecomissioning-nodename-length	NodeName DCF field value is not longer than 246 characters.
dcf-devicecomissioning-noderefd-length	NodeRefD DCF field value is not longer than 249 characters.
dcf-devicecomissioning-baudrate-range	Baudrate DCF field value is in accepted range (UNSIGNED16).
dcf-devicecomissioning-netnumber-range	NetNumber DCF field value is in accepted range (UNSIGNED32).
dcf-devicecomissioning-networkname-length	NetworkName DCF field value is not longer than 243 characters.
dcf-devicecomissioning-netrefd-length	NetRefD DCF field value is not longer than 249 characters.
dcf-devicecomissioning-canopenmanager-range	CANopenManager DCF field value is in accepted range (BOOLEAN).
dcf-devicecomissioning-lssserialnumber-range	LSS_SerialNumber DCF field value is in accepted range (UNSIGNED32).
dcf-deviceinfo-lssserialnumber-consistency	LSS_SerialNumber DCF field value and identity object (1018h) value at index 04h are equal.

Check	Description
dcf-dummyusage-names	All DCF fields' names for DummyUsage section are consistent with DCF format. Allowed names: - Dummy0001 - Dummy0002 - Dummy0003 - Dummy0004 - Dummy0005 - Dummy0006 - Dummy0007 - Dummy0012 - Dummy0013 - Dummy0014 - Dummy0014 - Dummy0015 - Dummy0016 - Dummy0018 - Dummy0019 - Dummy001A - Dummy001B
dcf-dummyusage-dummy-range	DummySection's DCF fields' values are in accepted range (BOOLEAN).
dcf-comments-names	All DCF fields' names for Comments section are consistent with DCF format. Allowed names: - Lines - LineN, where N is a decimal number
dcf-comments-lines-missing	Lines DCF field is provided.
dcf-comments-lines-range	Lines DCF field value is in accepted range (UNSIGNED16).
dcf-comments-line-length	LineN DCF field value is not longer than 249 characters.
dcf-comments-line-consistency	All LineN fields of Comments sections are consistent with Lines field value. The LineN fields shall use consecutive numbers for N, starting form 1 up to the value of Lines field.
dcf-manadatoryobjects-names	All DCF fields' names for MandatoryObjects section are consistent with DCF format. Allowed names: - SupportedObjects - N, where N is a decimal number
dcf-manadatoryobjects-missing	MandatoryObjects DCF section is provided.
dcf-manadatoryobjects-supportedobjects-missing	SupportedObjects DCF field is provided for MandatoryObjects section.
dcf-manadatoryobjects-supportedobjects-consistency	All fields of MandatoryObjects section are consistent with SupportedObjects field value. The fields shall use consecutive numbers for N, starting form 1 up to the value of SupportedObjects field.
dcf-manadatoryobjects-fields-range	MandatoryObjects fields' values are in accepted range (UNSIGNED16).
dcf-manadatoryobjects-fields-required	MandatoryObjects section contains at least 1000h, 1001h and 1018h objects.
dcf-manadatoryobjects-consistency	All objects listed in MandatoryObjects section are defined in the DCF.

Check	Description
dcf-optionalobjects-names	All DCF fields' names for OptionalObjects section are consistent with DCF format. Allowed names: - SupportedObjects - N, where N is a decimal number
dcf-optionalobjects-missing	OptionalObjects DCF section is provided.
dcf-optionalobjects-supportedobjects-missing	SupportedObjects DCF field is provided for OptionalObjects section.
dcf-optionalobjects-supportedobjects-consistency	All fields of OptionalObjects section are consistent with SupportedObjects field value. The fields shall use consecutive numbers for N, starting form 1 up to the value of SupportedObjects field.
dcf-optionalobjects-fields-range	OptionalObjects fields' values are in accepted range (UNSIGNED16).
dcf-optionalobjects-consistency	All objects listed in OptionalObjects section are defined in the DCF.
dcf-manufacturerobjects-names	All DCF fields' names for ManufacturerObjects section are consistent with DCF format. Allowed names: - SupportedObjects - N, where N is a decimal number
dcf-manufacturerobjects-missing	ManufacturerObjects DCF section is provided.
dcf-manufacturerobjects-supportedobjects-missing	SupportedObjects DCF field is provided for ManufacturerObjects section.
dcf-manufacturerobjects-supportedobjects-consistency	All fields of ManufacturerObjects section are consistent with SupportedObjects field value. The fields shall use consecutive numbers for N, starting form 1 up to the value of SupportedObjects field.
dcf-manufacturerobjects-fields-range	ManufacturerObjects fields' values are in accepted range (UNSIGNED16).
dcf-manufacturerobjects-consistency	All objects listed in ManufacturerObjects section are defined in the DCF.
dcf-object-consistency	All objects defined in the DCF are listed in MandatoryObjects, OptionalObjects or ManufacturerObjects sections.
dcf-object-dataobjects-forbidden	Defined objects do not use indexes for data types [0x0000, 0x0FFF].

Check	Description
dcf-object-names	All DCF fields' names for an object are consistent with DCF format. The allowed names shall be: - SubNumber - ParameterName - ObjectType - DataType - AccessType - LowLimit - HighLimit - DefaultValue - PDOMapping - ObjFlags - CompactSubObj - ParameterValue - UploadFile - DownloadFile - Denotation - ParamRefd
dcf-object-parametername-missing	ParameterName DCF field is provided for object and sub-object sections.
dcf-object-parametername-length	ParameterName DCF field value is not longer that 241.
dcf-object-objecttype-values	ObjectType field values is in the allowed set. Allowed values: - 00h for NULL - 02h for DOMAIN - 05h for DEFTYPE - 06h for DEFSTRUCT - 07h for VAR - 08h for ARRAY - 09h for RECORD

Check	Description
dcf-object-datatype-values	DataType field values is in the allowed set. The allowed values shall be: - 0x0001 for BOOLEAN - 0x0002 for INTEGER8 - 0x0003 for INTEGER16 - 0x0004 for INTEGER32 - 0x0005 for UNSIGNED8 - 0x0006 for UNSIGNED16 - 0x0007 for UNSIGNED32 - 0x0008 for REAL32 - 0x0009 for VISIBLE_STRING - 0x000A for OCTET_STRING - 0x000B for UNICODE_STRING - 0x000C for TIME_OF_DAY - 0x000D for TIME_DIFF - 0x000F for DOMAIN - 0x0010 for INTEGER24 - 0x0011 for REAL64 - 0x0012 for INTEGER40 - 0x0013 for INTEGER48 - 0x0014 for INTEGER56 - 0x0015 for INTEGER64 - 0x0016 for UNSIGNED24 - 0x0018 for UNSIGNED40 - 0x0019 for UNSIGNED48 - 0x001A for UNSIGNED56 - 0x001B for UNSIGNED64 - optionally value configured by user for TIME_SCET - optionally value configured by user for TIME_SUTC
dcf-object-datatype-var-missing	DataType DCF field is provided for object or sub-object of type VAR.
dcf-object-datatype-defstruct-missing	DataType DCF field is provided for object of type DEFSTRUCT and CompactSubObj option enabled.
dcf-object-datatype-array-missing	DataType DCF field is provided for object of type ARRAY and CompactSubObj option enabled.
dcf-object-datatype-record-missing	DataType DCF field is provided for object of type RECORD and CompactSubObj option enabled.
dcf-object-datatype-defstruct-forbidden	DataType DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.
dcf-object-datatype-array-forbidden	DataType DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-datatype-record-forbidden	DataType DCF field is not provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-datatype-domain-consistency	DataType DCF field is consistent with object type DOMAIN.

Check	Description
dcf-object-accesstype-values	AccessType field value is in the allowed set. The allowed values shall be: - ro for "read only" - wo for "write only" - rw for "read/write" - rww for "read/write on process input" - rww for "read/write on process output" - const for "constant value"
dcf-object-accesstype-var-missing	AccessType DCF field is provided for object or sub-object of type VAR.
dcf-object-accesstype-defstruct-missing	AccessType DCF field is provided for object of type DEFSTRUCT and CompactSubObj option enabled.
dcf-object-accesstype-array-missing	AccessType DCF field is provided for object of type ARRAY and CompactSubObj option enabled.
dcf-object-accesstype-record-missing	AccessType DCF field is provided for object of type RECORD and CompactSubObj option enabled.
dcf-object-accesstype-defstruct-forbidden	AccessType DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.
dcf-object-accesstype-array-forbidden	AccessType DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-accesstype-record-forbidden	AccessType DCF field is not provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-defaultvalue-boolean-range dcf-object-defaultvalue-integer8-range dcf-object-defaultvalue-integer16-range dcf-object-defaultvalue-integer24-range dcf-object-defaultvalue-integer32-range dcf-object-defaultvalue-integer40-range dcf-object-defaultvalue-integer48-range dcf-object-defaultvalue-integer56-range dcf-object-defaultvalue-integer64-range dcf-object-defaultvalue-unsigned8-range dcf-object-defaultvalue-unsigned16-range dcf-object-defaultvalue-unsigned24-range dcf-object-defaultvalue-unsigned32-range dcf-object-defaultvalue-unsigned40-range dcf-object-defaultvalue-unsigned48-range dcf-object-defaultvalue-unsigned56-range dcf-object-defaultvalue-unsigned64-range dcf-object-defaultvalue-real32-range dcf-object-defaultvalue-real64-range	DefaultValue DCF field value is in allowed range for an object with specified data type.
dcf-object-defaultvalue-defstruct-forbidden	DefaultValue DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.
dcf-object-defaultvalue-array-forbidden	DefaultValue DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-defaultvalue-record-forbidden	DefaultValue DCF field is not provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-defaultvalue-null-forbidden	DefaultValue DCF field is not provided for object of type NULL.
dcf-object-pdomapping-range	PDOMapping field value is in accepted range (BOOLEAN).
dcf-object-pdomapping-defstruct-forbidden	PDOMapping DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.

Check	Description
dcf-object-pdomapping-array-forbidden	PDOMapping DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-pdomapping-record-forbidden	PDOMapping DCF field is not provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-pdomapping-domain-forbidden	PDOMapping DCF field is not provided for object of type DOMAIN.
dcf-object-subnumber-range	SubNumber field value is in accepted range (UNSIGNED8).
dcf-object-subnumber-deftype-forbidden	SubNumber DCF field is not provided for object of type DEFTYPE.
dcf-object-subnumber-var-forbidden	SubNumber DCF field is not provided for object of type VAR.
dcf-object-subnumber-defstruct-forbidden	SubNumber DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option enabled.
dcf-object-subnumber-array-forbidden	SubNumber DCF field is not provided for object of type ARRAY and CompactSubObj option enabled.
dcf-object-subnumber-record-forbidden	SubNumber DCF field is not provided for object of type RECORD and CompactSubObj option enabled.
dcf-object-subnumber-defstruct-missing	SubNumber DCF field is provided for object of type DEFSTRUCT and CompactSubObj option disabled.
dcf-object-subnumber-array-missing	SubNumber DCF field is provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-subnumber-record-missing	SubNumber DCF field is provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-subnumber-domain-forbidden	SubNumber DCF field is not provided for object of type DOMAIN.
dcf-object-subnumber-consistency	SubNumber DCF filed value is consistent with sub-objects count.
dcf-object-lowlimit-boolean-range dcf-object-lowlimit-integer8-range dcf-object-lowlimit-integer16-range dcf-object-lowlimit-integer24-range dcf-object-lowlimit-integer32-range dcf-object-lowlimit-integer40-range dcf-object-lowlimit-integer48-range dcf-object-lowlimit-integer56-range dcf-object-lowlimit-integer64-range dcf-object-lowlimit-unsigned8-range dcf-object-lowlimit-unsigned16-range dcf-object-lowlimit-unsigned24-range dcf-object-lowlimit-unsigned32-range dcf-object-lowlimit-unsigned40-range dcf-object-lowlimit-unsigned48-range dcf-object-lowlimit-unsigned56-range dcf-object-lowlimit-unsigned64-range dcf-object-lowlimit-real32-range dcf-object-lowlimit-real64-range	LowLimit DCF field value is in allowed range for an object with specified data type.
dcf-object-lowlimit-defstruct-forbidden	LowLimit DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.
dcf-object-lowlimit-array-forbidden	LowLimit DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-lowlimit-record-forbidden	LowLimit DCF field is not provided for object of type RECORD and CompactSubObj option disabled.

Check	Description
dcf-object-lowlimit-domain-forbidden	LowLimit DCF field is not provided for object of type DOMAIN.
dcf-object-lowlimit-null-forbidden	LowLimit DCF field is not provided for object of type NULL.
dcf-object-highlimit-boolean-range dcf-object-highlimit-integer8-range dcf-object-highlimit-integer16-range dcf-object-highlimit-integer24-range dcf-object-highlimit-integer32-range dcf-object-highlimit-integer40-range dcf-object-highlimit-integer48-range dcf-object-highlimit-integer56-range dcf-object-highlimit-integer64-range dcf-object-highlimit-unsigned8-range dcf-object-highlimit-unsigned16-range dcf-object-highlimit-unsigned24-range dcf-object-highlimit-unsigned32-range dcf-object-highlimit-unsigned40-range dcf-object-highlimit-unsigned48-range dcf-object-highlimit-unsigned56-range dcf-object-highlimit-unsigned64-range dcf-object-highlimit-real32-range dcf-object-highlimit-real64-range	HighLimit DCF field value is in allowed range for an object with specified data type.
dcf-object-highlimit-defstruct-forbidden	HighLimit DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.
dcf-object-highlimit-array-forbidden	HighLimit DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-highlimit-record-forbidden	HighLimit DCF field is not provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-highlimit-domain-forbidden	HighLimit DCF field is not provided for object of type DOMAIN.
dcf-object-highlimit-null-forbidden	HighLimit DCF field is not provided for object of type NULL.
dcf-object-parametervalue-boolean-range dcf-object-parametervalue-integer8-range dcf-object-parametervalue-integer16-range dcf-object-parametervalue-integer24-range dcf-object-parametervalue-integer32-range dcf-object-parametervalue-integer40-range dcf-object-parametervalue-integer48-range dcf-object-parametervalue-integer56-range dcf-object-parametervalue-integer64-range dcf-object-parametervalue-unsigned8-range dcf-object-parametervalue-unsigned16-range dcf-object-parametervalue-unsigned24-range dcf-object-parametervalue-unsigned32-range dcf-object-parametervalue-unsigned40-range dcf-object-parametervalue-unsigned48-range dcf-object-parametervalue-unsigned56-range dcf-object-parametervalue-unsigned64-range dcf-object-parametervalue-real32-range dcf-object-parametervalue-real64-range	ParameterValue DCF field value is in allowed range for an object with specified data type.
dcf-object-parametervalue-defstruct-forbidden	ParameterValue DCF field is not provided for object of type DEFSTRUCT and CompactSubObj option disabled.

