ETAS VECU-BUILDER V1.2

User Guide

www.etas.com

VECU-BUILDER V1.2 I User Guide R07 EN I 04.2023

Contents

1 Safety and Privacy Information

In this chapter you can find information about the intended use, the addressed target group, and information about safety and privacy related topics.

1.1 Intended Use

The product is designed to produce a virtual ECU for microcontrollers from existing ECU source codes or from precompiled binaries. The virtual ECU is designed for simulation, debugging, and pre-calibration of ECU software in a PC-based virtual simulation environment.
In general, virtual ECUs may not be real-time capable. If you control physical devices with a virtual ECU, the system may respond unexpectedly. Take suitable precautions to ensure safe operation.
ETAS GmbH cannot be made liable for damage which is caused by incorrect use and not adhering to the safety information. Please adhere to the ETAS Safety Advice (see `documentation` folder).

1.2 Target Group

This product is directed at trained qualified personnel in development of automotive ECU software (e.g., function developer, application engineer, ECU software integrator, system engineer or calibration engineer) at OEMs, tier-1 or tier-2 suppliers in the auto-motive industry. Technical knowledge in control unit engineering is a prerequisite. In addition, programming knowledge in C/C++ is required. AUTOSAR Classic knowledge is helpful.

1.3 Privacy Notice

Your privacy is important to ETAS. We have created the following privacy notice that informs you, which data are processed in VECU-BUILDER, which data categories VECU-BUILDER uses, and which technical measure you must take to ensure the privacy of the users. Additionally, we provide further instructions where this product stores and where you can delete personal data.

1.3.1 Data Processing

Note that personal data or data categories are processed when using this product (e.g. in log files). The purchaser of this product is responsible for the legal conformity of processing the data in accordance with Article 4 No. 7 of the General Data Protection Regulation (GDPR). As the manufacturer, ETAS GmbH is not liable for any mishandling of this data.

1.3.2 Technical and Organizational Measures

This product itself does not encrypt the personal data or data categories that it records. Ensure the data security of the recorded data by suitable technical or organizational measures of your IT system, e.g., by classical anti-theft and access protection on the hardware. Personal data in log files can be deleted by tools in the operating system.

2 About VECU-BUILDER

VECU-BUILDER is designed to build a virtual ECU (vECU). The vECU can be used for simulation, debugging and pre-calibration of ECU software in a PC-based virtual simulation environment.

VECU-BUILDER supports the generation of Level-1, Level-2, and Level-3 vECUs according to the Prostep Definition of vECUs. Level-4 vECUs, i.e., hex-files for a specific target, are not supported.

VECU-BUILDER is based on Python and CMake. The inputs can either be C/C++ source codes or binaries like object files or shared libraries including symbol information. In contrast to AUTOSAR Classic, the configuration of a vECU is done in a single YAML file (`vEcuConf.yaml`). No ARXML files are processed. The properties are configured in this text-based file. This file is used to define the supported features of the vECU such as an XCP slave or initial data as part of simulated NVRAM. VECU-BUILDER wraps the binaries of the vECU into an FMU (FMI 2.0 for Co-Simulation). These FMUs can be integrated into any FMI-compliant simulation master.

2.1 Basics

The basic principle is to keep the data lean in a simple and smart way. The concept is the simplification of the ECU software stack and the ARXML file. The A2L file is patched by removing all hardware dependencies and updating memory addresses of all inputs, outputs, measurements, and characteristics. The software stack layers are represented by C and H files which are reflected in the `imported` folder (`vECU\imported`) in the vECU build process. The result is a stand-alone FMU containing the model description (e.g. its variables) as XML file, the access to calibration and measurement variables via patched A2L file and an executable model as DLL/SO file.

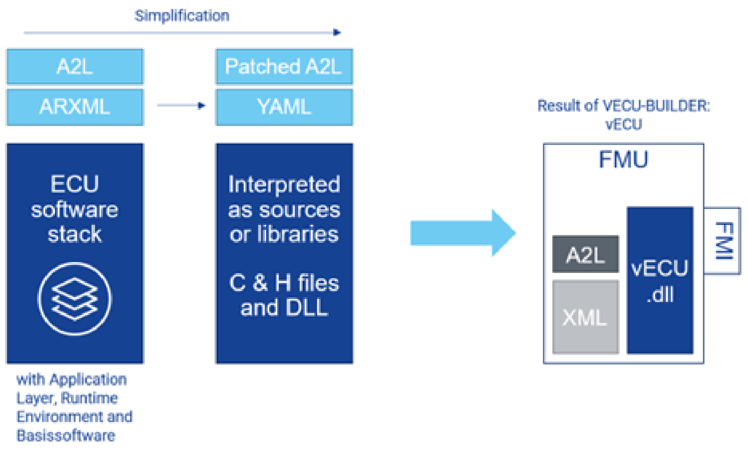

Simplification
A2L
ARXML
Patched A2L
YAML
ECU software stack
with Application Layer, Runtime Environment and Basissoftware
Interpreted as sources or libraries
C & H files and DLL
Result of VECU-BUILDER: vECU
FMU
A2L
XML
vECU .dll
FMI

Fig. 2-1: Basic concept and result of VECU-BUILDER

2.2 Virtual ECU

A vECU is a virtualized ECU which can be used as a real ECU. With the vECU you can test the ECU software and execute the software functionality without hardware. This gives you the possibility to test the communication between the ECUs before prototypes or hardware is available. The vECU contains the code, the parameters and the XCP slave as an alternative path to the hex code.

2.3 vECU Creation Process Workflow

The whole workflow is an iterative process to get to the final configuration of the YAML file. The listed points give a rough overview of the workflow. Section A and F are taking place out of the VECU-BUILDER.

A. Prepare sources
    - Directives that refer to header files in code must be fixed
    - Generate a script collecting the files you need from the various locations you found
B. Compile sources, incompatible sources must be removed
    - Generate new workspace
    - Copy sources into workspace
    - Build
    - Check error messages
    - Remove or patch code
C. Link sources and create stubs
    - Solve link errors with empty stubs
D. Define Inputs and Outputs (I/O) to make the vECU runnable
    - Use symbol information to generate I/O
    - Manually patch the sources of virtual devices
    - Use the C notation of the variables (e.g., `sensor.*`)
E. Create task model to run the tasks
    - Use text format to define task model
F. Operate for first time, apply SiL specific code changes
    - Debug code
    - Fill some stub functions with code or apply SiL specific code changes

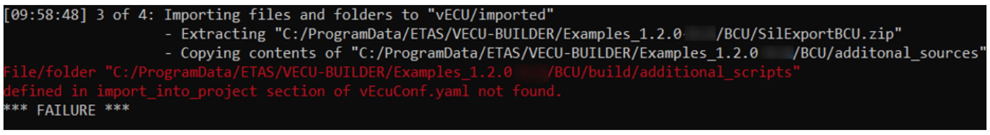
After building the first iteration of an vECU it can be used to perform further steps like (out of VECU-BUILDER):

- Integrate vECU with plant models and execute it in Co-Simulation-environment
- Run and test the vECU in an experiment environment
- Measurement and calibration of vECU
- Debugging with source code editor

2.4 Warning and Error Messages

VECU-BUILDER may encounter situations in which an Error or a Warning message is displayed.

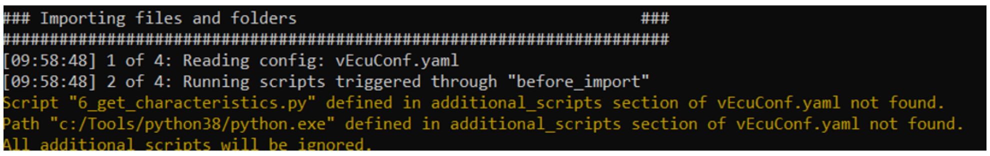
Errors are printed in red and indicate a severe issue which prevents the build from succeeding.



```
[09:58:48] 3 of 4: Importing files and folders to "vECU/imported"
                   - Extracting "C:/ProgramData/ETAS/VECU-BUILDER/Examples_1.2.0[redacted]/BCU/SilExportBCU.zip"
                   - Copying contents of "C:/ProgramData/ETAS/VECU-BUILDER/Examples_1.2.0[redacted]/BCU/additonal_sources"
File/folder "C:/ProgramData/ETAS/VECU-BUILDER/Examples_1.2.0[redacted]/BCU/build/additional_scripts"
defined in import_into_project section of vEcuConf.yaml not found.
*** FAILURE ***
```

Fig. 2-2: Error message

Warnings are printed in yellow and are meant to draw the attention to a certain issue during the build. The issue is not as severe as an error and thus the build continues.



```
### Importing files and folders                                    ###
######################################################################
[09:58:48] 1 of 4: Reading config: vEcuConf.yaml
[09:58:48] 2 of 4: Running scripts triggered through "before_import"
Script "6_get_characteristics.py" defined in additional_scripts section of vEcuConf.yaml not found.
Path "c:/Tools/python38/python.exe" defined in additional_scripts section of vEcuConf.yaml not found.
All additional scripts will be ignored.
```

Fig. 2-3: Warning message

3 Installation

This chapter provides information for preparing and performing the installation and for licensing the software. The installation can be fulfilled for the following operating systems:

- Windows 10
- Ubuntu 20.04 LTS

3.1 Hardware Requirements

The following Hardware Requirements need to be met:

Processor	min. 2 GHz 3 GHz Dual-Core or higher recommended
Memory	min. 8 GB RAM 32 GB RAM recommended
Free Disk Space	5 GB (not including the size for application data) >100 GB recommended

3.2 Preparation

Prior to the installation, check that your computer meets the Hardware and Software Requirements. Depending on the operating system used and network connection, you must ensure that you have the required user rights.

Note

Ensure that you have the necessary access privileges for the installation of the software. If in doubt, contact your system administrator.

3.3 Installation Content

The installation content can either be downloaded from ETAS license and download portal (https://license.etas.com/flexnet/operationsportal/logon.do) and then be installed or installed from the DVD.

It contains information about the open-source software attributions, important information like Safety Advice or the User Guide and the executable installation file (EXE).

3.4 Licensing

The use of VECU-BUILDER is protected by electronic licensing. Valid licenses are neces-sary to operate ETAS VECU-BUILDER and its add-ons. The use of unli-censed ETAS soft-ware is prohibited. The required licenses are not included in this delivery.

When you purchase VECU-BUILDER licenses, you receive a separate entitlement letter. Activate the license using a self-service portal on the ETAS website: https://www.etas.com/support/licensing

For assistance, please consult the help file available on the start page of the self-service portal. During the activation process, you receive the necessary license keys per e-mail.