Check	Description
dcf-object-parametervalue-array-forbidden	ParameterValue DCF field is not provided for object of type ARRAY and CompactSubObj option disabled.
dcf-object-parametervalue-record-forbidden	ParameterValue DCF field is not provided for object of type RECORD and CompactSubObj option disabled.
dcf-object-parametervalue-domain-forbidden	ParameterValue DCF field is not provided for object of type DOMAIN.
dcf-object-parametervalue-null-forbidden	ParameterValue DCF field is not provided for object of type NULL.
dcf-object-lowlimit-highlimit-consistency	LowLimit DCF field value is less than or equal to HighLimit DCF field value.
dcf-object-defaultvalue-highlimit-consistency	DefaultValue DCF field value is less than or equal to HighLimit DCF field value.
dcf-object-defaultvalue-lowlimit-consistency	DefaultValue DCF field value is greater than or equal to LowLimit DCF field value.
dcf-object-parametervalue-highlimit-consistency	ParameterValue DCF field value is less than or equal to HighLimit DCF field value.
dcf-object-parametervalue-lowlimit-consistency	ParameterValue DCF field value is greater than or equal to LowLimit DCF field value.
dcf-object-compactsubobj-range	CompactSubObj field value is in accepted range (UNSIGNED8).
dcf-object-compactsubobj-var-forbidden	CompactSubObj DCF field is not provided for object of type VAR.
dcf-object-compactsubobj-deftype-forbidden	CompactSubObj DCF field is not provided for object of type DEFTYPE.
dcf-object-compactsubobj-domain-forbidden	CompactSubObj DCF field is not provided for object of type DOMAIN.
dcf-object-value-consistency	DCF section with Value suffix is defined only for object with CompactSubObj field enabled.
dcf-object-value-names	All DCF fields' names for object values section are consistent with DCF format. Allowed names: - NrOfEntries - N, where N is a decimal number
dcf-object-value-nrofentries-missing	NrOfEntries DCF field is provided for object values section.
dcf-object-value-nrofentries-range	NrOfEntries DCF field value in object value section is in accepted range (UNSIGNED8).
dcf-object-value-nrofentries-consistency	All fields of object value section are consistent with NrOfEntries field value. The fields' count shall equal to the value of the NrOfEntries field.
dcf-object-value-compactsubobj-consistency	All fields of object value section are consistent with CompactSubObj field value of the parent object.
dcf-object-name-consistency	DCF section with Name suffix is defined only for object with CompactSubObj field enabled.
dcf-object-name-names	All DCF fields' names for object name section are consistent with DCF format. Allowed names: - NrOfEntries - N, where N is a decimal number
dcf-object-name-nrofentries-missing	NrOfEntries DCF field is provided for object name section.
dcf-object-name-nrofentries-range	NrOfEntries DCF field value in object name section is in accepted range (UNSIGNED8).

Check	Description
dcf-object-name-nrofentries-consistency	All fields of object name section are consistent with NrOfEntries field value. The fields' count shall equal to the value of the NrOfEntries field.
dcf-object-name-compactsubobj-consistency	All fields of object name section are consistent with CompactSubObj field value of the parent object.
dcf-sub-parent-consistency	Sub-objects are defined only for objects present in the DCF.
dcf-sub-compactsubobj-consistency	Sub-objects are defined only for objects with disabled CompactSubObj field in the DCF.
dcf-sub-objectype-consistency	Sub-objects are defined only for DEFSTRUCT, RECORD and ARRAY object types.
dcf-sub-00h-type	Sub-object at index 0 is of type UNSIGNED8.
dcf-sub-00h-access	Sub-object at index 0 is read only.
dcf-sub-00h-consistency	Value sub-object at index 0 is consistent with sub-objects count.
dcf-sub-ffh-forbidden	Sub-objects at index FFh are not defined.
od-1000h-objecttype	Object 1000h has VAR object type.
od-1000h-datatype	Object 1000h has UNSIGNED32 data type.
od-1000h-name	Object 1000h has "Device type" name.
od-1000h-access	Object 1000h has read-only access.
od-1000h-pdomapping	Object 1000h is not mapped to PDO.
od-1000h-missing	Object 1000h is present.
od-1001h-objecttype	Object 1001h has VAR object type.
od-1001h-datatype	Object 1001h has UNSIGNED8 data type.
od-1001h-name	Object 1001h has "Error register" name.
od-1001h-access	Object 1001h has read-only access.
od-1001h-missing	Object 1001h is present.
od-1002h-objecttype	Object 1002h has VAR object type.
od-1002h-datatype	Object 1002h has UNSIGNED32 data type.
od-1002h-name	Object 1002h has "Manufacturer status register" name.
od-1002h-access	Object 1002h has read-only access.
od-1003h-objecttype	Object 1003h has ARRAY object type.
od-1003h-datatype	Object 1003h has UNSIGNED32 data type.
od-1003h-name	Object 1003h has "Pre-defined error field" name.
od-1003h-access	Object 1003h has read-only access.
od-1003h-pdomapping	Object 1003h is not mapped to PDO.
od-1003h-number-of-errors	Object 1003h has valid entry at sub-index 0
od-1003h-standard-error-field	Object 1003h has valid entry at sub-index 1
od-1005h-objecttype	Object 1005h has VAR object type.
od-1005h-datatype	Object 1005h has UNSIGNED32 data type.
od-1005h-name	Object 1005h has "COB-ID SYNC" name.
od-1005h-pdomapping	Object 1005h is not mapped to PDO.
od-1005h-defaultvalue	Object 1005h has 0x00000080 or 0x80000080 default value.
od-1006h-objecttype	Object 1006h has VAR object type.
od-1006h-datatype	Object 1006h has UNSIGNED32 data type.
od-1006h-name	Object 1006h has "Communication cycle period" name.
od-1006h-pdomapping	Object 1006h is not mapped to PDO.
od-1007h-objecttype	Object 1007h has VAR object type.
od-1007h-datatype	Object 1007h has UNSIGNED32 data type.
od-1007h-name	Object 1007h has "Synchronous window length" name.
od-1007h-pdomapping	Object 1007h is not mapped to PDO.
od-1008h-objecttype	Object 1008h has VAR object type.

Check	Description
od-1008h-datatype	Object 1008h has VISIBLE_STRING data type.
od-1008h-name	Object 1008h has "Manufacturer device name" name.
od-1008h-access	Object 1008h has constant access.
od-1008h-pdomapping	Object 1008h is not mapped to PDO.
od-1009h-objecttype	Object 1009h has VAR object type.
od-1009h-datatype	Object 1009h has VISIBLE_STRING data type.
od-1009h-name	Object 1009h has "Manufacturer hardware version" name.
od-1009h-access	Object 1009h has constant access.
od-1009h-pdomapping	Object 1009h is not mapped to PDO.
od-100Ah-objecttype	Object 100Ah has VAR object type.
od-100Ah-datatype	Object 100Ah has VISIBLE_STRING data type.
od-100Ah-name	Object 100Ah has "Manufacturer software version" name.
od-100Ah-access	Object 100Ah has constant access.
od-100Ah-pdomapping	Object 100Ah is not mapped to PDO.
od-100Ch-objecttype	Object 100Ch has VAR object type.
od-100Ch-datatype	Object 100Ch has UNSIGNED16 data type.
od-100Ch-name	Object 100Ch has "Guard time" name.
od-100Ch-pdomapping	Object 100Ch is not mapped to PDO.
od-100Dh-objecttype	Object 100Dh has VAR object type.
od-100Dh-datatype	Object 100Dh has UNSIGNED8 data type.
od-100Dh-name	Object 100Dh has "Life time factor" name.
od-100Dh-pdomapping	Object 100Dh is not mapped to PDO.
od-1012h-ecss-compatibility	Object 1012h is not supported by CANSW in ECSS compatibility mode.
od-1012h-objecttype	Object 1012h has VAR object type.
od-1012h-datatype	Object 1012h has UNSIGNED32 data type.
od-1012h-name	Object 1012h has "COB-ID time stamp message" name.
od-1012h-pdomapping	Object 1012h is not mapped to PDO.
od-1013h-ecss-compatibility	Object 1013h is not supported by CANSW in ECSS compatibility mode.
od-1013h-objecttype	Object 1013h has VAR object type.
od-1013h-datatype	Object 1013h has UNSIGNED32 data type.
od-1013h-name	Object 1013h has "high resolution time stamp" name.
od-1014h-objecttype	Object 1014h has VAR object type.
od-1014h-datatype	Object 1014h has UNSIGNED32 data type.
od-1014h-name	Object 1014h has "COB-ID emergency message" name.
od-1014h-pdomapping	Object 1014h is not mapped to PDO.
od-1014h-defaultvalue	Object 1014h default value is consistent with the required format. The check shall ensure that CAN-ID in the value is consistent with "frame" bit value and that bit 30 is always zero.
od-1014h-parametervalue	Object 1014h parameter value is consistent with the required format. The check shall ensure that CAN-ID in the value is consistent with "frame" bit value and that bit 30 is always zero. The check shall apply only if parameter value field is defined in DCF.
od-1015h-objecttype	Object 1015h has VAR object type.
od-1015h-datatype	Object 1015h has UNSIGNED16 data type.
od-1015h-name	Object 1015h has "inhibit time EMCY" name.
od-1015h-pdomapping	Object 1015h is not mapped to PDO.
od-1016h-objecttype	Object 1016h has ARRAY object type.

Check	Description
od-1016h-datatype	Object 1016h has UNSIGNED32 data type.
od-1016h-name	Object 1016h has "Consumer heartbeat time" name.
od-1016h-pdomapping	Object 1016h is not mapped to PDO.
od-1017h-objecttype	Object 1017h has VAR object type.
od-1017h-datatype	Object 1017h has UNSIGNED16 data type.
od-1017h-name	Object 1017h has "Producer heartbeat time" name.
od-1017h-pdomapping	Object 1017h is not mapped to PDO.
od-1018h-objecttype	Object 1018h has RECORD object type.
od-1018h-datatype	Object 1018h has Identity data type.
od-1018h-name	Object 1018h has "Identity object" name.
od-1018h-access	Object 1018h has read-only access.
od-1018h-pdomapping	Object 1018h is not mapped to PDO.
od-1018h-missing	Object 1018h is present.
od-1018h-00h-range	Sub-object 1018h 00h has value in an allowed range [1,4].
od-1018h-01h-datatype	Sub-object 1018h 01h has UNSIGNED32 data type.
od-1018h-01h-name	Sub-object 1018h 01h has "Vendor-ID" name.
od-1018h-02h-datatype	Sub-object 1018h 02h has UNSIGNED32 data type.
od-1018h-02h-name	Sub-object 1018h 02h has "Product code" name.
od-1018h-03h-datatype	Sub-object 1018h 03h has UNSIGNED32 data type.
od-1018h-03h-name	Sub-object 1018h 03h has "Revision number" name.
od-1018h-04h-datatype	Sub-object 1018h 04h has UNSIGNED32 data type.
od-1018h-04h-name	Sub-object 1018h 04h has "Serial number" name.
od-1019h-objecttype	Object 1019h has VAR object type.
od-1019h-datatype	Object 1019h has UNSIGNED8 data type.
od-1019h-name	Object 1019h has "Synchronous counter overflow value" name.
od-1019h-pdomapping	Object 1019h is not mapped to PDO.
od-1019h-defaultvalue	Object 1019h has default value in allowed range [2,240].
od-1019h-parametervalue	Object 1019h has parameter value in allowed range [2,240].
od-1020h-objecttype	Object 1020h has ARRAY object type.
od-1020h-datatype	Object 1020h has UNSIGNED32 data type.
od-1020h-name	Object 1020h has "Verify configuration" name.
od-1020h-pdomapping	Object 1020h is not mapped to PDO.
od-1020h-highest-sub-index-supported	Object 1020h has valid entry at sub-index 0
od-1020h-configuration-date	Object 1020h has valid entry at sub-index 1
od-1020h-configuration-time	Object 1020h has valid entry at sub-index 2
od-1028h-objecttype	Object 1028h has ARRAY object type.
od-1028h-datatype	Object 1028h has UNSIGNED32 data type.
od-1028h-name	Object 1028h has "Emergency consumer" name.
od-1028h-pdomapping	Object 1028h is not mapped to PDO.
od-1028h-highest-sub-index-supported	Object 1028h has valid entry at sub-index 0
od-1028h-emergency-consumer-1	Object 1028h has valid entry at sub-index 1
od-1028h-emergency-consumer-x	Object 1028h has valid entries at sub-indices [2, 7Fh]
od-1029h-objecttype	Object 1029h has ARRAY object type.
od-1029h-datatype	Object 1029h has UNSIGNED8 data type.
od-1029h-name	Object 1029h has "Error behaviour" name.
od-1029h-pdomapping	Object 1029h is not mapped to PDO.
od-1029h-highest-sub-index-supported	Object 1029h has valid entry at sub-index 0
od-1029h-communication-error	Object 1029h has valid entry at sub-index 1
od-1029h-profile-manufacturer-specific-error	Object 1029h has valid entries at sub-indices [2, FEh]
od-1200h-objecttype	Objects 1200h-127Fh have RECORD object type.

Check	Description
od-1200h-datatype	Objects 1200h-127Fh have SDO parameter record data type.
od-1200h-name	Objects 1200h-127Fh have "SDO server parameter" name.
od-1200h-00h-missing	Sub-object 00h in objects 1200h-127Fh is present.
od-1200h-00h-range	Sub-object 00h in objects 1200h-127Fh has value in an allowed range. The allowed range shall be [2] forObject 1200h and [2,3] for objects 1201h-127Fh.
od-1200h-00h-name	Sub-object 00h in objects 1200h-127Fh has "Highest sub-index supported" name.
od-1200h-01h-missing	Sub-object 01h in objects 1200h-127Fh is present.
od-1200h-01h-defaultvalue	Sub-object 01h inObject 1200h has default value of 600h + Node-ID.
od-1200h-01h-name	Sub-object 01h in objects 1200h-127Fh has "COB-ID client -> server (rx)" name.
od-1200h-02h-missing	Sub-object 02h in objects 1200h-127Fh is present.
od-1200h-02h-defaultvalue	Sub-object 02h inObject 1200h has default value of 580h + Node-ID.
od-1200h-02h-name	Sub-object 02h in objects 1200h-127Fh has "COB-ID server -> client (tx)" name.
od-1200h-03h-range	Sub-object 03h in objects 1200h-127Fh has value in an allowed range [1, 7Fh], if present.
od-1200h-03h-name	Sub-object 03h in objects 1200h-127Fh has "Node-ID of the SDO client" name, if present.
od-1280h-objecttype	Objects 1280h-12FFh have RECORD object type.
od-1280h-datatype	Objects 1280h-12FFh have SDO Parameter data type.
od-1280h-name	Objects 1280h-12FFh have "SDO client parameter" name.
od-1280h-00h-missing	Sub-object 00h in objects 1280h-12FFh is present.
od-1280h-00h-range	Sub-object 00h in objects 1280h-12FFh has value in an allowed range [3].
od-1280h-00h-defaultvalue	Sub-object 00h in objects 1280h-12FFh has default value od 3.
od-1280h-00h-name	Sub-object 00h in objects 1280h-12FFh has "Highest sub-index supported" name.
od-1280h-01h-missing	Sub-object 01h in objects 1280h-12FFh is present.
od-1280h-01h-name	Sub-object 01h in objects 1280h-12FFh has "COB-ID client -> server (tx)" name.
od-1280h-02h-missing	Sub-object 02h in objects 1280h-12FFh is present.
od-1280h-02h-name	Sub-object 02h in objects 1280h-12FFh has "COB-ID server -> client (rx)" name.
od-1280h-03h-missing	Sub-object 03h in objects 1280h-12FFh is present.
od-1280h-03h-range	Sub-object 03h in objects 1280h-12FFh has value in an allowed range [1, 7Fh].
od-1280h-03h-name	Sub-object 03h in objects 1280h-12FFh has "Node-ID of the SDO server" name.
od-1400h-objecttype	Objects 1400h-15FFh have RECORD object type.
od-1400h-datatype	Objects 1400h-15FFh have PDO communication parameter record data type.
od-1400h-name	Objects 1400h-15FFh have "RPDO communication parameter" name.
od-1400h-00h-missing	Sub-object 00h in objects 1400h-15FFh is present.
od-1400h-00h-range	Sub-object 00h in objects 1400h-15FFh has value in an allowed range [2,6].

Check	Description
od-1400h-01h-missing	Sub-object 01h in objects 1400h-15FFh is present.
od-1400h-02h-missing	Sub-object 02h in objects 1400h-15FFh is present.
od-1600h-objecttype	Objects 1600h-17FFh have RECORD object type.
od-1600h-datatype	Objects 1600h-17FFh have PDO mapping parameter record data type.
od-1600h-name	Objects 1600h-17FFh have "RPDO mapping parameter" name.
od-1600h-00h-missing	Sub-object 00h in objects 1600h-17FFh is present.
od-1600h-01h-missing	Sub-object 01h in objects 1600h-17FFh is present.
od-1600h-application-object-x-missing	Objects 1600h-17FFh have valid entries at sub-indices [2, 40h]
od-1800h-objecttype	Objects 1800h-19FFh have RECORD object type.
od-1800h-datatype	Objects 1800h-19FFh have PDO communication parameter record data type.
od-1800h-name	Objects 1800h-19FFh have "TPDO communication parameter" name.
od-1800h-00h-missing	Sub-object 00h in objects 1800h-19FFh is present.
od-1800h-00h-range	Sub-object 00h in objects 1800h-19FFh has value in an allowed range [2,6].
od-1800h-01h-missing	Sub-object 01h in objects 1800h-19FFh is present.
od-1800h-02h-missing	Sub-object 02h in objects 1800h-19FFh is present.
od-1A00h-objecttype	Objects 1A00h-1BFFh have RECORD object type.
od-1A00h-datatype	Objects 1A00h-1BFFh have PDO mapping parameter record data type.
od-1A00h-name	Objects 1A00h-1BFFh have "TPDO mapping" name.
od-1A00h-00h-missing	Sub-object 00h in objects 1A00h-1BFFh is present.
od-1A00h-01h-missing	Sub-object 01h in objects 1A00h-1BFFh is present.
od-1A00h-application-object-x-missing	Objects 1A00h-1BFFh have valid entries at sub-indices [2, 40h]
cpj-topology-section	CPJ file contains exactly one Topology section.
cpj-topology-nodes-mandatory	CPJ file contains exactly one Nodes key under the Topology section.
cpj-topology-nodepresent-mandatory	CPJ file contains valid number of NodeXPresent keys under the Topology section.
cpj-topology-nodepresent-nodeid	X in NodeXPresent represents a valid and unique device Node-ID (1-127).
cpj-topology-nodedcfname-missing	CPJ file contains a NodeXDCFName key for every node under the Topology section.
cpj-topology-nodedcfname-nodeid	X in NodeXDCFName represents a valid and unique device Node-ID (1-127).
cpj-topology-existing-dcfs	Each NodeXDCFName key refers to an existing .dcf file
node-1005h-missing	Object 1005h is present if using synchronous PDO.
node-1006h-missing	Object 1006h is present if required. This check shall be applied to SYNC producers (based on bit 30 in the value of 1005h).
node-csdo-self-reference	Defined SDO clients with static configuration are not referring to SDO servers defined on the same node.
node-rpdo-comm-mapping-consistency	1600h-17FFh objects correspond to 1400h-15FFh objects
node-rpdo-mapped-objects-consistency	1600h-17FFh objects provide mapping definitions corresponding to defined objects
node-tpdo-comm-mapping-consistency	1A00h-1BFFh objects correspond to 1800h-19FFh objects
node-tpdo-mapped-objects-consistency	1A00h-1BFFh objects provide mapping definitions corresponding to defined objects

Check	Description
net-nodeid-uniqueness	There are no Node-ID duplications.
net-sync-one-producer	There is exactly one SYNC producer.
net-sync-counter-overflow-consistency	Check that the definition of the 1019h object is consistent across producer and consumers
net-sync-counter-tpdo-consistency	The 1019h object's value on the SYNC producer shall be the least common multiple of all the TPDO transmission types used. CheckSub-object 02h of TPDP communication parameter, if the value is in range 01h f0h, then it is synchronous cyclic and shall be collected, LCM of all collected values shall be value of 1019h in sync producer, also verify if theSub-object 06 (sync start value) is lower than value in sync producer.
net-heartbeat-matching-nodes	Consumer heartbeat times should refer to Node-ID that are heartbeat producers.
net-heartbeat-consumer-producer-time	Consumer heartbeat times are greater than corresponding producer heartbeat times.
net-emcy-matching-ids	The Emergency consumer object refers to CAN-IDs of EMCY producers.
net-sdo-matching-client-server	For every SDO client with static configuration defined, there is be a corresponding SDO server defined.
net-pdo-consistency	Every defined RPDO matches a defined TPDO
net-pdo-mapping-consistency	Every RPDO mapping matches a corresponding TPDO mapping.
net-ecss-redundancy-consistency	Every node has a matching ECSS redundancy configuration

10.4.1.2 dcfnetmon

CDSSW *dcfnetmon* CLI call consist of two groups of command line arguments: *options* and *file*.

Table 3 lists available options.

Monitor can process standard input (when *file* argument is omitted) and output to standard output (when *--output* option is omitted), but it can also work from file to file.

Providing the network definition (as CPJ file) via *--network* option is required.

Table 3 – *dcfnetmon* CLI options.

Option	Description
`-h,--help`	Prints help message and finishes CDSSW execution.
`-v,--version`	Prints CDSSW version and finishes its execution.
`--network=<CPJFILE>`	Use CPJFILE for interpretation of CAN bus traffic (required).
`--output=<OUTPUT>`	Use OUTPUT to save traffic report
`--format=[auto|console|file]`	Sets the format of file to be parsed ('auto' by default, which tries to detect the format based on first frame). Output from *candump* differs from its log format.
`--output-format=[html|text]`	Selects report format ('html' by default).
`--timestamp=[auto|none|absolute]`	Sets the format of timestamp for parsed data ('auto' by default, which tries to detect the format based on first frame). See *candump* timestamp options.
`--redundant-bus=<CANBUS>`	Name of the CAN bus from log file that should be treated as redundant. Optional.

10.4.1.3 dcfnetsim

CDSSW *dcfnetsim* CLI call consist of only of *options*, with two of them being required – network interface and simulation plan.

Table 4 lists available options.

Simulation plan consists of simulation steps, executed in sequential manner. Listing 9 provides generic layout of the simulation plan and Table 5 lists possible steps.

Table 4 – *dcfnetsim* CLI options.

Option	Description
`-h,--help`	Prints help message and finishes CDSSW execution.
`-v,--version`	Prints CDSSW version and finishes its execution.
`--plan=<PLAN>`	Use PLAN YAML file as simulation plan and config. All paths specified inside simulation plan are relative to the plan's directory
`--interface=<INTERFACE>`	CAN interface to use for master node traffic.
`--slave-simulation`	Enables slave node simulation. By default, simulator runs in asingle (master) node simulation mode, as configured in simulation plan - this allows it to be plugged to existing CAN networks without interfering with other nodes. Using this argument switches the simulator into network simulation mode, as it will simulate both the master and all remaining (slave) nodes in the network. Slave nodes will process CANopen events as configured in their DCF files, e.g. will respond to SDO transactions.

Listing 9 – Simulation plan layout.

```
config: # plan's configuration
  cpj-file: network.cpj # path to network file, relative to plan's path
  master-node-id: 1 # node-id of simulated node
steps: # list of steps
  - type: reset # every step has a type
  - type: send-can-frame
    # most steps require additional parameters
    can-id: 0x100
    data: [0x11, 0x22, 0x33]
  - type: schedule-can-frames
    period: 200
    delay: 500
    frames:
      - can-id: 0x42
        data: [0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF]
  - type: wait
    duration: 1000 # durations are typically measured in milliseconds
  - type: stop
```

Table 5 – *dcfnetsim* simulation plan steps.

Step	**Description**
`reset`	Triggers NMT reset of master node. If `--slave-simulation` argument is used, also triggers NMT reset of slave nodes.
`wait`	Runs the simulation for specified amount of time, processing CANopen events of master node. **Parameters:** • `duration` – time (in milliseconds) to run simulation for.
`stop`	Stops the simulation by triggering master node deconfiguration. All steps after this one are ignored.
`write-objdict`	Writes a value to master node’s object dictionary. **Parameters:** • `index` – object’s index • `subindex` – object's subindex • `value-type` – type of written value, one of `u8/i8/u16/i16/u32/i32/u64/i64` • `value` – value to be written

Step	Description
`send-can-frame`	Triggers transmission of custom or pre-defined CAN frame. **Parameters:** • `frame-type` (*optional)* – if not provided or set to `custom`, sends CAN frame defined by custom frame parameters. Otherwise, sends pre-defined frame specified by parameters dependent of its kind. One of `custom/sync/nmt` **Custom frame parameters:** • `can-id` – ID of the frame • `data` – Data to transmit, up to 8 bytes. • `flags` *(optional)* - CAN frame flags, in numeric format. **Pre-defined frame parameters:** • `sync` – transmits SYNC frame ○ `can-id` – ID of the SYNC frame ○ `counter` *(optional)* – SYNC frame counter • `nmt-command` – transmits NMT command ○ `node-id` – Node-ID of command target ○ `command` – Command to perform, one of `start/stop/enter-preop/reset-node/reset-comm` • `nmt-heartbeat` – transmits NMT heartbeat ○ `node-id` – Node-ID of simulated node ○ `state` – NMT state, one of `boot-up/stopped/operational/pre-operational` • `emcy` – transmits EMCY frame ○ `cob-id` – ID of the frame ○ `error-code` – EMCY error code, 16-bit value ○ `error-register` – EMCY error register, 8-bit value ○ `manufacturer-error-code` – optional, an array with up to 6 bytes.