License keys are valid for a major version. If you have a valid service contract, you will receive a new entitlement automatically for successive major version (e.g., from V4.x to V5.x). You do not need a new license file for updates and main-tenance versions, e.g., for the refresh V5.0.1 or update V5.1.0 to major version V5.0.0.

VECU-BUILDER checks

- the product license when building FMUs.
- the runtime license during tun-time of the vECU.
- the XCP license before establishing an XCP connection.
- the GO license during build-time. If it is valid, it will prevent all license checks during run-time.

3.5 Installation on Windows 10

3.5.1 Software Requirements for Windows 10

The following Software Requirements need to be met:

Required Software	CMake (version ≥3.15)
Recommended Software	Notepad++
Optional Software	Microsoft Visual Studio 2015, 2017, 2019, 2022 Microsoft Visual Studio Code Python

3.5.2 Manual Installation

1. Go to the directory where the installation file is located and execute the `VECU_BUILDER_installer_1.2.0.exe` file.

⇨ The Setup Wizard opens.

2. Klick **Next**.

⇨ The "End User License Agreement" window opens.

3. Read the License Agreement carefully, then select I accept the terms of the License Agreement.
4. Klick **Next**.

⇨ The "Safety Advice" window opens.

5. Read the Safety Advice carefully, then select I read and selected the Safety Advice.
6. Klick **Next**.

⇨ The "Installation Path" window opens.

7. Accept the default path (click **Next**) or click **Browse** to select a custom loc-ation.

⇨ The "Ready to Install" window opens.

8. Click **Install**.

⇨ The installation is performed, its progress is shown via a progress bar.

9. Click **Next**.

⇨ The "Third-party Software" window opens.

10. Install CMake (required) and Notepad++ (recommended).

    See the links below in the installation dialog:

    CMake (version 3.15 or higher)

    Notepad++
11. Click **Next**.

⇨ The "Completing VECU-BUILDER Setup" window opens.

12. Optionally, activate the Open VECU-BUILDER documentation checkbox to open the documentation folder.
13. Click Finish.

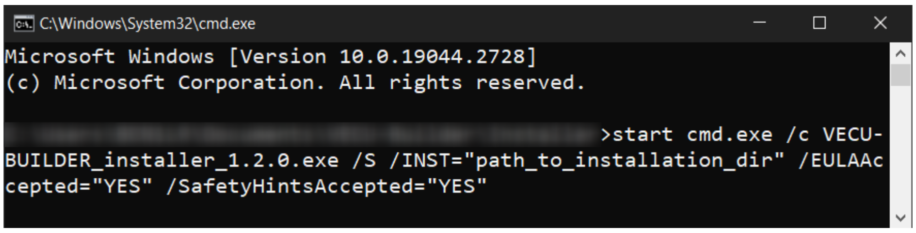
⇨ The installation is completed, and the VECU-BUILDER can now be used.

3.5.3 Silent Installation

Besides the Manual Installation, you also can use the Silent Installation. Installation differs between using the Command Prompt and the PowerShell.

Silent Installation using Command Prompt

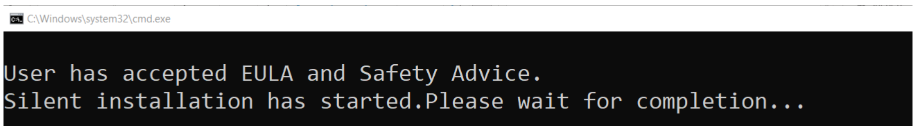
1. Open the command prompt.
2. Navigate to the directory where the installer (`VECU-BUILDER_installer_1.2.0.exe`) is located.
3. Execute the following command:
   `start cmd.exe /c VECU-BUILDER_installer_1.2.0.exe /S /INST="path_to_installation_dir" /EULAAccepted="YES" /SafetyHintsAccepted="YES"`
   where `path_to_installation_dir` contains a path to a directory where the software is to be installed.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2728]
(c) Microsoft Corporation. All rights reserved.

[redacted]>start cmd.exe /c VECU-BUILDER_installer_1.2.0.exe /S /INST="path_to_installation_dir" /EULAAccepted="YES" /SafetyHintsAccepted="YES"
```

⇨ A new command prompt window opens and installation starts.

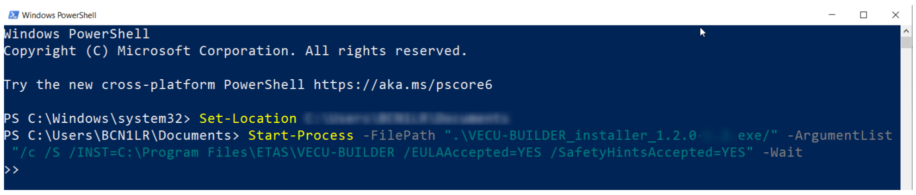


```
C:\Windows\system32\cmd.exe

User has accepted EULA and Safety Advice.
Silent installation has started.Please wait for completion...
```

Silent Installation using PowerShell

1. Open the PowerShell.
2. Navigate to the directory where the installer (`VECU-BUILDER_installer_1.2.0.exe`) is located.
3. Execute the following command:
   Start-Process -FilePath `".\VECU-BUILDER_installer_1.2.0.exe/" -ArgumentList "/c /S /INST= path_to_installation_dir /EULAAccepted=YES /SafetyHintsAccepted=YES" -Wait`
   where `path_to_installation_dir` contains a path to a directory where the software is to be installed.



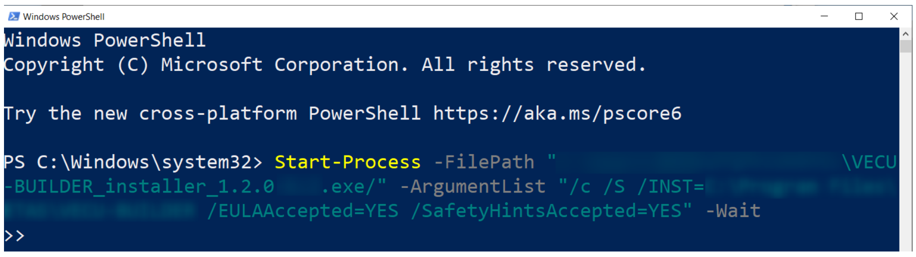
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> Set-Location [redacted]
PS C:\Users\BCN1LR\Documents> Start-Process -FilePath ".\VECU-BUILDER_installer_1.2.0[redacted]exe/" -ArgumentList
 "/c /S /INST=C:\Program Files\ETAS\VECU-BUILDER /EULAAccepted=YES /SafetyHintsAccepted=YES" -Wait
>>
```

Or

4. Execute the following command:
   `Start-Process -FilePath " path_to \VECU-BUILDER_installer_1.2.0.exe/" -ArgumentList "/c /S /INST= path_to_installation_dir /EULAAccepted=YES /SafetyHint-sAccepted=YES" -Wait`
   where `path_to` contains the path where the installer (`VECU-BUILDER_installer_1.2.0.exe`) is located and `path_to_installation_dir` contains a path to a directory where the software is to be installed.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> Start-Process -FilePath "[redacted]\VECU
-BUILDER_installer_1.2.0[redacted].exe/" -ArgumentList "/c /S /INST=[redacted]
[redacted] /EULAAccepted=YES /SafetyHintsAccepted=YES" -Wait
>>
```

⇨ Installation starts.

```
User has accepted EULA and Safety Advice.
Silent installation has started.Please wait for completion...
```

3.5.4 Uninstalling on Windows 10

1. Open the location where you installed VECU-BUILDER.
   If you used the default installation location, you can find it under
   `C:\Program Files\ETAS\VECU-BUILDER`
2. Execute the `uninstall.exe` with double-click.

3.6 Installation on Ubuntu 20.04 LTS

3.6.1 Software Requirements for Ubuntu 20.04 LTS

The following Software Requirements need to be met:

Required Software	cmake build-essential gcc-multilib g++-multilib libssl-dev:i386 linux-libc-dev:i386 nano xterm
Optional Software	Microsoft Visual Studio Code Python

3.6.2 Installing on Ubuntu 20.04 LTS

1. Navigate to the directory where the Debian Software Package file `VECU-BUILDER_installer_1.2.0.deb` is located.
2. Open a new terminal.
3. Execute the command of

   ```
   sudo apt install ./VECU-BUILDER_installer_1.2.0.deb
   ```
4. Enter your password.
5. Accept the EULA.
6. Accept the Safety Advice.

⇨ The VECU-BUILDER package deployment is completed.

7. Logout and login to enable environment variables to be set.
8. For Ubuntu 20.04 LTS, VECU-BUILDER has dependencies to other software. Make sure the software is installed prior to using VECU-BUILDER. You may use the following commands to install the required software:
   - `sudo apt install cmake`
   - `sudo apt install build-essential`
   - `sudo apt install gcc-multilib`
   - `sudo apt install g++-multilib`
   - `sudo apt install libssl-dev:i386`
   - `sudo apt install linux-libc-dev:i386`
   - `sudo apt install nano`
   - `sudo apt install xterm`

3.6.3 Uninstalling on Ubuntu 20.04 LTS

1. Open a new terminal.
2. Execute the command of

   `sudo apt remove vecu-builder`

⇨ You are asked if you want to continue uninstalling.

3. To continue, enter `Y` and hit **Enter**.

⇨ The VECU-BUILDER package is removed.

3.7 Installed Files and Folders

VECU-BUILDER Tool

The default installation location is
`C:\Program Files\ETAS\VECU-BUILDER\1.2.0` on **Windows**

Or

`/opt/ETAS/VECU-BUILDER/1.2.0` on **Ubuntu 20.04 LTS**.

It is recommended not to alter the installation location.

An environment variable of `VECUBUILDER_HOME` points to this folder.

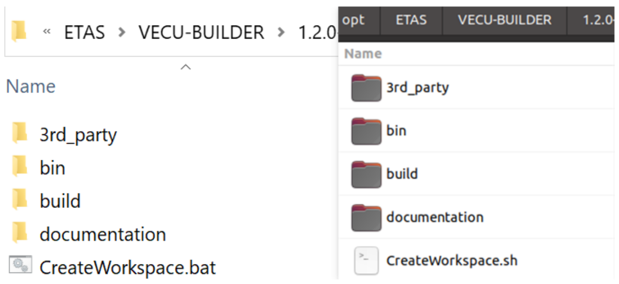


Fig. 3-1: Installation content (left: Windows, right: Ubuntu 20.04 LTS)

The content of this folder consists of several subfolders and one command script:

- `3rd_party`: Contains the third party tools of FMU Checker and MinGW.
- `bin`: Contains library and execution files for the build process. These files are important for the build and must not be altered.
- `build`: Contains templates, resources, and scripts for the build process. These files are important for the build and must not be altered.
- `documentation`: Contains the VECU-BUILDER User Guide, the OSS Attribution and the ETAS Safety Advice documents.
- `CreateWorkspace.bat` (Windows) / `CreateWorkspace.sh` (Ubuntu 20.04 LTS): Creates a new workspace. After executing, you will be guided through the process step by step.

VECU-BUILDER Examples

You can find ready-to-use examples in the following location:

`C:\ProgramData\ETAS\VECU-BUILDER\Examples_1.2.0` on **Windows**

Or

`/opt/ETAS/VECU-BUILDER/Examples_1.2.0` on **Ubuntu 20.04 LTS**.

An environment variable of `VECUBUILDER_EXAMPLES` points to this folder.

The following two examples are delivered along with the tool:

- BCU (Body Control Unit)
- SimpleExample

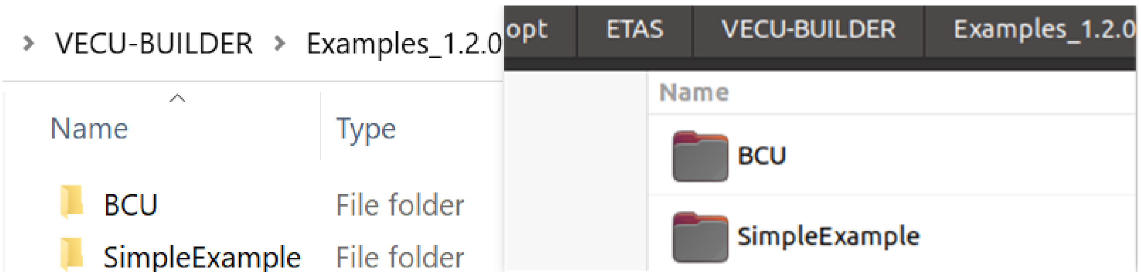


Fig. 3-2: Delivered examples (left: Windows, right: Ubuntu 20.04 LTS)

VECU-BUILDER Workspaces

As location for all your workspaces we recommend the default folder, where you should create a dedicated subfolder for each workspace.

The default folder is created during the installation process on **Windows** under `C:\Users\Public\Documents\VECU-BUILDER_Workspaces`

Or

`/opt/ETAS/VECU-BUILDER_Workspaces` on **Ubuntu 20.04 LTS**.

Access to Artefacts in Windows

You can access all artefacts in Windows via their respective Start Menu entries.

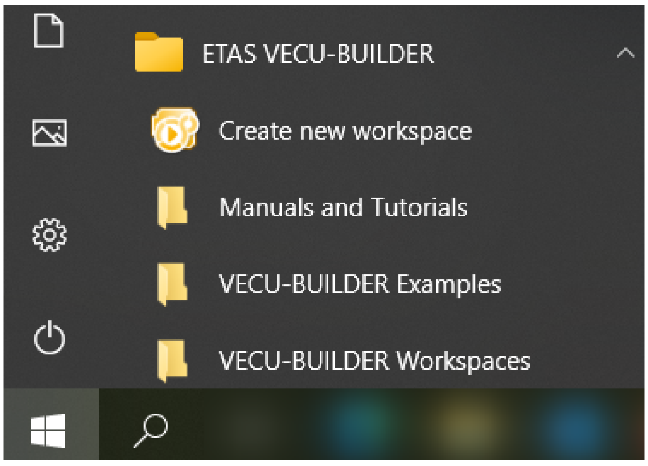


Fig. 3-3: Start Menu entries

4 Working with VECU-BUILDER

To commence your learning, we recommend following the bellow path:

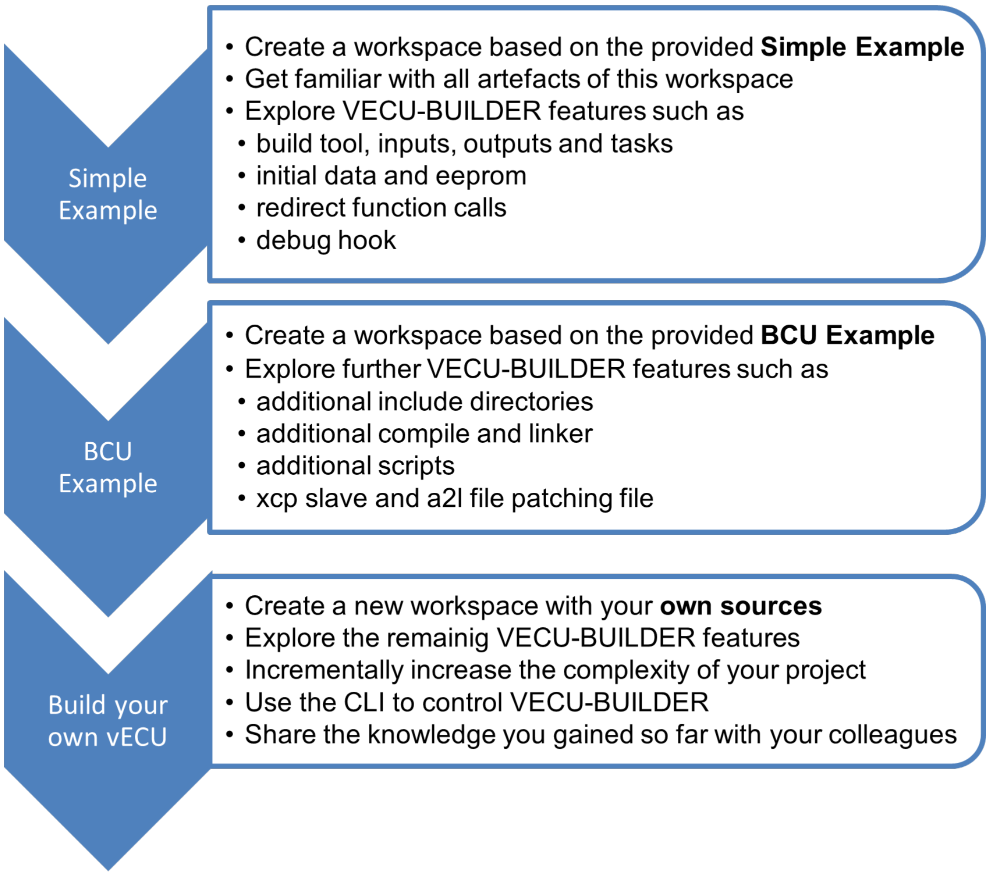

Simple Example
• Create a workspace based on the provided **Simple Example**
• Get familiar with all artefacts of this workspace
• Explore VECU-BUILDER features such as
• build tool, inputs, outputs and tasks
• initial data and eeprom
• redirect function calls
• debug hook
BCU Example
• Create a workspace based on the provided **BCU Example**
• Explore further VECU-BUILDER features such as
• additional include directories
• additional compile and linker
• additional scripts
• xcp slave and a2l file patching file
Build your own vECU
• Create a new workspace with your **own sources**
• Explore the remainig VECU-BUILDER features
• Incrementally increase the complexity of your project
• Use the CLI to control VECU-BUILDER
• Share the knowledge you gained so far with your colleagues

Fig. 4-1: The learning path

This section guides you through the process of creating a vECU in four distinct stages. Each stage can be triggered individually, and you can choose to continue with the next one.

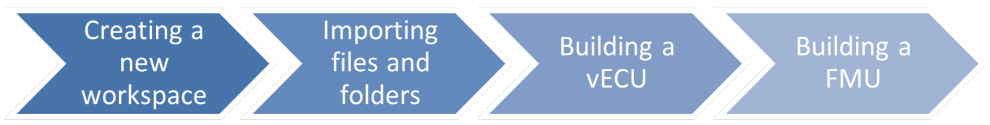


Fig. 4-2: VECU-BUILDER stages

By following the steps described in the next chapters, you will build your first vECU based on the Simple Example. This is the ideal starting point for your vir-tualization leaning journey.

4.1 Creating a New Workspace

The very first step, required at the beginning of every project, is to create a workspace.

Note

Workspaces are designed for parallel use.

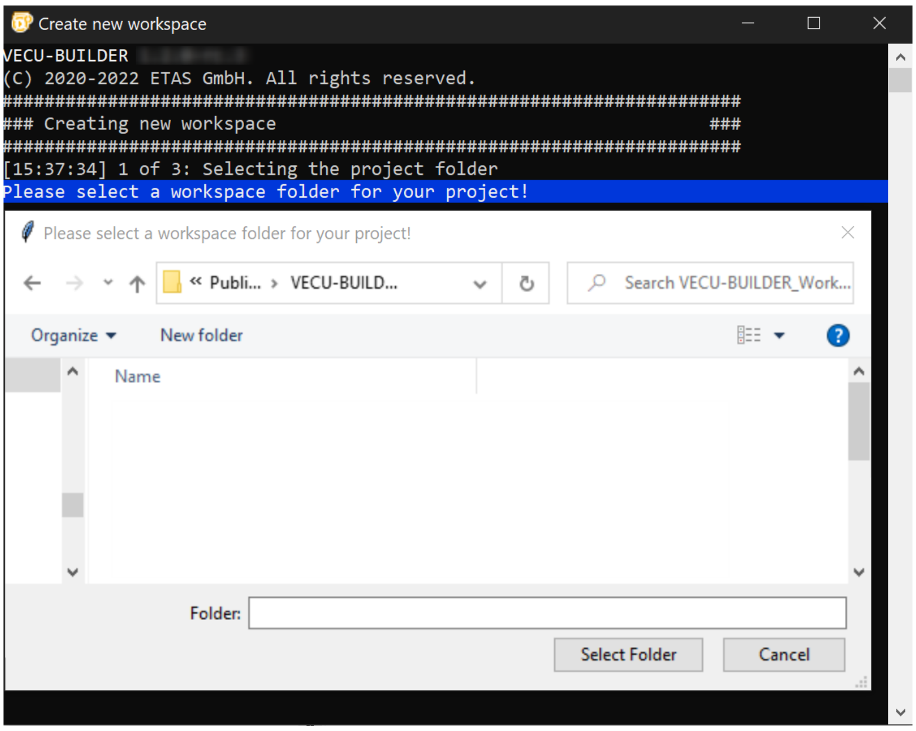
A single workspace cannot be used for tasks running in parallel.

4.1.1 Creating a Workspace on Windows

1. Launch “Create new workspace” from the Start Menu.

⇨ A console window opens providing details on the overall process, various stages it goes through and their individual steps.

⇨ In the first step of “Create new workspace” you will be asked to select a folder where your workspace will be saved.