Step	Description
`schedule-can-frames`	Triggers transmission of custom or pre-defined CAN frames based on a schedule. **Parameters:** • `period` – period between frame transmissions, in milliseconds • `delay` *(optional)* - delay of first frame, in milliseconds. • `duration` *(optional)* - how long the schedule should last, in milliseconds • `limit` *(optional)* - maximum number of frames transmitted by this schedule • `repeat-mode` *(optional)* - Specifies how the schedule should behave after transmitting the last frame o `no` – schedule ends after sending last frame o `last` – last frame on the list is transmitted repeatedly o `loop` – schedule starts from the first frame again • `frames` – list of frames to send, defined as in `send-can-frame` step.
`enable-sync`	Enables SYNC production on master node. **Parameters:** • `can-id` – CAN ID of SYNC frames • `period` – SYNC period, in milliseconds • `overflow` *(optional)* - overflow value of SYNC counter, SYNC is generated without counter if this parameter is missing.
`disable-sync`	Disables SYNC production on master node.
`enable-tpdo`	Enables specified TPDO. **Parameters:** • `pdo` – PDO to activate
`disable-tpdo`	Disables specified TPDO. **Parameters:** • `pdo` – PDO to deactivate

Step	Description
`set-tpdo-tx-type`	Configures transmission type for specified TPDO. **Parameters:** • `pdo` – PDO to configure • `tx-type` – transmission type, accepts one of following values o `acyclic` – acyclic TPDO transmission o `cyclic` – transmission on specified SYNC event, requires additional parameters: ▪ `cycle` – SYNC cycle that triggers TPDO transmission o `sync-rtr` – RTR-driven transmission on SYNC o `event` – Event-driven transmission, requires additional parameters: `event` – event triggering the transmission, one of `rtr/manufacturer/device`
`trigger-tpdo`	Triggers a TPDO by generating an internal event. **Parameters:** • `pdo` – PDO to trigger, 0 to trigger all.
`schedule-tpdo-event`	Schedules TPDO triggering based on a schedule. **Parameters:** • `period` – period between triggers, in milliseconds • `delay` *(optional)* - delay of first trigger, in milliseconds. • `duration` *(optional)* - how long the schedule should last, in milliseconds • `limit` *(optional)* - maximum number of triggers generated by this schedule

Step	Description
`sdo-sync/async-download`	Starts SDO download to a remote server as a client node. `sdo-async-download` does not complete immediately, it runs in background until completion, timeout, or simulation stops. `sdo-sync-download` blocks the simulation until transaction is completed, or timeout hits. Allows downloading data from files. Synchronous download will usually be performed as block download, while asynchronous as segmented/expedited. **Parameters:** • `node-id` – Node-ID of the SDO server • `index` – object's index • `subindex` – object's subindex • `value-type` – type of downloaded value, one of `u8/i8/u16/i16/u32/i32/u64/i64/file` • `value` – value to be written, or path to file with data if `value-type` is `file`. Path is relative to plan file directory. • `timeout` – maximum time for SDO to complete, in milliseconds
`sdo-sync/async-upload`	Starts SDO upload from a remote server as a client node. `sdo-async-upload` does not complete immediately, runs in background until completion, timeout, or simulation stops. `sdo-sync-upload` blocks the simulation until transaction is completed, or timeout hits. Allows uploading data to files. Synchronous upload will usually be performed as block upload, while asynchronous as segmented/expedited. **Parameters:** • `node-id` – Node-ID of the SDO server • `index` – object's index • `subindex` – object's subindex • `value-type` – type of downloaded value, one of `u8/i8/u16/i16/u32/i32/u64/i64/file` • `value` – path to file with data if `value-type` is `file`, unused otherwise. Path is relative to plan file directory. • `timeout` – maximum time for SDO to complete, in milliseconds

10.4.1.4 dcfnetlintd

CDSSW *dcfnetlintd* CLI call consist of a single optional *option*. Nominal execution of the tool (running it as a daemon) is when no options are provided.

Table 6 lists available options.

Table 6 – *dcfnetlintd* CLI options.

Option	**Description**
`-h,--help`	Prints help message and finishes CDSSW execution.
`-v,--version`	Prints CDSSW version and finishes its execution.

10.4.2 GUI

For Visual Studio Code manuals and tutorials refer to: https://code.visualstudio.com/docs

For available plugin features refer to 10.3.

10.5 Error messages

10.5.1 CLI

When executed incorrectly the CDSSW produces message prefixed with `dcfnetlint: ERROR:` (name of the application depends on exact called CDSSW CLI tool). See 9.7.1 for more details.

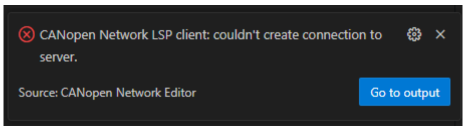
In *dcfnetlint* detected issues result in two kinds of messages:

- `error` – issue prevented CDSSW CLI to perform further analysis (e.g. parsing error),
- `warning` – issue resulted from performing one of the available checks.

List of all available linter checks can be found in Table 2. Each linter check corresponds to a specific requirement, with justification and notes – refer to SRS Annex A [RD1] for details.

10.5.2 GUI

When there's a configuration issue with Visual Studio Code extension (e.g. a path to one of the tools is invalid), it shows a message box in bottom-right corner.



CANopen Network LSP client: couldn't create connection to server.

Source: CANopen Network Editor

Figure 21 – Error message from Visual Studio Code

11 Tutorial

11.1 Introduction

This tutorial serves as an extension to the chapter 9. User might start with this chapter, but for complete overview of the application reading both is recommended. When necessary explicit references are provided. Chapter 10 provides detailed reference on all available options, it is recommended to read it after some initial experience with the software, or to search for specific function.

11.2 Getting started

11.2.1 CLI

After performing actions from chapter 9.2.1 CDSSW CLI tools should be ready to use. As mentioned there – user might choose to add them to the system search path (to be available in every console) or explicitly pass complete path to the executable when performing any operations. For the sake of the tutorial, it's assumed that all the binaries are available via system search path.

11.2.2 GUI

After performing actions from chapter 9.2.2 CDSSW GUI should be ready to use. For complete feature set following the CDSSW CLI setup (9.2.1) is also necessary.

For users not familiar with the Visual Studio Code, see tutorial offered by the IDE:

- https://code.visualstudio.com/docs/getstarted/getting-started

11.3 Using the software on a typical task

11.3.1 CLI

11.3.1.1 Analysing network node (DCF)

Checking DCF files can be performed using `dcfnetlint`. DCF file from

Listing 10 will be used for this tutorial, referenced as as *`master.dcf`*.

Listing 10 – Content of `master.dcf` file from tutorial.

```
[DeviceInfo]
VendorName=N7 Space Sp. z o.o.
VendorNumber=0x004E3753
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1

[MandatoryObjects]
```

```
SupportedObjects=3
1=0x1000
2=0x1001
3=0x1018

[OptionalObjects]
SupportedObjects=6
1=0x1005
2=0x1006
3=0x1019
4=0x1280
5=0x1800
6=0x1A00

[ManufacturerObjects]
SupportedObjects=1
1=0x2000

[1000]
ParameterName=Device type
DataType=0x0007
AccessType=ro

[1001]
ParameterName=Error register
DataType=0x0005
AccessType=ro

[1005]
ParameterName=COB-ID SYNC
DataType=0x0007
AccessType=rw
ParameterValue=0x00000080
DefaultValue=0x00000080

[1006]
ParameterName=Communication cycle period
DataType=0x0007
AccessType=rw
DefaultValue=500000

[1018]
SubNumber=5
ParameterName=Identity object
ObjectType=0x09

[1018sub0]
ParameterName=Highest sub-index supported
DataType=0x0005
AccessType=const
DefaultValue=0x04

[1018sub1]
ParameterName=Vendor-ID
DataType=0x0007
AccessType=ro
DefaultValue=0x004E3753
```

```
[1018sub2]
ParameterName=Product code
DataType=0x0007
AccessType=ro

[1018sub3]
ParameterName=Revision number
DataType=0x0007
AccessType=ro

[1018sub4]
ParameterName=Serial number
DataType=0x0007
AccessType=ro

[1019]
ParameterName=Synchronous counter overflow value
DataType=0x0005
AccessType=rw
DefaultValue=3

[1280]
SubNumber=4
ParameterName=SDO client parameter
ObjectType=0x09

[1280sub0]
ParameterName=Highest sub-index supported
DataType=0x0005
AccessType=const
DefaultValue=0x03

[1280sub1]
ParameterName=COB-ID client -> server (tx)
DataType=0x0007
AccessType=rw
DefaultValue=0x602

[1280sub2]
ParameterName=COB-ID server -> client (rx)
DataType=0x0007
AccessType=rw
DefaultValue=0x582

[1280sub3]
ParameterName=Node-ID of the SDO server
DataType=0x0005
AccessType=rw
DefaultValue=0x02

[1800]
SubNumber=3
ParameterName=TPDO communication parameter
ObjectType=0x09

[1800sub0]
ParameterName=Highest sub-index supported
DataType=0x0005
```

```
AccessType=const
DefaultValue=0x02

[1800sub1]
ParameterName=COB-ID used by TPDO
DataType=0x0007
AccessType=rw
DefaultValue=$NODEID+0x180

[1800sub2]
ParameterName=Transmission type
DataType=0x0005
AccessType=rw
DefaultValue=0

[1A00]
ParameterName=TPDO mapping parameter
ObjectType=0x09
DataType=0x0007
AccessType=rw
CompactSubObj=1

[1A00Value]
NrOfEntries=1
1=0x20000020

[2000]
ParameterName=TestUnsigned32
DataType=0x0007
AccessType=rw
PDOMapping=1
DefaultValue=0xC0DE1234
```

To perform a basic consistency check using all available rules, use the command presented on Listing 11.

Listing 11 – `dcfnetlint` invocation checking the `master.dcf` file.

```
$ dcfnetlint master.dcf
master.dcf:114:0: warning: Object 1280 is missing data type definition,
expected data type '0x0022' (SDO_PARAMETER) [od-1280h-datatype]
master.dcf:143:0: warning: Object 1800 is missing data type definition,
expected data type '0x0020' (PDO_COMMUNICATION_PARAMETER) [od-1800h-
datatype]
master.dcf:158:26: error: Cannot parse entry value, $NODEID is undefined
master.dcf:166:0: warning: Object 1A00 has data type '0x0007' (UNSIGNED32),
expected data type '0x0021' (PDO_MAPPING) [od-1A00h-datatype]

$ echo $?
1
```

Linter has detected some issues with that file and finished with non-zero code. One of the issue points to missing `$NODEID`, which can be fixed by providing it via argument (in base 10), as presented on Listing 12.

Listing 12 – `dcfnetlint` invocation with Node-ID set to 100.

```
$ dcfnetlint --node-id=100 master.dcf
master.dcf:114:0: warning: Object 1280 is missing data type definition,
expected data type '0x0022' (SDO_PARAMETER) [od-1280h-datatype]
master.dcf:143:0: warning: Object 1800 is missing data type definition,
expected data type '0x0020' (PDO_COMMUNICATION_PARAMETER) [od-1800h-
datatype]
master.dcf:166:0: warning: Object 1A00 has data type '0x0007' (UNSIGNED32),
expected data type '0x0021' (PDO_MAPPING) [od-1A00h-datatype]
```

The remaining issues point to missing/invalid data type definitions in some objects. However, they may be treated as false positives (for example, if used CANopen stack does not require explicit definitions for SDO/PDO parameters and communication mappings) and disabled. To configure performed checks, `--checks` argument can be used, as seen on Listing 13.

Listing 13 – Setting the list of enabled `dcfnetlint` checks.

```
$ dcfnetlint --node-id=100 --checks="*,-od-1280h-datatype,-od-1800h-
datatype,-od-1A00h-datatype" master.dcf
$ echo $?
0
```

`--checks` argument accepts ECMAScript, case-insensitive regular expressions (with minor difference of asterisk (`*`) working like glob) separated by commas (`,`) with optional minus sign (`-`) prefix that indicates the check should be disabled. The list of checks is processed from left to right. Using this argument overrides the default behaviour of enabling all checks – instead list starts as empty and is populated using the arguments. In this example, the first check enables all the checks, and next ones disable the checks that appeared earlier. Notice the list of checks is in quotes – some shells may try to expand the asterisks, which makes no sense in that context, so it's recommended to wrap the list in single- (‘) or double-quotes (“).

To verify the list of performed checks, use `--list-enabled-checks` flag, as in Listing 14.

Listing 14 – Listing enabled linter checks.

```
$ dcfnetlint --list-enabled-checks --checks="dcf-*"
dcf-comments-line-consistency
dcf-comments-line-length
dcf-comments-lines-missing
...
```

As mentioned before, asterisk is *special* and matches zero or more alphanumeric characters and dashes without prefixing for convenience. To list all supported checks, use `--list-all-checks` flag.

Linter also can print the content of object dictionary based on provided DCF file, via `--dump-objdict` argument. An example is presented on Listing 15.

Listing 15 – Dumping the content of node's object dictionary

```
$ dcfnetlint --dump-objdict master.dcf
[1000]
```

```
ParameterName=Device type
ObjectType=0x07
DataType=0x0007
AccessType=ro

...
```

The output is in DCF file format, but it may be different from DCF file provided as input – for example, compact sub objects will be expanded to full definitions, and metadata (like `DeviceInfo` section) will be lost.

`dcfnetlint` can be used to check validity of a CANopen network, if provided with CPJ file. Let's create a new DCF file describing slave node and call it `slave.dcf`. The content of that file is in Listing 16.

Listing 16 – Content of `slave.dcf` file.

```
[DeviceInfo]
VendorName=N7 Space Sp. z o.o.
VendorNumber=0x004E3753
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1

[MandatoryObjects]
SupportedObjects=3
1=0x1000
2=0x1001
3=0x1018

[OptionalObjects]
SupportedObjects=0

[ManufacturerObjects]
SupportedObjects=2
1=0x2000
2=0x2001

[1000]
ParameterName=Device type
DataType=0x0007
AccessType=ro

[1001]
ParameterName=Error register
DataType=0x0005
AccessType=ro

[1018]
SubNumber=5
ParameterName=Identity object
```

```
ObjectType=0x09

[1018sub0]
ParameterName=Highest sub-index supported
DataType=0x0005
AccessType=const
DefaultValue=0x04

[1018sub1]
ParameterName=Vendor-ID
DataType=0x0007
AccessType=ro

[1018sub2]
ParameterName=Product code
DataType=0x0007
AccessType=ro

[1018sub3]
ParameterName=Revision number
DataType=0x0007
AccessType=ro

[1018sub4]
ParameterName=Serial number
DataType=0x0007
AccessType=ro

[2000]
ParameterName=TestUnsigned64
DataType=0x001B
AccessType=rw
DefaultValue=0x1234567890ABCDEF

[2005]
ParameterName=TestDomain
ObjectType=0x02
DataType=0x000F
AccessType=rw
UploadFile=slave-upload-data.txt
DownloadFile=slave-download-data.txt
```

And the network file, `network.cpj`, as on Listing 17.

Listing 17 – Content of `network.cpj` file.

```
[Topology]
Nodes=3

Node1Present=0x1
Node1DCFName=master.dcf

Node2Present=0x1
Node2DCFName=slave.dcf

Node5Present=0x1
```

```
Node5DCFName=nonexistent-slave.dcf
```

Checking the network works the same as checking the DCF file, except the linter will return issues detected in every DCF file in the network, CPJ file, and the network itself. The command (and it's result) is shown on Listing 18.

Listing 18 – Result of linting a CANopen network.

```
dcfnetlint network.cpj
master.dcf:114:0: warning: Object 1280 is missing data type definition,
expected data type '0x0022' (SDO_PARAMETER) [od-1280h-datatype]
master.dcf:141:13: warning: The Node-ID 0x002 (2) is not SDO server [net-
sdo-matching-client-server]
master.dcf:143:0: warning: Object 1800 is missing data type definition,
expected data type '0x0020' (PDO_COMMUNICATION_PARAMETER) [od-1800h-
datatype]
master.dcf:166:0: warning: Object 1A00 has data type '0x0007' (UNSIGNED32),
expected data type '0x0021' (PDO_MAPPING) [od-1A00h-datatype]
network.cpj:1:0: warning: The network does not contain SYNC producer. [net-
sync-one-producer]
network.cpj:11:14: warning: File 'nonexistent-slave.dcf' does not exist
[cpj-topology-existing-dcfs]
slave.dcf:46:2: warning: Missing object '2001' referenced in field '2' in
section 'ManufacturerObjects' [dcf-manufacturerobjects-consistency]
slave.dcf:95:0: warning: Object '2005' is not used in MandatoryObjects nor
OptionalObjects nor ManufacturerObjects [dcf-object-consistency]
```

As in the previous example, unwanted checks can be silenced with `--checks` argument. Some checks are always reported for the CPJ file as a whole (first character of the file) – those are related to the state of the network, in this case linter have not found a SYNC producer object because the COB-ID SYNC parameter's value has SYNC generation bit set to 0. When the `ParameterValue` of object `1005` in `master.dcf` is changed to `0x40000080`, the warning disappears. The `slave.dcf` issues are caused by invalid object index (should be `2001` instead of `2005`). SDO server related warning is caused by the fact that linter does not consider default server to be enabled, if it is – the warning can be disabled. Non-existent node should be removed, and the number of nodes in the network should be changed to 2.

11.3.1.2 Capturing and analysing network traffic

To analyse network traffic, it must be captured using `candump` tool. Let's consider the following traffic log from the nodes in the network described in previous part of the tutorial (**after applying fixes mentioned there**). The log was captured using `candump -f can.log vcan-cdssw` command, and is presented on Listing 19.

Listing 19 – Example `candump` log from tutorial network.

```
(1752679501.156743) vcan-cdssw 702#00
(1752679501.156759) vcan-cdssw 701#00
(1752679501.157186) vcan-cdssw 602#4000200000000000
(1752679501.157262) vcan-cdssw 582#4100200008000000
(1752679501.157286) vcan-cdssw 602#6000000000000000
```

```
(1752679501.157324) vcan-cdssw 582#00EFCDAB90785634
(1752679501.157349) vcan-cdssw 602#7000000000000000
(1752679501.157362) vcan-cdssw 582#1D12000000000000
(1752679501.463671) vcan-cdssw 080#01
(1752679502.020113) vcan-cdssw 080#02
(1752679502.020176) vcan-cdssw 181#3412DEC0
(1752679502.419479) vcan-cdssw 602#2100200008000000
(1752679502.419588) vcan-cdssw 582#6000200000000000
(1752679502.419617) vcan-cdssw 602#0078563412DDCCBB
(1752679502.419641) vcan-cdssw 582#2000000000000000
(1752679502.419664) vcan-cdssw 602#1DAA000000000000
(1752679502.419694) vcan-cdssw 582#3000000000000000
(1752679502.479766) vcan-cdssw 080#03
(1752679503.122592) vcan-cdssw 080#04
(1752679503.122648) vcan-cdssw 181#3412DEC0
(1752679503.541131) vcan-cdssw 080#05
(1752679503.541550) vcan-cdssw 602#4000200000000000
(1752679503.541606) vcan-cdssw 582#4100200008000000
(1752679503.541638) vcan-cdssw 602#6000000000000000
(1752679503.541671) vcan-cdssw 582#0078563412DDCCBB
(1752679503.541701) vcan-cdssw 602#7000000000000000
(1752679503.541734) vcan-cdssw 582#1DAA000000000000
(1752679503.967062) vcan-cdssw 080#01
(1752679503.967121) vcan-cdssw 181#3412DEC0
(1752679504.603244) vcan-cdssw 080#02
```

Using `dcfnetmon` this log can be transformed into text or HTML report containing details of the traffic and performed transactions. To see the details in text form, use the command presented on Listing 20.

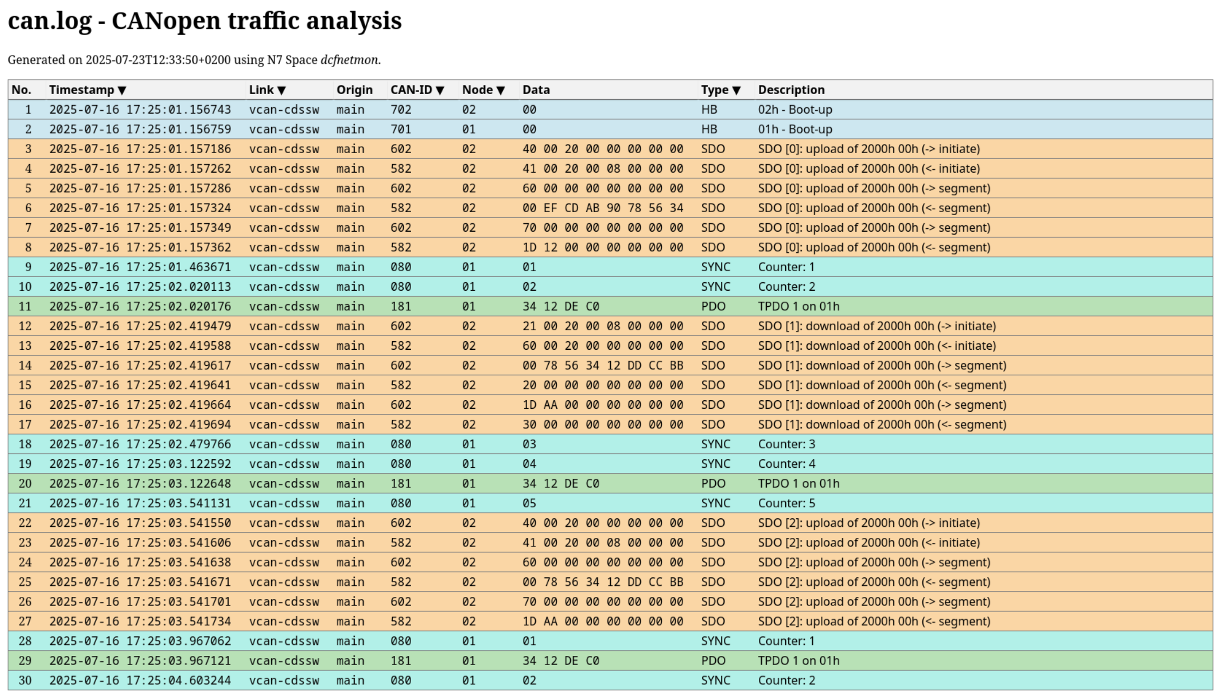
Listing 20 – Call to `dcfnetmon` analyzing the `candump` log.

```
$ dcfnetmon --network=network.cpj --output-format=text can.log
   0 1752679501.156743 Dev: vcan-cdssw Origin: main CanID: 1794 ( 0x702)
NodeID:   2 ( 0x2) Length: 1 Data: 00
      HB         state: Boot-Up
   1 1752679501.156759 Dev: vcan-cdssw Origin: main CanID: 1793 ( 0x701)
NodeID:   1 ( 0x1) Length: 1 Data: 00
      HB         state: Boot-Up
   2 1752679501.157186 Dev: vcan-cdssw Origin: main CanID: 1538 ( 0x602)
NodeID:   2 ( 0x2) Length: 8 Data: 40 00 20 00 00 00 00 00
      SDO        client->server: InitiateUploadRequest  in transaction 0
                 | Object 2000 00
...
```

HTML report can be generated by using `--output-format=html` argument and providing the output file name via `--output` argument, as seen on Listing 21. The generated report is presented on Figure 22.

Listing 21 – `dcfnetmon` call generating HTML report.

```
$ dcfnetmon --network=network.cpj --output-format=html --
output=traffic.html can.log
```



can.log - CANopen traffic analysis

Generated on 2025-07-23T12:33:50+0200 using N7 Space *dcfnetmon*.

No.	Timestamp ▼	Link ▼	Origin	CAN-ID ▼	Node ▼	Data	Type ▼	Description
1	2025-07-16 17:25:01.156743	vcan-cdssw	main	702	02	00	HB	02h - Boot-up
2	2025-07-16 17:25:01.156759	vcan-cdssw	main	701	01	00	HB	01h - Boot-up
3	2025-07-16 17:25:01.157186	vcan-cdssw	main	602	02	40 00 20 00 00 00 00 00	SDO	SDO [0]: upload of 2000h 00h (-> initiate)
4	2025-07-16 17:25:01.157262	vcan-cdssw	main	582	02	41 00 20 00 08 00 00 00	SDO	SDO [0]: upload of 2000h 00h (<- initiate)
5	2025-07-16 17:25:01.157286	vcan-cdssw	main	602	02	60 00 00 00 00 00 00 00	SDO	SDO [0]: upload of 2000h 00h (-> segment)
6	2025-07-16 17:25:01.157324	vcan-cdssw	main	582	02	00 EF CD AB 90 78 56 34	SDO	SDO [0]: upload of 2000h 00h (<- segment)
7	2025-07-16 17:25:01.157349	vcan-cdssw	main	602	02	70 00 00 00 00 00 00 00	SDO	SDO [0]: upload of 2000h 00h (-> segment)
8	2025-07-16 17:25:01.157362	vcan-cdssw	main	582	02	1D 12 00 00 00 00 00 00	SDO	SDO [0]: upload of 2000h 00h (<- segment)
9	2025-07-16 17:25:01.463671	vcan-cdssw	main	080	01	01	SYNC	Counter: 1
10	2025-07-16 17:25:02.020113	vcan-cdssw	main	080	01	02	SYNC	Counter: 2
11	2025-07-16 17:25:02.020176	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
12	2025-07-16 17:25:02.419479	vcan-cdssw	main	602	02	21 00 20 00 08 00 00 00	SDO	SDO [1]: download of 2000h 00h (-> initiate)
13	2025-07-16 17:25:02.419588	vcan-cdssw	main	582	02	60 00 20 00 00 00 00 00	SDO	SDO [1]: download of 2000h 00h (<- initiate)
14	2025-07-16 17:25:02.419617	vcan-cdssw	main	602	02	00 78 56 34 12 DD CC BB	SDO	SDO [1]: download of 2000h 00h (-> segment)
15	2025-07-16 17:25:02.419641	vcan-cdssw	main	582	02	20 00 00 00 00 00 00 00	SDO	SDO [1]: download of 2000h 00h (<- segment)
16	2025-07-16 17:25:02.419664	vcan-cdssw	main	602	02	1D AA 00 00 00 00 00 00	SDO	SDO [1]: download of 2000h 00h (-> segment)
17	2025-07-16 17:25:02.419694	vcan-cdssw	main	582	02	30 00 00 00 00 00 00 00	SDO	SDO [1]: download of 2000h 00h (<- segment)
18	2025-07-16 17:25:02.479766	vcan-cdssw	main	080	01	03	SYNC	Counter: 3
19	2025-07-16 17:25:03.122592	vcan-cdssw	main	080	01	04	SYNC	Counter: 4
20	2025-07-16 17:25:03.122648	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
21	2025-07-16 17:25:03.541131	vcan-cdssw	main	080	01	05	SYNC	Counter: 5
22	2025-07-16 17:25:03.541550	vcan-cdssw	main	602	02	40 00 20 00 00 00 00 00	SDO	SDO [2]: upload of 2000h 00h (-> initiate)
23	2025-07-16 17:25:03.541606	vcan-cdssw	main	582	02	41 00 20 00 08 00 00 00	SDO	SDO [2]: upload of 2000h 00h (<- initiate)
24	2025-07-16 17:25:03.541638	vcan-cdssw	main	602	02	60 00 00 00 00 00 00 00	SDO	SDO [2]: upload of 2000h 00h (-> segment)
25	2025-07-16 17:25:03.541671	vcan-cdssw	main	582	02	00 78 56 34 12 DD CC BB	SDO	SDO [2]: upload of 2000h 00h (<- segment)
26	2025-07-16 17:25:03.541701	vcan-cdssw	main	602	02	70 00 00 00 00 00 00 00	SDO	SDO [2]: upload of 2000h 00h (-> segment)
27	2025-07-16 17:25:03.541734	vcan-cdssw	main	582	02	1D AA 00 00 00 00 00 00	SDO	SDO [2]: upload of 2000h 00h (<- segment)
28	2025-07-16 17:25:03.967062	vcan-cdssw	main	080	01	01	SYNC	Counter: 1
29	2025-07-16 17:25:03.967121	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
30	2025-07-16 17:25:04.603244	vcan-cdssw	main	080	01	02	SYNC	Counter: 2

Figure 22 – CANopen traffic analysis report.