```
VECU-BUILDER [redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
########################################################################
### Creating new workspace                                           ###
########################################################################
[15:37:34] 1 of 3: Selecting the project folder
Please select a workspace folder for your project!
```

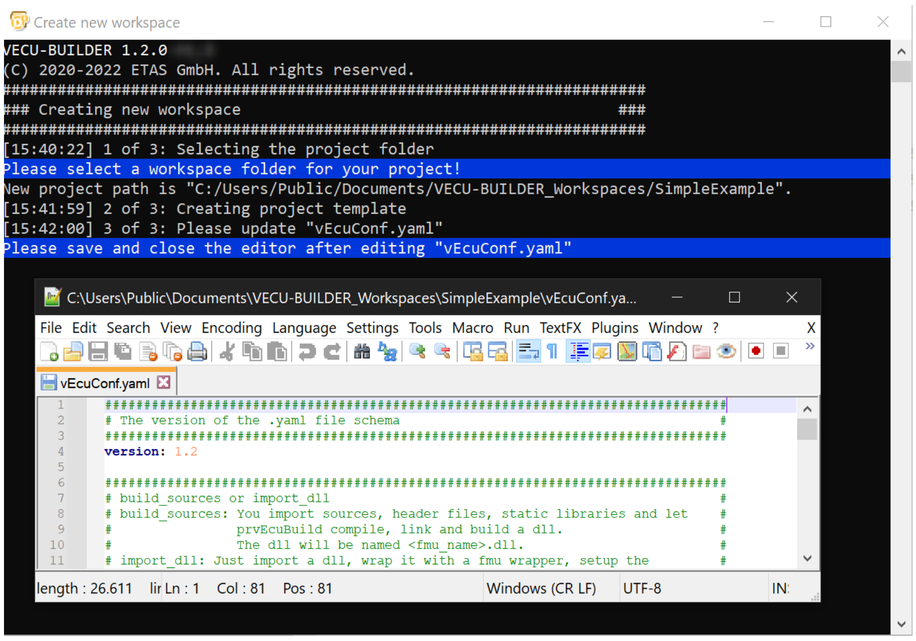
Create new workspace
Please select a workspace folder for your project!
« Publi... › VECU-BUILD...
Search VECU-BUILDER_Work...
Organize
New folder
Name
Folder:
Select Folder
Cancel

Fig. 4-3: Select workspace location (Windows)

2. Navigate to the default location of your workspaces `C:\Users\Public\Documents\VECU-BUILDER_Workspaces` and select an existing folder or create a new one.

⇨ The configuration file `vEcuConf.yaml` opens in Notepad++.

⇨ Per default, this is the configuration file of the Simple Example.


Create new workspace
C:\Users\Public\Documents\VECU-BUILDER_Workspaces\SimpleExample\vEcuConf.ya...
File Edit Search View Encoding Language Settings Tools Macro Run TextFX Plugins Window ? X
vEcuConf.yaml

```
VECU-BUILDER 1.2.0
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Creating new workspace                                         ###
######################################################################
[15:40:22] 1 of 3: Selecting the project folder
Please select a workspace folder for your project!
New project path is "C:/Users/Public/Documents/VECU-BUILDER_Workspaces/SimpleExample".
[15:41:59] 2 of 3: Creating project template
[15:42:00] 3 of 3: Please update "vEcuConf.yaml"
Please save and close the editor after editing "vEcuConf.yaml"
```

```
################################################################################
# The version of the .yaml file schema                                         #
################################################################################
version: 1.2

################################################################################
# build_sources or import_dll                                                  #
# build_sources: You import sources, header files, static libraries and let    #
#                prvEcuBuild compile, link and build a dll.                    #
#                The dll will be named <fmu_name>.dll.                         #
# import_dll: Just import a dll, wrap it with a fmu wrapper, setup the         #
```

```
length : 26.611   lir Ln : 1   Col : 81   Pos : 81          Windows (CR LF)   UTF-8        IN:
```

Fig. 4-4: Default configuration (Windows)

3. Keep the configuration file as is and close the Notepad ++ application.

⇨ Your new workspace is now created.

The process will automatically continue with the next stage.

4.1.2 Creating a Workspace on Ubuntu 20.04 LTS

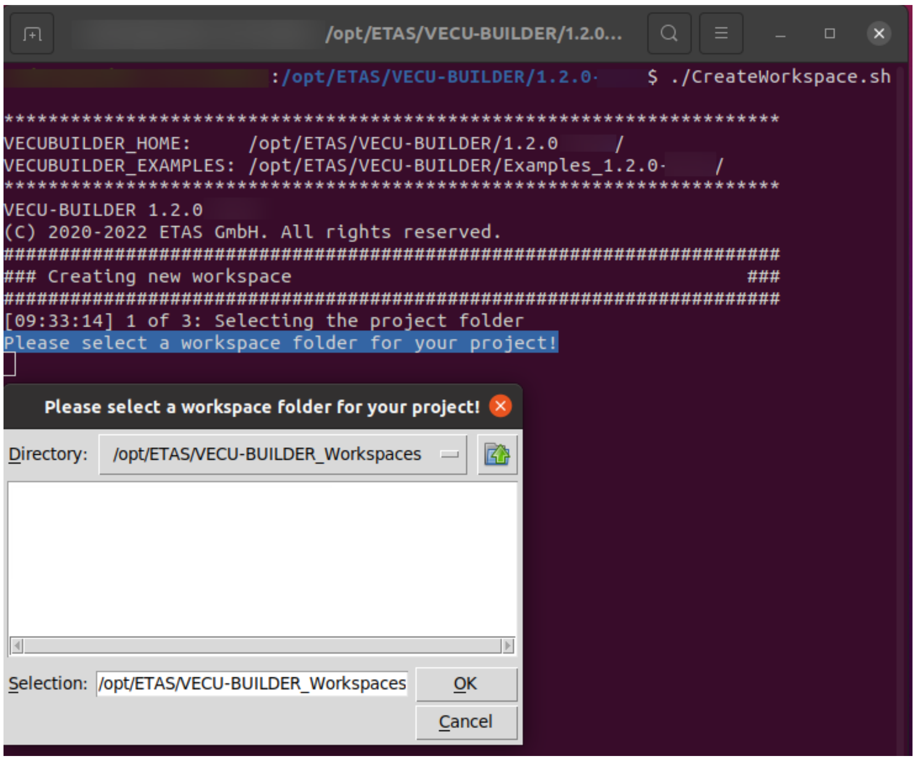
Note

In Ubuntu 20.04 LTS the folder, that should be used as workspace, needs to exist before the workspace creation is proceeded.

1. Navigate to the folder, where the `CreateWorkspace.sh` is located. The default path is `opt/ETAS/VECU-BUILDER/1.2.0`.
2. Open a new terminal.
3. Enter `./CreateWorkspace.sh`.

⇨ In the first step of "Create new workspace" you will be asked to select a folder where your workspace will be saved.

4. Navigate to the default location of your workspaces `/opt/ETAS/VECU-BUILDER_Workspaces` and select an existing folder.



```
[redacted] /opt/ETAS/VECU-BUILDER/1.2.0...
[redacted]:/opt/ETAS/VECU-BUILDER/1.2.0-[redacted]$ ./CreateWorkspace.sh

**********************************************************************
VECUBUILDER_HOME:     /opt/ETAS/VECU-BUILDER/1.2.0[redacted]/
VECUBUILDER_EXAMPLES: /opt/ETAS/VECU-BUILDER/Examples_1.2.0-[redacted]/
**********************************************************************
VECU-BUILDER 1.2.0[redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Creating new workspace                                         ###
######################################################################
[09:33:14] 1 of 3: Selecting the project folder
Please select a workspace folder for your project!
```

Please select a workspace folder for your project!

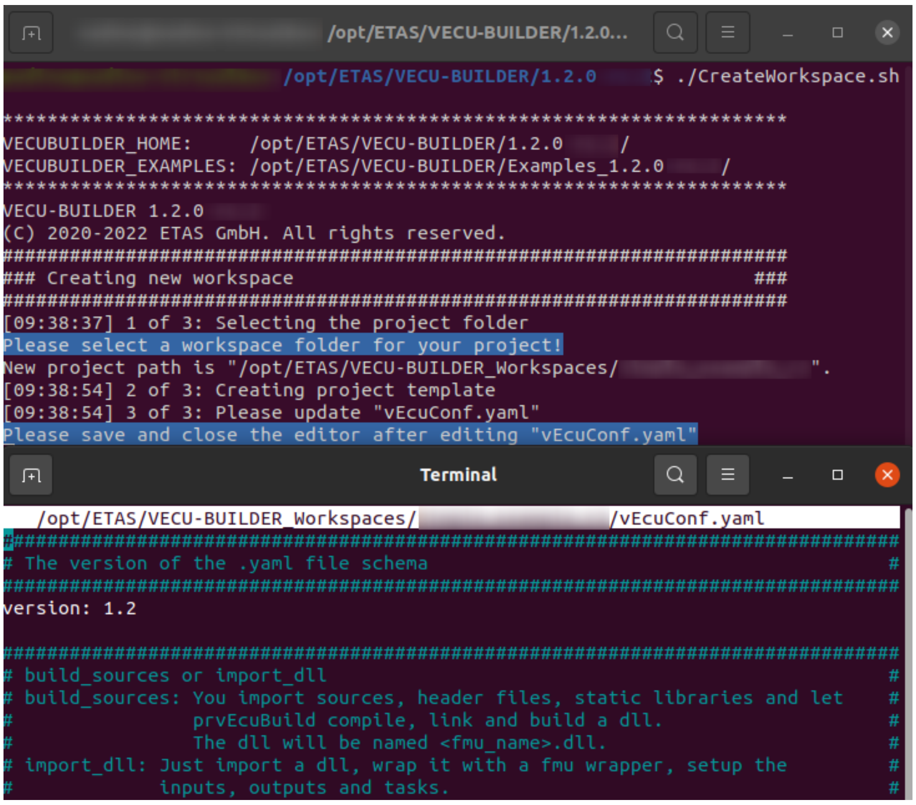
Directory: /opt/ETAS/VECU-BUILDER_Workspaces

Selection: /opt/ETAS/VECU-BUILDER_Workspaces

Fig. 4-5: Select workspace location (Ubuntu 20.04 LTS)

⇨ The configuration file `vEcuConf.yaml` opens.

⇨ Per default, this is the configuration file of the Simple Example.



```
[redacted] /opt/ETAS/VECU-BUILDER/1.2.0[redacted] $ ./CreateWorkspace.sh

**********************************************************************
VECUBUILDER_HOME:     /opt/ETAS/VECU-BUILDER/1.2.0[redacted]/
VECUBUILDER_EXAMPLES: /opt/ETAS/VECU-BUILDER/Examples_1.2.0[redacted]/
**********************************************************************
VECU-BUILDER 1.2.0[redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Creating new workspace                                         ###
######################################################################
[09:38:37] 1 of 3: Selecting the project folder
Please select a workspace folder for your project!
New project path is "/opt/ETAS/VECU-BUILDER_Workspaces/[redacted]".
[09:38:54] 2 of 3: Creating project template
[09:38:54] 3 of 3: Please update "vEcuConf.yaml"
Please save and close the editor after editing "vEcuConf.yaml"
```

```
   /opt/ETAS/VECU-BUILDER_Workspaces/[redacted]/vEcuConf.yaml
################################################################################
# The version of the .yaml file schema                                         #
################################################################################
version: 1.2

################################################################################
# build_sources or import_dll                                                  #
# build_sources: You import sources, header files, static libraries and let    #
#                prvEcuBuild compile, link and build a dll.                    #
#                The dll will be named <fmu_name>.dll.                         #
# import_dll: Just import a dll, wrap it with a fmu wrapper, setup the         #
#             inputs, outputs and tasks.                                       #
```

Fig. 4-6: Default configuration (Ubuntu 20.04 LTS)

5. Keep the configuration file as is and close it.

⇨ Your new workspace is now created.

The process will automatically continue with the next stage.

4.2 Importing Files and Folders

During this stage, the sources defined in your `vEcuConf.yaml` are copied to the "`vEcu/imported`" folder in your workspace.

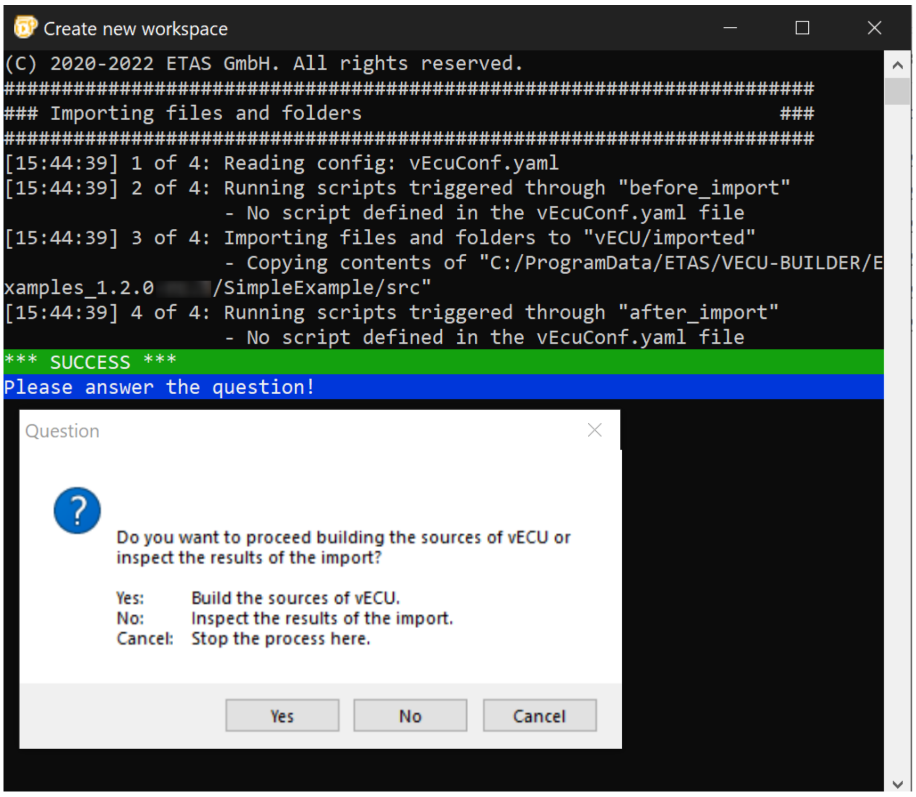
Note

During the import stage, files and folders get copied into the workspace. For reasons of portability, it is recommended to create workspaces that are self-contained.

After successful completion of the previous stage Creating a New Workspace you were forwarded to the next stage Importing Files and Folders and the process continues.

If you work in an already existing workspace, you can trigger this stage by running `1_Import.bat` on **Windows** or `1_Import.sh` on **Ubuntu 20.04 LTS**.

⇨ After successful completion of this stage Importing Files and Folders a dialog opens asking you whether you want to continue with the next stage Building the vECU or inspect the results of this stage.



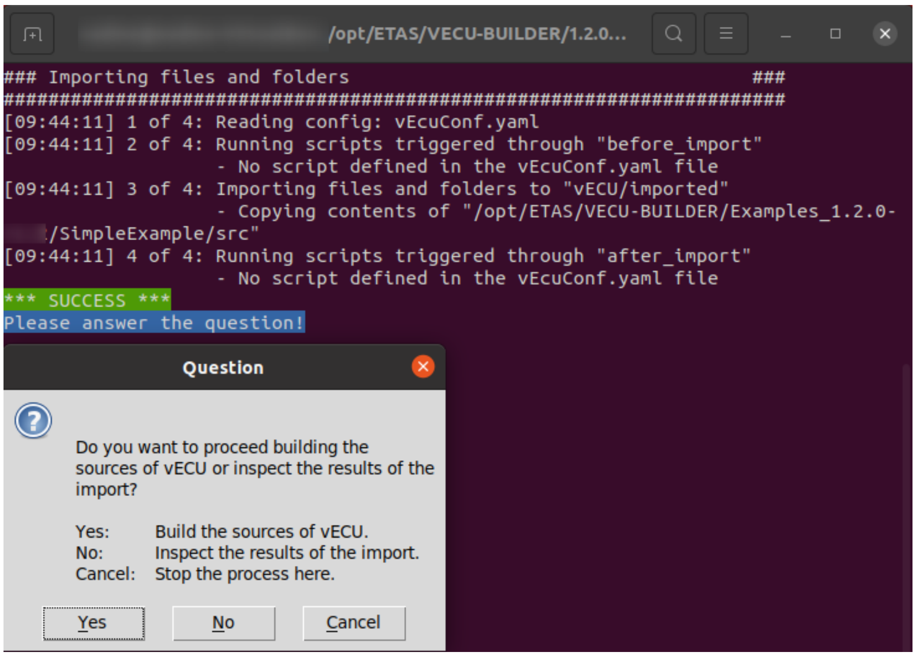
```
(C) 2020-2022 ETAS GmbH. All rights reserved.
########################################################################
### Importing files and folders                                      ###
########################################################################
[15:44:39] 1 of 4: Reading config: vEcuConf.yaml
[15:44:39] 2 of 4: Running scripts triggered through "before_import"
                   - No script defined in the vEcuConf.yaml file
[15:44:39] 3 of 4: Importing files and folders to "vECU/imported"
                   - Copying contents of "C:/ProgramData/ETAS/VECU-BUILDER/E
xamples_1.2.0[redacted]/SimpleExample/src"
[15:44:39] 4 of 4: Running scripts triggered through "after_import"
                   - No script defined in the vEcuConf.yaml file
*** SUCCESS ***
Please answer the question!
```

Question

Do you want to proceed building the sources of vECU or inspect the results of the import?

Yes:	Build the sources of vECU.
No:	Inspect the results of the import.
Cancel:	Stop the process here.

Fig. 4-7: Proceed with vECU Build dialog or inspect the results (Windows)



[redacted] /opt/ETAS/VECU-BUILDER/1.2.0...

```
### Importing files and folders                                      ###
######################################################################
[09:44:11] 1 of 4: Reading config: vEcuConf.yaml
[09:44:11] 2 of 4: Running scripts triggered through "before_import"
                   - No script defined in the vEcuConf.yaml file
[09:44:11] 3 of 4: Importing files and folders to "vECU/imported"
                   - Copying contents of "/opt/ETAS/VECU-BUILDER/Examples_1.2.0-
[redacted]/SimpleExample/src"
[09:44:11] 4 of 4: Running scripts triggered through "after_import"
                   - No script defined in the vEcuConf.yaml file
*** SUCCESS ***
Please answer the question!
```

Question

Do you want to proceed building the sources of vECU or inspect the results of the import?

Yes:	Build the sources of vECU.
No:	Inspect the results of the import.
Cancel:	Stop the process here.

Fig. 4-8: Proceed with vECU Build dialog or inspect the results (Ubuntu 20.04 LTS)

1. Click **Yes**.

➩ Your new workspace is now created.

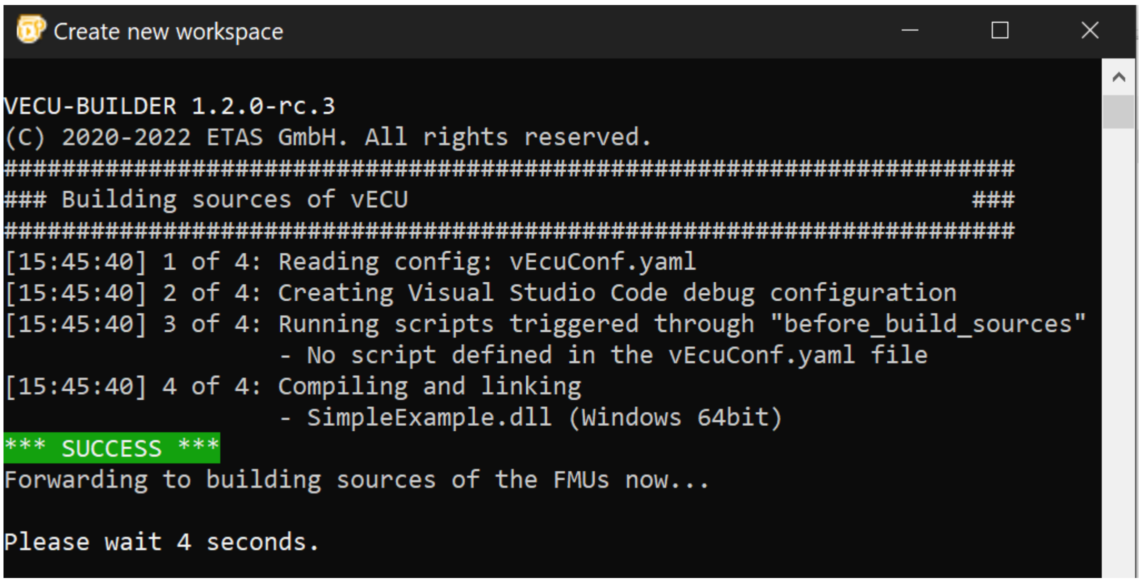
The process will continue with the next stage.

4.3 Building the vECU

During this stage, the sources imported into your workspace are compiled and linked into a DLL/SO file forming the core functionality of your future vECU.

After successful completion of the previous stage Importing Files and Folders and selecting to proceed with the build of the vECU you were forwarded to the next stage Building the FMU and the process continues.

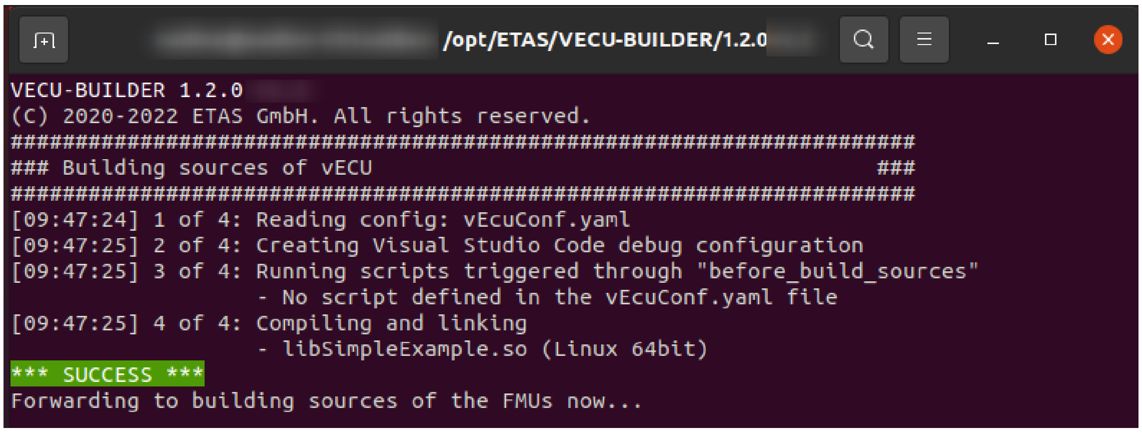
If you work in an already existing workspace, you can trigger this stage by running `2_Build.bat` on **Windows** or `2_Build.sh` on **Ubuntu 20.04 LTS**.