A detailed description of the generated HTML report is provided in chapter 11.3.2.3.

`dcfnetmon` supports multiple `candump` log formats. The format of `candump` log is auto-detected by `dcfnetmon`, but it can also specified manually if preferred. Timestamp's existence is also auto-detected, but only absolute timestamps are supported. By default, `candump -f` outputs the traffic with absolute timestamps in file format. When `candump` is logging traffic to console, it uses console format without timestamps, which is supported with `dcfnetmon --format=console --timestamp=none` command. User can pipe `candump` directly to `dcfnetmon` in that mode, but keep in mind that `dcfnetmon` will output the analyzed traffic only after the `candump` exits, so if the link does not go down by itself (or if `candump -D` is used), it's recommended to use `candump -T` or `-n` to set the idle timeout/frame limit, or kill `candump` manually with external script/command. Example call of `candump` with `dcfnetmon` is shown on Listing 22.

Listing 22 – Example of `candump` output being piped directly to `dcfnetmon`.

```
$ candump vcan-cdssw | dcfnetmon --format=console --timestamp=none
--output-format=text --network=network.cpj
   0 Dev: vcan-cdssw Origin: main CanID: 1794 ( 0x702) NodeID:   2 ( 0x2)
Length: 1 Data: 00
      HB         state: Boot-Up
   1 Dev: vcan-cdssw Origin: main CanID: 1793 ( 0x701) NodeID:   1 ( 0x1)
Length: 1 Data: 00
      HB         state: Boot-Up
   2 Dev: vcan-cdssw Origin: main CanID: 1538 ( 0x602) NodeID:   2 ( 0x2) 
Length: 8 Data: 40 00 20 00 00 00 00 00
      SDO       client->server: InitiateUploadRequest  in transaction 0
                | Object 2000 00
...
```

11.3.1.3 Simulating network traffic

`dcfnetsim` tool can be used to simulate the behavior of a single CANopen node (or the whole network, although with very limited functionality). The simulation is performed based on the network (CPJ) file, and simulation plan in YAML format, and it generates CAN traffic on a user-selected interface. For the sake of this tutorial, a virtual CAN interface called `vcan-cdssw` will be used. To create that interface, use the commands from Listing 23 (notice that super-user permissions are required).

Listing 23 – Creating a new virtual CAN interface (requires super-user privileges).

```
$ ip link add dev vcan-cdssw type vcan
$ ip link set vcan-cdssw up
```

To perform the simulation, a plan file is required. This file contains a list of sequential steps of the simulation. Listing 24 shows a simple example of a plan that resets the node (or all nodes, depending on the `dcfnetsim` arguments), waits for 500 milliseconds, sends a custom CAN frame, waits another 500ms and stops the simulation – save it as `plan.yml`.

Listing 24 – Example simulation plan.

```
config:
  cpj-file: network.cpj
  master-node-id: 1
steps:
  - type: reset
  - type: wait
    duration: 500
  - type: send-can-frame
    can-id: 0xAA
    data: [0xC0, 0xDE]
  - type: wait
    duration: 500
  - type: stop
```

Listing 25 shows how to execute that simulation, and what logs should be expected.

Listing 25 – Execution of a simulation.

```
$ dcfnetsim --plan=plan.yml --interface=vcan-cdssw
[2025-07-23T14:49:24.673363764+0200] Found slave node 2 with DCF file 
slave.dcf
[2025-07-23T14:49:24.679983316+0200] Found master node 1 with DCF file 
master.dcf
[2025-07-23T14:49:24.680223404+0200] Starting simulator on CAN interface 
vcan-cdssw based on plan plan.yml
```

```
[2025-07-23T14:49:24.680815142+0200] Master node (ID: 1) created
[2025-07-23T14:49:24.680867281+0200] Resetting master...
[2025-07-23T14:49:24.681112759+0200] Idling for 500ms...
[2025-07-23T14:49:25.182182712+0200] Sending CAN packet with ID: AA, flags:
00, data: C0 DE
[2025-07-23T14:49:25.182356123+0200] Idling for 500ms...
[2025-07-23T14:49:25.683401901+0200] Stopping the simulator!
[2025-07-23T14:49:25.683517822+0200] MasterNode destroyed
```

The simulator outputs human-readable information, but not the traffic. In order to capture the generated frames, `candump` should be used. Both simulator and `candump` can be run in a single command, as seen on Listing 26 – it's recommended to specify the idle time/frame limit for `candump` to force it to exit gracefully.

Listing 26 – Running a simulation and logging CAN traffic in parallel.

```
$ dcfnetsim --plan=plan.yml --interface=vcan-cdssw & candump vcan-cdssw -f
sim.log -T 1000
[1] 100718
Disabled standard output while logging.
Enabling Logfile 'sim.log'
[2025-07-23T14:50:10.510325443+0200] Found slave node 2 with DCF file
slave.dcf
[2025-07-23T14:50:10.517006555+0200] Found master node 1 with DCF file
master.dcf
[2025-07-23T14:50:10.517119320+0200] Starting simulator on CAN interface
vcan-cdssw based on plan plan.yml
[2025-07-23T14:50:10.517592190+0200] Master node (ID: 1) created
[2025-07-23T14:50:10.517661232+0200] Resetting master...
[2025-07-23T14:50:10.517862435+0200] Idling for 500ms...
[2025-07-23T14:50:11.021070556+0200] Sending CAN packet with ID: AA, flags:
00, data: C0 DE
[2025-07-23T14:50:11.021290104+0200] Idling for 500ms...
[2025-07-23T14:50:11.522350129+0200] Stopping the simulator!
[2025-07-23T14:50:11.522556271+0200] MasterNode destroyed
[1]  + 100718 done       dcfnetsim --plan=plan.yml --interface=vcan-cdssw
```

After running the command specified above, `sim.log` file should be created with content similar to Listing 27.

Listing 27 – CAN traffic generated by simulator.

```
(1753275010.517850) vcan-cdssw 701#00
(1753275010.826648) vcan-cdssw 080#01
(1753275011.021219) vcan-cdssw 0AA#C0DE
(1753275011.318789) vcan-cdssw 080#02
```

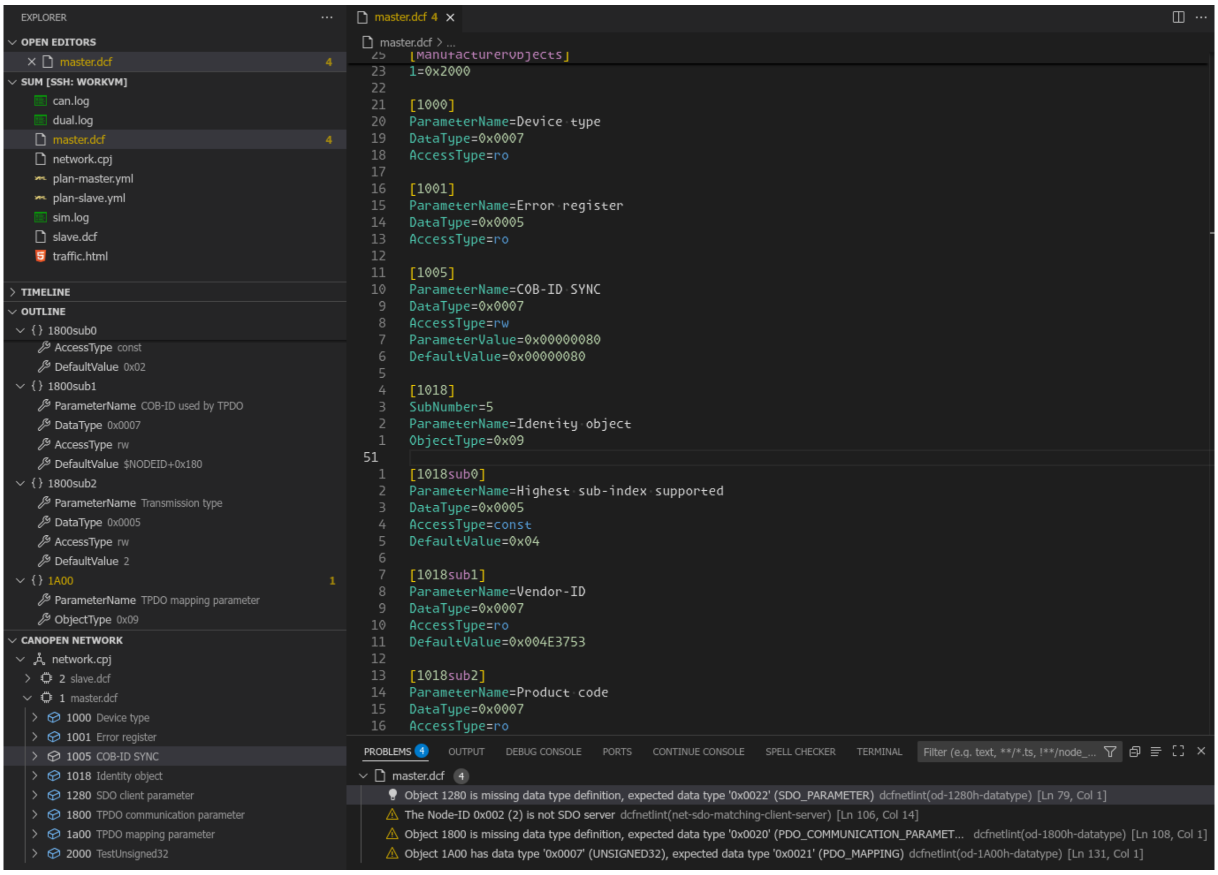
The reset step is required for most simulations – after performing it, the node starts its NMT state machine and begins processing CANopen events. Wait step is used to delay the next step of simulation – during that time, the nodes keep processing the network events (like SYNC, PDOs, SDOs, etc.). Stop step is required if the simulation should to end by itself – otherwise, it will keep running until manually stopped by the user by killing the `dcfnetsim` process.

Simulator supports variety of steps that can be used to send CAN frames, write the data to simulated node's object dictionary, and configure/trigger SYNC, PDO and SDO services. Full list of supported steps and their arguments can be found in chapter 10.4.1.3.

`dcfnetsim` can also simulate other network nodes – although in a very limited capacity. In the logs, user can notice that it found two nodes in the network – master node is the one that's currently being simulated, while all the other ones are slave nodes. Slave node simulation can be used, for example, to simulate an SDO transaction without the need of other physical nodes to exists on the network. By default, only the master node is simulated – to enable slave node simulation, use `--slave-simulation` argument.

11.3.2 GUI

After installing Visual Studio Code extension and configuring the paths to CDSSW tools, as described in chapter 9.2.2, it should be ready to work. Opening the directory with files created during the previous chapter of the tutorial and viewing the `master.dcf` file should result in a similar view to the one on Figure 23.



```
{} 1800sub0
    AccessType const
    DefaultValue 0x02
{} 1800sub1
    ParameterName COB-ID used by TPDO
    DataType 0x0007
    AccessType rw
    DefaultValue $NODEID+0x180
{} 1800sub2
    ParameterName Transmission type
    DataType 0x0005
    AccessType rw
    DefaultValue 2
{} 1A00                                  1
    ParameterName TPDO mapping parameter
    ObjectType 0x09
```

```
network.cpj
  2 slave.dcf
  1 master.dcf
    1000 Device type
    1001 Error register
    1005 COB-ID SYNC
    1018 Identity object
    1280 SDO client parameter
    1800 TPDO communication parameter
    1a00 TPDO mapping parameter
    2000 TestUnsigned32
```

```
[ManufacturerObjects]
1=0x2000

[1000]
ParameterName=Device type
DataType=0x0007
AccessType=ro

[1001]
ParameterName=Error register
DataType=0x0005
AccessType=ro

[1005]
ParameterName=COB-ID SYNC
DataType=0x0007
AccessType=rw
ParameterValue=0x00000080
DefaultValue=0x00000080

[1018]
SubNumber=5
ParameterName=Identity object
ObjectType=0x09

[1018sub0]
ParameterName=Highest sub-index supported
DataType=0x0005
AccessType=const
DefaultValue=0x04

[1018sub1]
ParameterName=Vendor-ID
DataType=0x0007
AccessType=ro
DefaultValue=0x004E3753

[1018sub2]
ParameterName=Product code
DataType=0x0007
AccessType=ro
```

```
master.dcf 4
  Object 1280 is missing data type definition, expected data type '0x0022' (SDO_PARAMETER) dcfnetlint(od-1280h-datatype) [Ln 79, Col 1]
  The Node-ID 0x002 (2) is not SDO server dcfnetlint(net-sdo-matching-client-server) [Ln 106, Col 14]
  Object 1800 is missing data type definition, expected data type '0x0020' (PDO_COMMUNICATION_PARAMET... dcfnetlint(od-1800h-datatype) [Ln 108, Col 1]
  Object 1A00 has data type '0x0007' (UNSIGNED32), expected data type '0x0021' (PDO_MAPPING) dcfnetlint(od-1A00h-datatype) [Ln 131, Col 1]
```

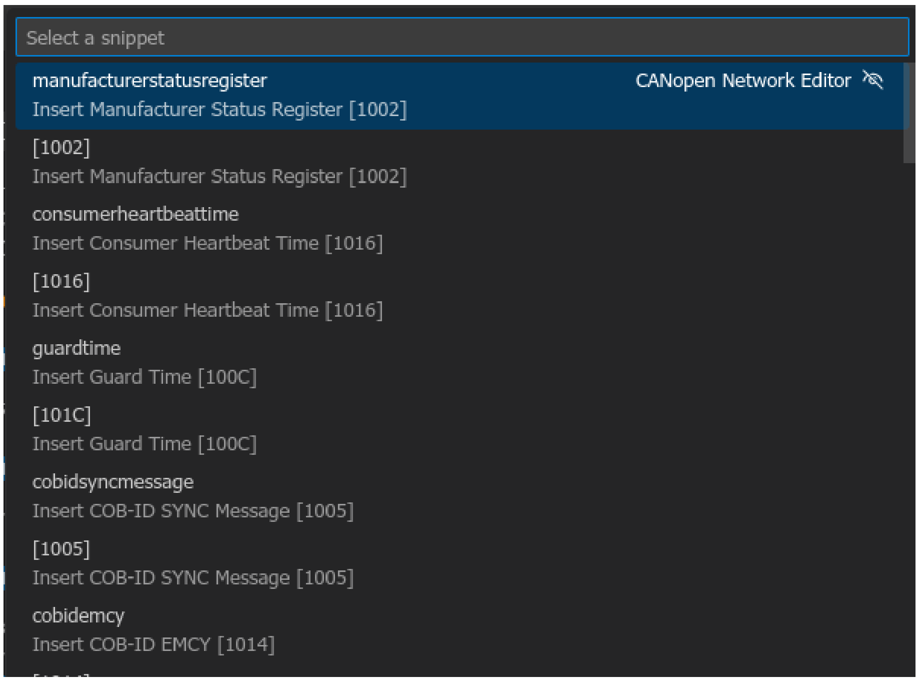
Figure 23 - Visual Studio Code with properly configured DCF tools

If the syntax highlighting is active, and user can see the issues detected in that file, then the plugin is installed and configured correctly.

11.3.2.1 Linting and modifying the network

LSP will continuously lint the network detected in the current workspace and provide real-time diagnostics and utility features to Visual Studio Code.

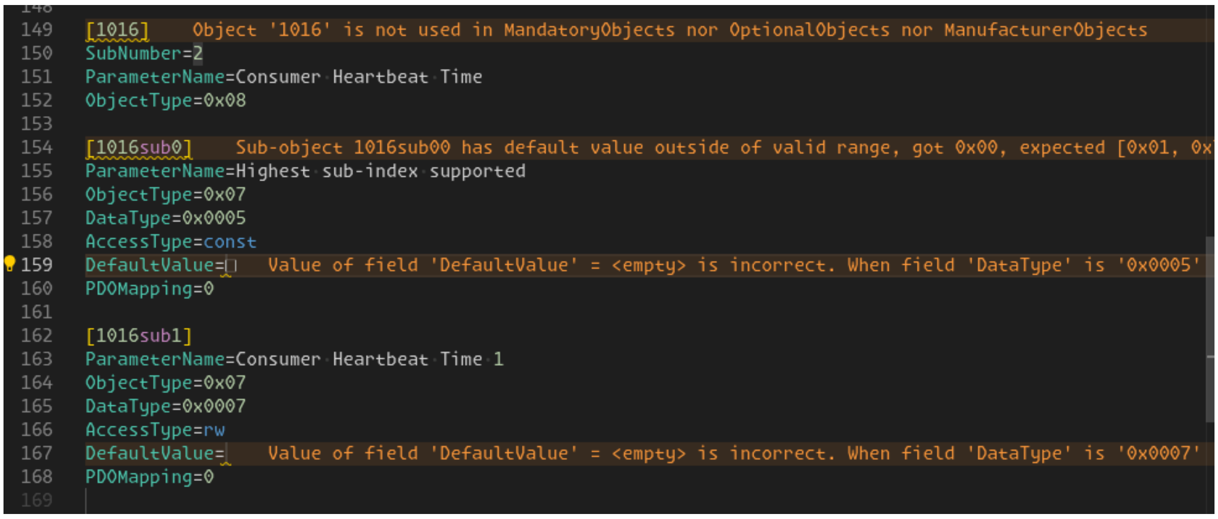
To simplify the creation of common objects in DCF file, CANopen Network Editor provides multiple snippets. To insert a snippet, trigger the command menu with Ctrl+Shift+P shortcut and use the “Snippets: Insert Snippet” option. User should see a list of snippets similar to one on Figure 24.



```
Select a snippet
manufacturerstatusregister                                   CANopen Network Editor
Insert Manufacturer Status Register [1002]
[1002]
Insert Manufacturer Status Register [1002]
consumerheartbeattime
Insert Consumer Heartbeat Time [1016]
[1016]
Insert Consumer Heartbeat Time [1016]
guardtime
Insert Guard Time [100C]
[101C]
Insert Guard Time [100C]
cobidsyncmessage
Insert COB-ID SYNC Message [1005]
[1005]
Insert COB-ID SYNC Message [1005]
cobidemcy
Insert COB-ID EMCY [1014]
```

Figure 24 – List of available snippets.

After selecting one of the snippets, the editor will automatically insert selected object (and required sub-objects) and fill the fields with standard-defined values, as seen on Figure 25. Remaining fields must be filled manually, and editor provides a handy way of jumping between them using Tab key.

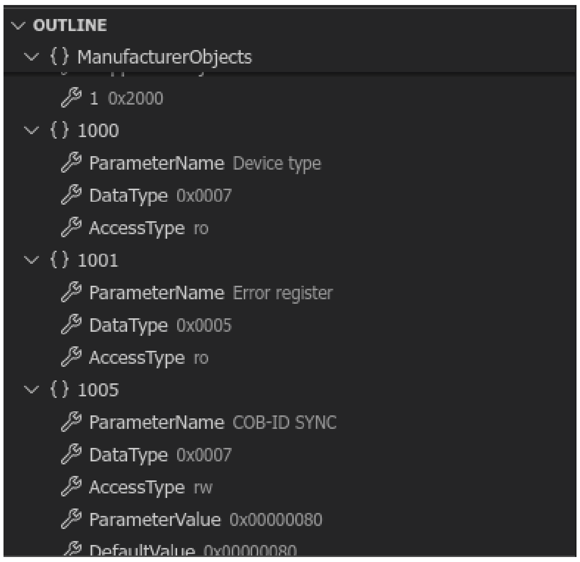


```
[1016]    Object '1016' is not used in MandatoryObjects nor OptionalObjects nor ManufacturerObjects
SubNumber=2
ParameterName=Consumer Heartbeat Time
ObjectType=0x08

[1016sub0]    Sub-object 1016sub00 has default value outside of valid range, got 0x00, expected [0x01, 0x
ParameterName=Highest sub-index supported
ObjectType=0x07
DataType=0x0005
AccessType=const
DefaultValue=    Value of field 'DefaultValue' = <empty> is incorrect. When field 'DataType' is '0x0005'
PDOMapping=0

[1016sub1]
ParameterName=Consumer Heartbeat Time 1
ObjectType=0x07
DataType=0x0007
AccessType=rw
DefaultValue=    Value of field 'DefaultValue' = <empty> is incorrect. When field 'DataType' is '0x0007'
PDOMapping=0
```

Figure 25 – Inserted object definition snippet.

Keep in mind, however, that newly created objects must be added to `Mandatory/Optional/ManufacturerObject` sections manually.

All the objects (and their properties) in currently opened DCF file are visible in Outline view, shown on Figure 26.

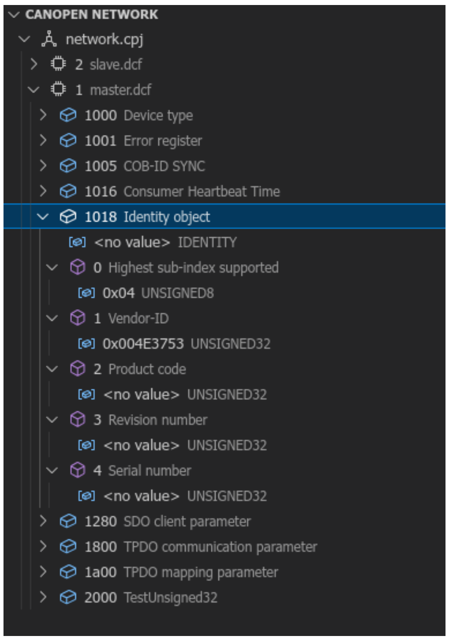


```
OUTLINE
  { } ManufacturerObjects
      1 0x2000
  { } 1000
      ParameterName Device type
      DataType 0x0007
      AccessType ro
  { } 1001
      ParameterName Error register
      DataType 0x0005
      AccessType ro
  { } 1005
      ParameterName COB-ID SYNC
      DataType 0x0007
      AccessType rw
      ParameterValue 0x00000080
      DefaultValue 0x00000080
```

Figure 26 – Outline view of a DCF file.

Clicking on an item in that view will move the cursor to selected property, and double-clicking will select it.

The plugin also provides an overview of the network – see Figure 27. If a CPJ file is found in the current workspace, CANopen Network view will show a tree with nodes and their objects, along with their properties.

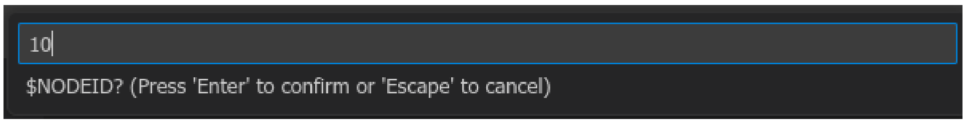


```
CANOPEN NETWORK
  network.cpj
    2 slave.dcf
    1 master.dcf
      1000 Device type
      1001 Error register
      1005 COB-ID SYNC
      1016 Consumer Heartbeat Time
      1018 Identity object
        <no value> IDENTITY
        0 Highest sub-index supported
          0x04 UNSIGNED8
        1 Vendor-ID
          0x004E3753 UNSIGNED32
        2 Product code
          <no value> UNSIGNED32
        3 Revision number
          <no value> UNSIGNED32
        4 Serial number
          <no value> UNSIGNED32
      1280 SDO client parameter
      1800 TPDO communication parameter
      1a00 TPDO mapping parameter
      2000 TestUnsigned32
```

Figure 27 – CANopen network view.

This view allows user to easily add new nodes to the network. To create a new network node, right-click on the network and select "New CANopen node" option.

An input window will show at the top of the screen, asking for the Node-ID, as seen on Figure 28

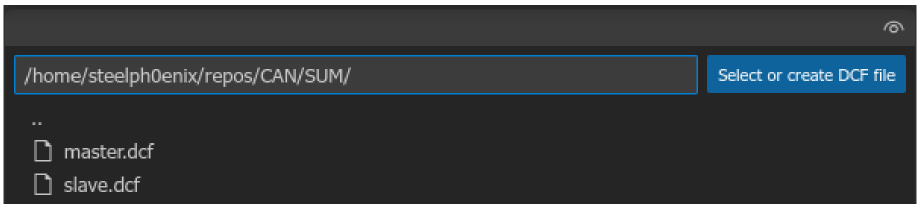


```
10
$NODEID? (Press 'Enter' to confirm or 'Escape' to cancel)
```

Figure 28 - Setting the Node-ID of created node.

After providing it, user can either choose to create a node from scratch, or using an existing DCF file. If "Create new DCF file" option is chosen, an empty DCF file will be created, and a new entry will be added to the CPJ file. Note that the "Nodes" option in the "Topology" section will remain unchanged, so it must be manually set to the correct number of nodes afterwards.

If the "Use existing DCF file" option is chosen, a file selection window will appear, similar to one on Figure 29.

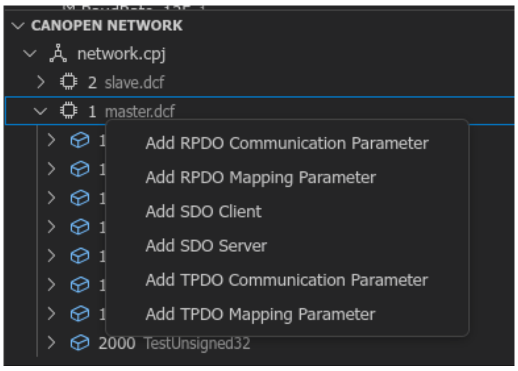


```
/home/steelph0enix/repos/CAN/SUM/
..
master.dcf
slave.dcf
```

Figure 29 – Selecting an existing DCF file for new CANopen node.

After selecting the DCF, a new entry will be added to the CPJ file. The DCF file will not be copied or modified.

This view also provides a way of adding new PDO parameters/mappings and SDO clients/servers to selected nodes, as presented on Figure 30. To add one, right-click a node in the network tree and select the option from the list.



CANOPEN NETWORK
network.cpj
2 slave.dcf
1 master.dcf
Add RPDO Communication Parameter
Add RPDO Mapping Parameter
Add SDO Client
Add SDO Server
Add TPDO Communication Parameter
Add TPDO Mapping Parameter
2000 TestUnsigned32

Figure 30 – Adding new objects to CANopen node via network view.

After that, a snippet will appear in the selected DCF file. It works just like the snippets described earlier.

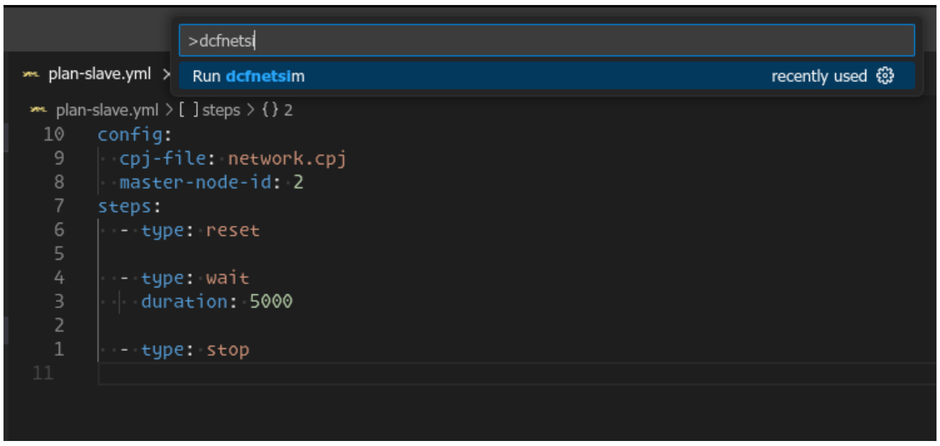
11.3.2.2 Editing network simulation plans

Visual Studio Code provides syntax highlighting for YAML files out of the box, however the DCF extension also provides snippets for all supported simulation steps and simulation file template.