```
VECU-BUILDER 1.2.0-rc.3
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Building sources of vECU                                       ###
######################################################################
[15:45:40] 1 of 4: Reading config: vEcuConf.yaml
[15:45:40] 2 of 4: Creating Visual Studio Code debug configuration
[15:45:40] 3 of 4: Running scripts triggered through "before_build_sources"
                   - No script defined in the vEcuConf.yaml file
[15:45:40] 4 of 4: Compiling and linking
                   - SimpleExample.dll (Windows 64bit)
*** SUCCESS ***
Forwarding to building sources of the FMUs now...

Please wait 4 seconds.
```

Fig. 4-9: Building vECU completed (Windows)



```
VECU-BUILDER 1.2.0 [redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Building sources of vECU                                       ###
######################################################################
[09:47:24] 1 of 4: Reading config: vEcuConf.yaml
[09:47:25] 2 of 4: Creating Visual Studio Code debug configuration
[09:47:25] 3 of 4: Running scripts triggered through "before_build_sources"
                   - No script defined in the vEcuConf.yaml file
[09:47:25] 4 of 4: Compiling and linking
                   - libSimpleExample.so (Linux 64bit)
*** SUCCESS ***
Forwarding to building sources of the FMUs now...
```

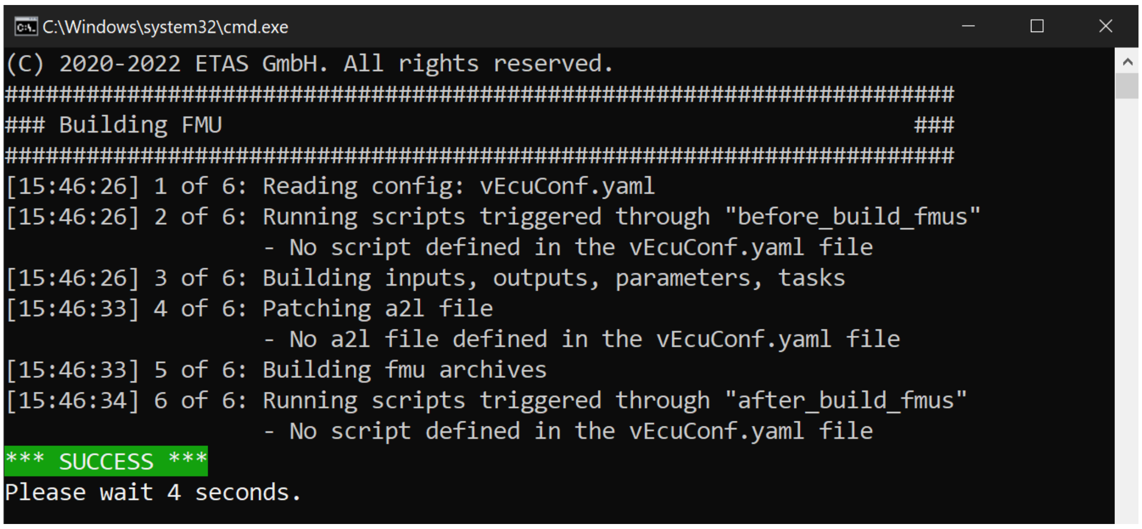
Fig. 4-10: Building vECU completed (Ubuntu 20.04 LTS)

The process will automatically continue with the next stage.

4.4 Building the FMU

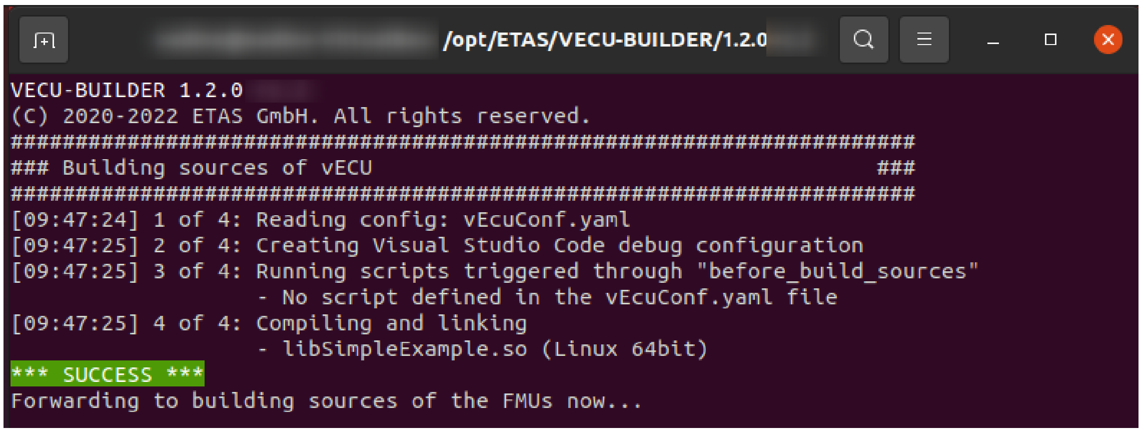
During this stage, the DLL/SO file created in the previous stage will be wrapped into an FMU container representing your vECU.

After successful completion of the previous stage Building the vECU and selecting to proceed with the build of the vECU you were forwarded to the next stage Building the FMU where the process completes.



```
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Building FMU                                                   ###
######################################################################
[15:46:26] 1 of 6: Reading config: vEcuConf.yaml
[15:46:26] 2 of 6: Running scripts triggered through "before_build_fmus"
                   - No script defined in the vEcuConf.yaml file
[15:46:26] 3 of 6: Building inputs, outputs, parameters, tasks
[15:46:33] 4 of 6: Patching a2l file
                   - No a2l file defined in the vEcuConf.yaml file
[15:46:33] 5 of 6: Building fmu archives
[15:46:34] 6 of 6: Running scripts triggered through "after_build_fmus"
                   - No script defined in the vEcuConf.yaml file
*** SUCCESS ***
Please wait 4 seconds.
```

Fig. 4-11: Building FMU completed (Windows)

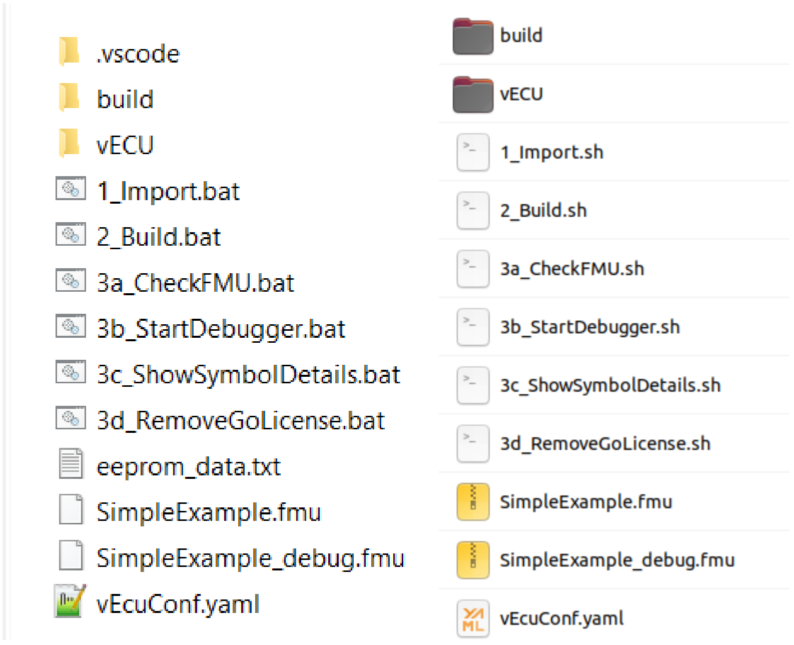