Usage of those snippets is exactly the same as snippets for DCF files, except they will be enabled only in YAML files. In order to select a snippet, either start writing the step name (list should automatically pop up and let user select appropriate snippet), or open the command window (Ctrl+Shift+P) and select “Snippets: Insert snippet” option.

11.3.2.3 Performing network simulations

The editor is integrated with *dcfnetsim*, and provides a Visual Studio Code command to run node/network simulation. To start one, user must first open the simulation plan file and focus the editor on it. Then, open the command window (Ctrl+Shift+P) and select the “Run dcfnetsim” option, as shown on Figure 31.



```yaml
config:
  cpj-file: network.cpj
  master-node-id: 2
steps:
  - type: reset

  - type: wait
    duration: 5000

  - type: stop
```

Figure 31 – Executing currently opened simulation plan.

Afterwards, user will be asked for the network interface name (see Figure 32), and whether to enable the slave node simulation or not.

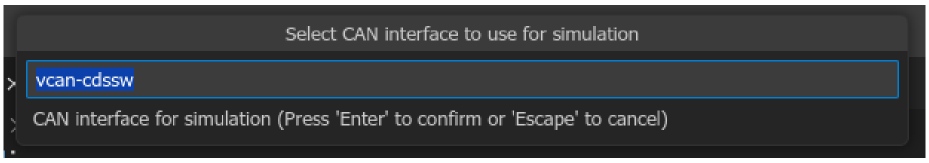
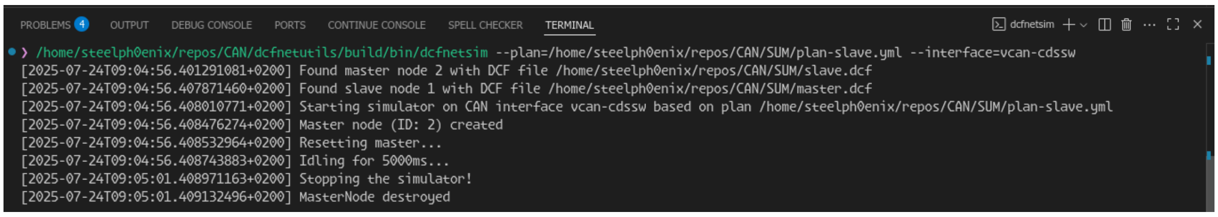

Select CAN interface to use for simulation
vcan-cdssw
CAN interface for simulation (Press 'Enter' to confirm or 'Escape' to cancel)

Figure 32 – Selecting the CAN interface for simulation.

After specifying the arguments, a new command line window will be created (or reused) and `dcfnetsim` will begin execution – it should look similarly to the command line window on Figure 33. The plugin currently does not support invoking candump in parallel, so in order to record the simulation session it must be invoked manually.

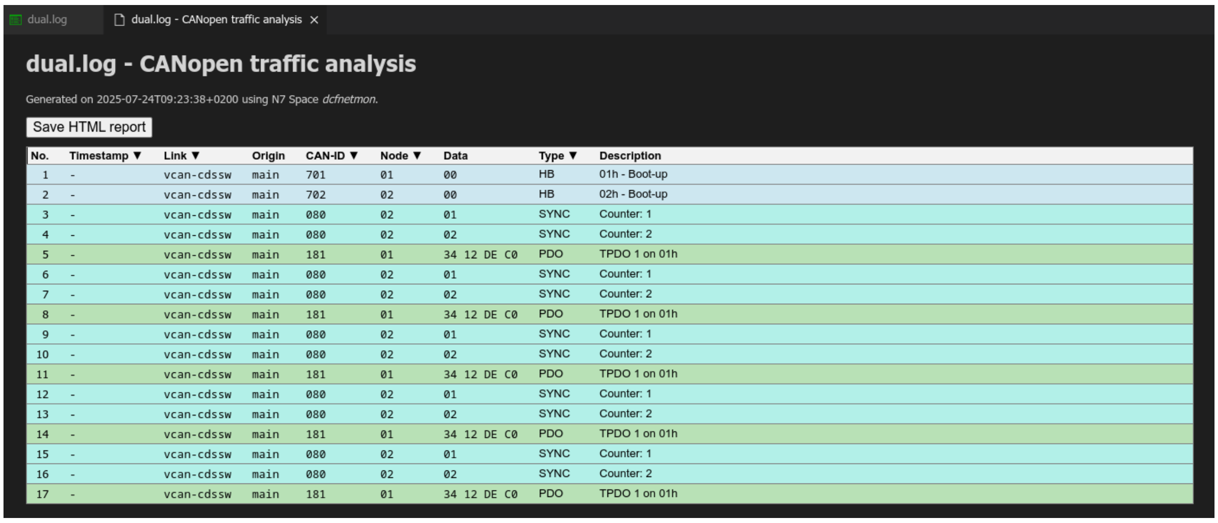


```
❯ /home/steelph0enix/repos/CAN/dcfnetutils/build/bin/dcfnetsim --plan=/home/steelph0enix/repos/CAN/SUM/plan-slave.yml --interface=vcan-cdssw
[2025-07-24T09:04:56.401291081+0200] Found master node 2 with DCF file /home/steelph0enix/repos/CAN/SUM/slave.dcf
[2025-07-24T09:04:56.407871460+0200] Found slave node 1 with DCF file /home/steelph0enix/repos/CAN/SUM/master.dcf
[2025-07-24T09:04:56.408010771+0200] Starting simulator on CAN interface vcan-cdssw based on plan /home/steelph0enix/repos/CAN/SUM/plan-slave.yml
[2025-07-24T09:04:56.408476274+0200] Master node (ID: 2) created
[2025-07-24T09:04:56.408532964+0200] Resetting master...
[2025-07-24T09:04:56.408743883+0200] Idling for 5000ms...
[2025-07-24T09:05:01.408971163+0200] Stopping the simulator!
[2025-07-24T09:05:01.409132496+0200] MasterNode destroyed
```

Figure 33 – Network simulation logs.

11.3.2.4 Generating and viewing dcfnetmon report

The plugin also provides integration with dcfnetmon. To generate and view a report, open the command window (Ctrl+Shift+P) and select the “Run dcfnetmon” option. User will be asked for the path of `candump` log file, and its format (user can use “auto” to trigger auto-detection of the log/timestamp format). After specifying those, a new tab will open in Visual Studio Code with the generated report, as shown on Figure 34.



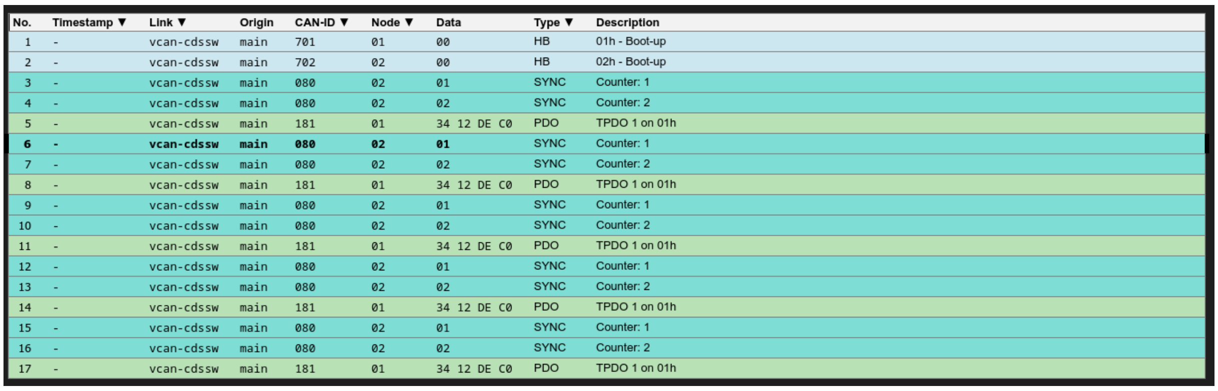
dual.log - CANopen traffic analysis

Generated on 2025-07-24T09:23:38+0200 using N7 Space *dcfnetmon*.

No.	Timestamp ▼	Link ▼	Origin	CAN-ID ▼	Node ▼	Data	Type ▼	Description
1	-	vcan-cdssw	main	701	01	00	HB	01h - Boot-up
2	-	vcan-cdssw	main	702	02	00	HB	02h - Boot-up
3	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
4	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
5	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
6	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
7	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
8	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
9	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
10	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
11	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
12	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
13	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
14	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
15	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
16	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
17	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h

Figure 34 – Generated monitor report in Visual Studio Code.

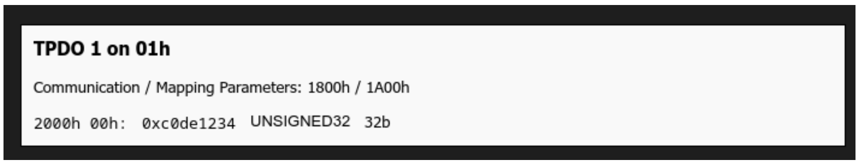
The generated report is interactive. Clicking on a frame will highlight all the related rows, for example – clicking on a SYNC frame will highlight all the other SYNC frames, as seen on Figure 35. User can also move between the frames using W/S keys. Holding Shift key while navigating with keyboard will restrict the movement to the currently selected group of frames.



No.	Timestamp ▼	Link ▼	Origin	CAN-ID ▼	Node ▼	Data	Type ▼	Description
1	-	vcan-cdssw	main	701	01	00	HB	01h - Boot-up
2	-	vcan-cdssw	main	702	02	00	HB	02h - Boot-up
3	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
4	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
5	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
6	**-**	**vcan-cdssw**	**main**	**080**	**02**	**01**	SYNC	Counter: 1
7	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
8	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
9	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
10	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
11	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
12	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
13	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
14	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h
15	-	vcan-cdssw	main	080	02	01	SYNC	Counter: 1
16	-	vcan-cdssw	main	080	02	02	SYNC	Counter: 2
17	-	vcan-cdssw	main	181	01	34 12 DE C0	PDO	TPDO 1 on 01h

Figure 35 – Group highlighting in the monitor report.

Some frames contain additional information that isn’t visible by default. For example, after clicking on a PDO frame, a box (see Figure 36) with more details will appear next to the table.



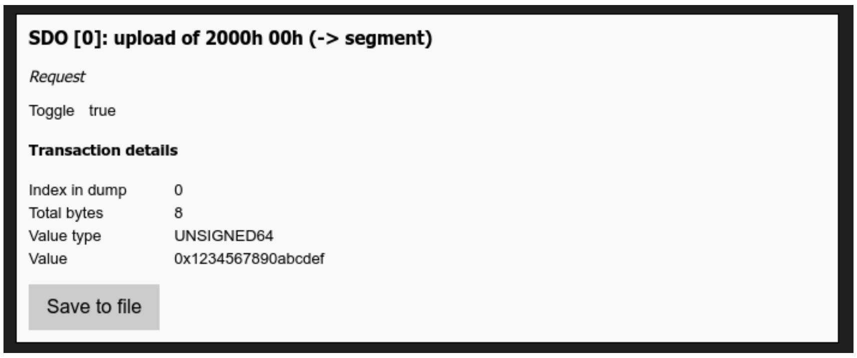
TPDO 1 on 01h

Communication / Mapping Parameters: 1800h / 1A00h

```
2000h 00h:  0xc0de1234  UNSIGNED32  32b
```

Figure 36 – Details of a PDO frame.

Those details contain the interpreted data from the selected frame, and information about the object specifying the PDO. Similar details are provided for SDO transactions, as shown on Figure 37. It's also possible to save the data transferred via SDO directly to a binary file.



SDO [0]: upload of 2000h 00h (-> segment)

Request

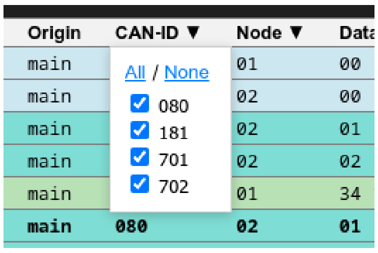
Toggle true

Transaction details

Index in dump	0
Total bytes	8
Value type	UNSIGNED64
Value	0x1234567890abcdef

Figure 37 – Details of an SDO frame and transaction.

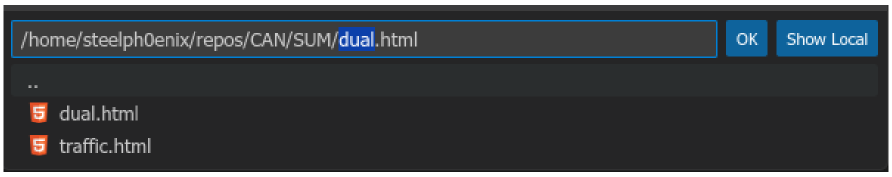
The table header contains arrows that, when clicked on, reveal the filters (see Figure 38). Filtering can be done based on the frame timestamp (if provided), CAN link, CAN ID, Node ID, and frame type.



Origin	CAN-ID ▼	Node ▼	Data
main		01	00
main		02	00
main		02	01
main		02	02
main		01	34
main	**080**	**02**	**01**

Figure 38 – CAN-ID filter in monitor report.

When the report is opened in Visual Studio Code, the "Save HTML report" button will be created above the table. Clicking it will open the file selection window (see Figure 39) and copy the report from memory to a selected file.



```
/home/steelph0enix/repos/CAN/SUM/dual.html
..
dual.html
traffic.html
```

Figure 39 – Saving the generated report as HTML file.

12 Analytical Index

N/A

13 Lists

13.1 List of Tables

13.2 List of Figures

13.3 List of Listings