```
VECU-BUILDER 1.2.0 [redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Building sources of vECU                                       ###
######################################################################
[09:47:24] 1 of 4: Reading config: vEcuConf.yaml
[09:47:25] 2 of 4: Creating Visual Studio Code debug configuration
[09:47:25] 3 of 4: Running scripts triggered through "before_build_sources"
                   - No script defined in the vEcuConf.yaml file
[09:47:25] 4 of 4: Compiling and linking
                   - libSimpleExample.so (Linux 64bit)
*** SUCCESS ***
Forwarding to building sources of the FMUs now...
```

Fig. 4-12: Building FMU completed (Ubuntu 20.04 LTS)

4.5 Workspace Content

You have now successfully created the VECU-BUILDER workspace and built your first vECU based on the provided Simple Example sources. In this chapter, you find a description of the workspace contents for Windows and Ubuntu 20.04 LTS.



.vscode
build
vECU
1_Import.bat
2_Build.bat
3a_CheckFMU.bat
3b_StartDebugger.bat
3c_ShowSymbolDetails.bat
3d_RemoveGoLicense.bat
eeprom_data.txt
SimpleExample.fmu
SimpleExample_debug.fmu
vEcuConf.yaml

build
vECU
1_Import.sh
2_Build.sh
3a_CheckFMU.sh
3b_StartDebugger.sh
3c_ShowSymbolDetails.sh
3d_RemoveGoLicense.sh
SimpleExample.fmu
SimpleExample_debug.fmu
vEcuConf.yaml

Fig. 4-13: Workspace contents (left: Windows, right: Ubuntu 20.04 LTS)

The content of the workspace consists of several artefacts:

– `vscode` folder:
  - `launch.json` file for vECU debugging in VS Code

  Note

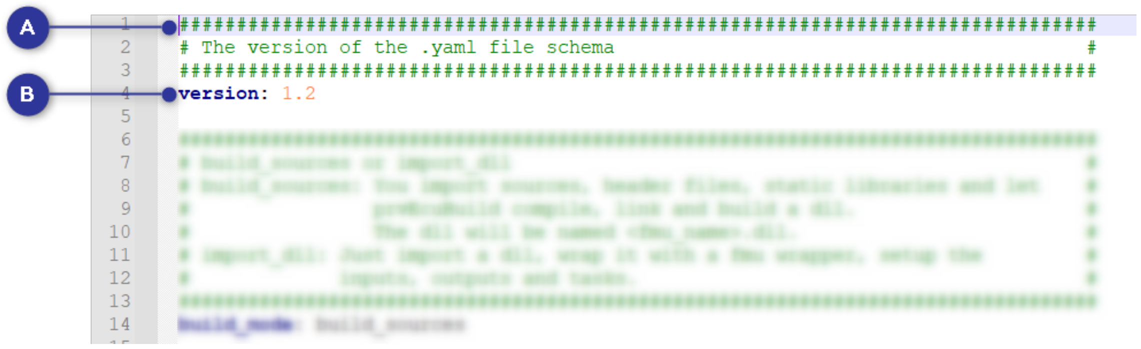
  The `.vscode` folder is only included in Windows.

– `build` folder:
  - `additional_scripts` folder: location for your project specific additional scripts
  - `log` folder:
    log files from executed stages
  - `scripts` folder: command and shell scripts to perform the individual stages
  - `last_build_footprint.txt`: details of last performed build stage
  - `RawSymbolDetails.txt`: subset of `SymbolDetails` and for internal purposes only
  - `SymbolDetails.txt`: symbols within your sources and their attributes

- `vECU` folder:
  - `buildArtifacts` folder: Library file and its associated debug information
  - `CMake` folder: CMake project artifacts
  - `imported` folder: all imported artifacts
  - `CMakeLists.txt`: set of directives and instructions for building your sources
- `1_Import.bat`/`1_Import.sh`
  file to trigger the Importing Files and Folders stage.
- `2_Build.bat`/`2_Build.sh`
  file to trigger the Building the vECU stage.
- `3a_CheckFMU.bat`/`3a_CheckFMU.sh`
  file to invoke the FMU Checker and inspect the vECU outputs.
- `3b_StartDebugger.bat`/`3b_StartDebugger.sh`
  file to invoke MSVC or VS Code as debugger.
- `3c_ShowSymbolDetails.bat`/`3c_ShowSymbolDetails.sh`
  file to invoke Notepad++ (Windows) / new Terminal (Ubuntu 20.04 LTS) and display the Symbol Details.
- `3d_RemoveGoLicense.bat`/`3d_RemoveGoLicense.sh`
  file to remove the GO license from the vECU (only relevant if vECU was built with GO-license).
- `SimpleExample.fmu`
  release version of your vECU, for more details see Simple Example.
- `SimpleExample_debug.fmu`
  debug version of your vECU, for more details see Simple Example.
- `vEcuConf.yaml`
  the YAML configuration file, for more details see Configuration.

4.6 Configuration

The YAML file contains the configurations for the import and build process as well as for the vECU itself. It is the only configuration you need to create and maintain. The YAML file is divided into several sections, each section configuring a particular attribute. You are guided through the YAML file with comments on each section and configuration attributes. Every section is structured in a standardized way:



```
################################################################################
# The version of the .yaml file schema                                         #
################################################################################
version: 1.2

################################################################################
# build_sources or import_dll                                                  #
# build_sources: You import sources, header files, static libraries and let    #
#                preEcuBuild compile, link and build a dll.                    #
#                The dll will be named <fmu_name>.dll.                         #
# import_dll: Just import a dll, wrap it with a fmu wrapper, setup the         #
#             inputs, outputs and tasks.                                       #
################################################################################
build_mode: build_sources
```

`A`: comment with information on the corresponding section

`B`: configuration attributes and values

The following is a list of all attributes available in the YAML file:

- `version`

  This is the version of the used YAML file schema and must not be changed.
- `build_mode`

  You can select between 2 modes:

  `build_sources`: You import source code (either as AUTOSAR Classic compliant or legacy C-code), header files, and static libraries. VECU-BUILDER then builds your vECU in the form of an FMU container.

  The vECU will be named `<fmu_name>.fmu`.

  `import_dll`: You import an existing, already compiled and linked, software in the form of a DLL/SO containing the functionality of your vECU.

  VECU-BUILDER then wraps it in an FMU container, sets up the inputs, outputs and tasks, patches the a2l file, sets up the xcp slave port, etc.
- `fmu_name`

  Enter the name of you vECU.

  The code of your vECU is located inside the FMU in the folder" `resources/<fmu_name>.dll`".

  This and other DLL/SO files are loaded and executed by the FMU runner.

- `import_into_project`

  Enter the paths to the files and folders to be imported.

  You can specify paths to folders and/or individual files such as `*.c`, `*.h`, `*.cpp`, `*.hpp` or `*.zip` archives which will be extracted during import.

  The import target is the "`vEcu/imported`" folder in your workspace.

  Environment variables can be used like this:

  '`${VECUBUILDER_EXAMPLES}\SimpleExample\src`'

- `additional_resources`

  Additional resources can be used to resolve dependencies by making `.dll`/`.so` libraries your application depends on part of the build and execution process.

  Specify all additional resources that are to be included in the FMU, i.e. '`${VECUBUILD-ER_WORKSPACE}\vECU\imported\additional_DLLs\myDependentLibrary.dll`' and they will be copied to the resources folder of the FMU during the Building FMU stage.

- `import_external_vecu_dll`

  Only needed if you selected `import_dll` as `build_mode`.

  That DLL/SO already contains the code of your vECU, you can skip the compiling and linking and just import your DLL/SO into the FMU wrapper.

  Here you enter the DLL/SO name and the path for updates:

  `dll_name`: The name of the DLL/SO. The DLL/SO must contain private symbol information.

  `get_updates_from`: If VECU-BUILDER can find a DLL/SO including private symbol information, the imported DLL/SO will be updated.

  Environment variables can be used like this:'`${SystemDrive}\Sandbox`'.

- `architecture`

  Specify the architecture.

  When importing sources, the setting of this attribute has to match the integration and simulation system where the vECU is to be used.

  In case you are importing an DLL/SO precompiled for either 32bit or 64bit architecture, this attribute must be set to the same.

- `xcp_slave`

  Enter the port and IP address of the XCP Slave to be setup in your vECU.

  These values are transferred to the patched A2L file. The used protocol is TCP. For more details, see A2L File Patching.

  > **Note**
  >
  > A socket (IP address + port + protocol) for the XCP connection between INCA and XCP slave can only be used once. If a port is busy, you must define another port in the YAML file.

  > **Note**
  >
  > xcp_slave is supported for Windows only.

- `operating_system`

  Enter the operating system. Currently only Windows and Ubuntu 20.04 LTS supported.

- `build_tool`

  Enter your preferred build tool. `Build_tool` differs between Windows and Ubuntu 20.04 LTS.

  Windows:

  Several MSVC versions and MinGW Makefiles are supported.

  In case Visual Studio is selected, a Visual Studio Solution is generated.

  If you choose MinGW Makefiles, a CMake project is generated.

  These artefacts are stored in the “`vECU\CMake`” folder in your workspace.

- `path_to_mingw`: If the user-specific MinGW is defined, CMake builds the sources using this MinGW version.

  > **Note**
  >
  > `path_to_mingw` is supported for Windows only.

  Ubuntu 20.04 LTS:

  You can chose Unix Makefiles and a make file for use with GNU compiler is generated.

- `cmake_generator_toolset`

  Define which toolset should be used by CMake during the build process.

  For more details, see CMAKE_GENERATOR_TOOLSET.

- `inputs, outputs, parameters, locals`

  Enter the variables you wish to expose as ports of your FMU.

  Inputs, outputs, parameters, and locals refer to the causality of the FMI.

  Wildcards of `*` and `?` are allowed. Arrays can be added using `myArray*`, the same goes for structures. If your wildcard expression breaks the YAML compatibility, put it in single apostrophes.

  Example: '`*a`' finds all symbols ending with an 'a'.

  Aliases can be defined for variables, which results in renaming of FMI ports. The aliases are used in the `modelDescription.xml` and the original variable names are used in the `resources.txt`.

  Note

  Variables of type enumeration will be interpreted as integers in the `modelDescription.xml` file of the FMU.

  The name-value mapping of enumerations will be ignored when enumerations are used as interfaces. Only the integer value will be exchanged.

  Note

  The use of bitfields for inputs, outputs, parameters and locals is not supported.

- `initial_data`

  Enter the path for source and target destination to define the initial values of calibration variables.

  The initial data is virtually flashed into memory during initialization. The data file in the FMU (defined by destination) is read and its values are written to RAM. This simulates a part of the NVRAM (non-volatile RAM).

  `source`: Where to get the file. During build-time this file will be copied from source.

  `destination`: Where to store the file inside the FMU, relative to the resources folder of the FMU (optional). This file is used during run-time.

  Supported formats:

  `.VarVal`: list of pairs separated by one space, where the lhs refers to the C variable and the rhs to the value.

  `.dcm` (only experimental support)

  For more information, see `InitialData Functionality`.

- `eeprom`

  Specify the eeprom simulation attributes.

  The eeprom data is loaded from a file to RAM during vECU initialization. The data is saved to the file before running terminate tasks and when unloading the vECU. This can be used to simulate a soft reset behavior where EEPROM stored data are preserved and not lost once the simulation of vECU terminates. A typical application of this feature is the storage of total mileage information in the ESP controller.

  `source`: Path where to get the file. This is used during the build.

  `destination`: Path where to store the file relative to the resources folder of the FMU. This is the working copy (optional).

  `sync`: This can be a UNC Path or a regular pathname. When the vECU is initializing, this file is copied to the 'destination', if it exists. When the vECU terminates, the updated file in 'destination' is copied to the 'sync' location (optional). To setup the UNC Path, see `Windows Cannot Access Localhost While Using Sync Attribute in EEPROM`.

  `c_variables`: The C variable names that store the eeprom data.

  Supported format:

  `.txt`: A line starting with '`#`' is a comment. All other lines store the data stream to be flashed to the C variables. The order of the data stream lines is the same as the order of the c_variables listed.

  A data stream is a sequence of bytes in hex format. Each byte is separated by a space. E.g.: 01 02 ee 4f. In the default YAML file the sync is commented out.

  To get more information about eeprom, see `eeprom Functionality`.

- `tasks`

  Define the tasks that are to be executed and their attributes.

  To simulate the microcontroller behavior with its periodically executed functions of your software, these functions are to be defined as tasks in this section.

  A function can be defined as a task only once, duplicated functions will be ignored.

  `function_name`: '<function name>', without brackets, set in apostrophes, no arguments allowed.

  `trigger`: Choose between cyclic, initial or terminate, the default is cyclic.

  `period:` <number> [in seconds], the default is 1.0.

  `first_call`: <number> [in seconds] for the cyclic tasks, the default is period.

  `priority`: The lower the number the higher the priority, the default is 0.

  `max_calls`: <number>, -1 means infinite, 0 means no call.

- `redirect_function_calls`

  Enter the names functions to be replaced and their substitutes.

  The function signatures of the two functions must be identical. This allows you to test the behavior of your software using alternative implementation without changing the original source code or to replace unfinished or hardware-dependent functions with mock functions.

  `replaced_function`: Enter the function name of the function to be replaced.

  `substitute_function`: The function name of the function that substitutes the replaced function.
- `build_include_filters & build_exclude_filters`

  Only usable if you selected `build_sources` as `build_mode`.

  You can select files and/or folders that should be included or excluded in/-from the vECU build process.

  Files are only included into the build if they are matched by at least one `build_include_filter` and are not matched by any `build_exclude_filter`.
- `assembly_list_files`

  Specify your assembly list files for the build process.

  Of the given sources defined by "`build_include_filters`" and "`build_exclude_filters`", only those listed in a file are passed to the compiler.

  If no assembly list files are configured, all sources are compiled.
- `additional_include_directories & additional_defines`

  Only usable if you selected `build_sources` as `build_mode`.
- These values are passed to the preprocessor. This is useful if you need to set/unset some defines to adapt them to the new PC target.

  Brackets '(', ')' must be escaped as '\(', '\)'.

Note

The following limitations apply to filename paths, command line and response-file lengths in the Windows environment.

- Filename paths cannot be longer than MAX_PATH (260) characters.
- Command-line lengths cannot be longer than 32,768 characters.
- Response-file lengths cannot be longer than 131,072 characters.

- `additional_compile_flags`

  Only usable if you selected `build_sources` as `build_mode`.

  Specify how the compiler should work. Each individual flag must be written in a separate line and put in single apostrophes, i.e. '`/ZI`'.

  The flags are written into the CMakeLists.

  For more details, see MSVC compiler options or gcc compiler options.
- `additional_static_libraries`

  Only usable if you selected `build_sources` for `build_mode`.

  The libraries need to be located in the folder "`./projects/vEcu/imported`".
- `environment_variables`

  You can define process-level environment variables that are set by the build process and by the FMI wrapper during the vECU execution.

  Example: `PATH=c:\Temp;${PATH}`

  These variables can be configured and modified in one location and can be accessed from scripts and configuration files. Process-level environment variable of `VECUBUILDER_WORKSPACE` is created automatically during the build process with its value pointing to the current workspace.
- `additional_scripts`

  Define your additional scripts for execution.

  Project-specific scripts can be configured to be executed at various phases of the import and/or the build process.

  You can utilize these to copy or modify files, add files to the FMU archive, parse files, etc. You may use Python, Perl, .cmd scripts, .bat batch/shell script files as long as these can be executed on your machine.

  `filename`: Your script name (default location for such scripts is "`build\additional_scripts`" in your workspace) or full absolute path.

  `arguments`: Optionally, you may define arguments to be passed to the respective interpreter.

  `command_line`: Full absolute path to the interpreter.

  `trigger`: Select when should your script be executed from these options:

  - `before_import`
  - `after_import`
  - `before_build_sources` (deprecated: `before_build`)
  - `before_build_fmus`
  - `after_build_fmus` (deprecated: `after_build`)

  `priority`: Define with which priority should your script be executed.

- `patch_a2l_file`

  `filename`: Enter the name of your A2L file to be patched.

  An A2L File is required to connect an MCD tool such as INCA to the running vECU. The A2L file needs to be located in the folder: "`vEcu/imported`". By using `a2l_name_mappings` in the `.yaml` file, symbol names can be mapped to names in the A2L File.

  Changes are done by using regular expressions for search and substitute. One such regular expression allows to map multiple names at once. To see an example, see the following table.

RegEx	(array)\[(\d+)\]	->	\1_\2
Mapping	array[1]	->	array_1

  VECU-BUILDER will update the memory addresses of all measurements and characteristics in the provided A2L file. The original A2L file is renamed by appending `.bak` to its name. For more details, see `A2L File Patching` and `A2L Name Mapping`.

- `debug_hook`

  Specify whether to enable or disable a debug hook. When enabled, the FMU execution is interrupted when the FMU is instantiated until a debugger is attached. For more details, see `Debugging vECU`.

- `additional_link_flags`

  Only usable if you selected build_sources as build_mode.

  Specify how the linker should work. Each individual flag must be written in a separate line and put in single apostrophes, i.e. '`/DEBUG`'.

  The flags are written into the `CMakeLists.txt`.

  For more details, see MSVC linker options or gcc linker options.

– `simple_file_modifications`

  Specify file modifications that shall be applied to files imported in "`vECU/imported`" folder.

  In case you specify multiple modifications, they will be applied sequentially following the order in which they were specified.

  The next two attributes are mandatory for all types of modifications.

  `file_regex`: Specify the search RegEx for a file or a set of files that shall be modified.

  `trigger`: Specify when the modification shall be applied from the 2 below options:

  - `after_import` (default)
  - `before_build_sources`

  You can specify a single or multiple actions (modification types) from the 4 below options:

  - `comment_line`: Comment out a single line of code by adding '`//`' at the beginning of the line.
  - `search_and_replace`: Replace a line of code that matches the `Search_regex` with the `replacement`.
  - `insert_code_above`: Insert code above a matched line.
  - `insert_code_belo`: Insert code below a matched line.

  You must `specify line_regex` and to which match(es) the modification are to be applied to (`apply_to`) for each action from the below 3 options:

  - `all_matches` (default)
  - `last_match`
  - `first_match`

  For `insert_code_above` and `insert_code_below`, you must specify the code section that is to be inserted.

  When using `simple_file_modifications`, consider the following procedure to make sure, modifications are not included in `.bak` file.

  1. Get the set of files and apply the file filter.

  2. Revert backups for all files to be modified: Move the .bak files to overwrite the normal filename.

  → The backup file is deleted.

  3. Create the backup on all files that need to be modified, excluding files ending with `.bak`.

  4. Apply the file modifications to all files that need to be modified.

  Note

  If you need more sophisticated file modifications, use a project-specific script via the `additional_scripts`.

5 Exploring the Examples

This chapter contains details on the two examples that are delivered along with the tool and provides pointers on how to experiment within their respective workspaces.

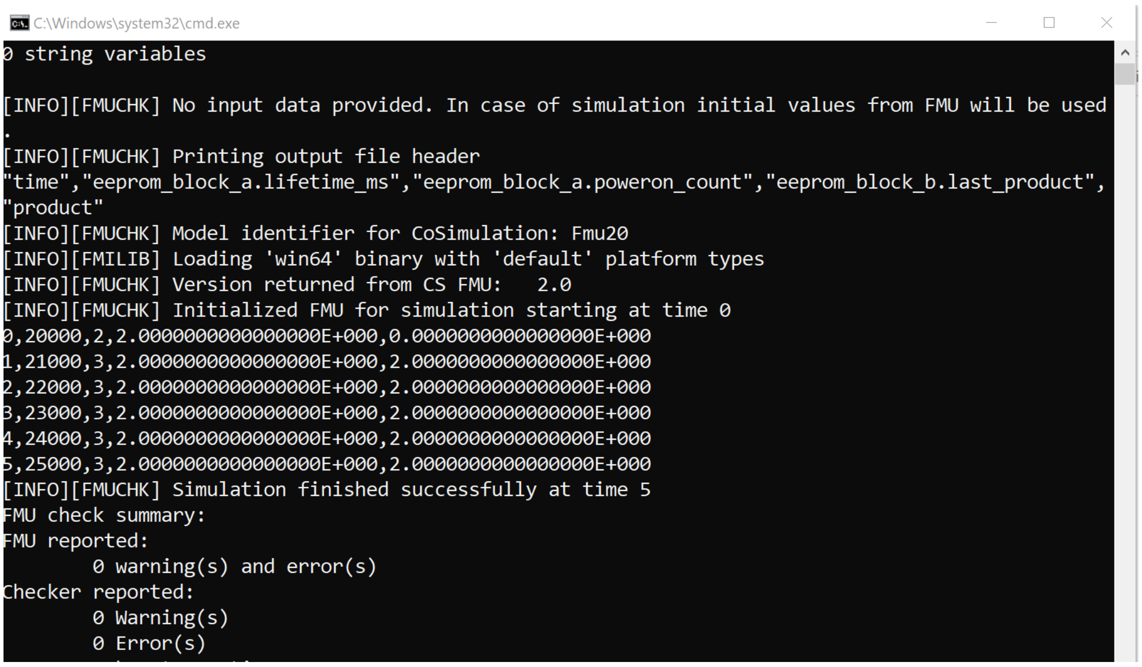
5.1 Simple Example

If you followed the instructions in the chapter Working with VECU-BUILDER, you now have a workspace on your PC which is based on the Simple Example.

5.1.1 FMU Checker

To conduct a quick smoke test of the created vECU, the FMU check tool is delivered along with VECU-BUILDER. This tool can be invoked via the `3a_CheckFMU.bat` on **Windows** or `3a_CheckFMU.sh` on **Ubuntu 20.04 LTS**. Simply execute this file to run the release vECU or drag-and-drop the debug vECU into this batch/shell script file to run the debug vECU.

This tool opens a terminal where details of the FMU are displayed and the time and values of defined outputs are printed. The batch/shell script file is configured so that the simulation runs for 10 seconds. You can change this by altering the batch/shell script file.



```
0 string variables

[INFO][FMUCHK] No input data provided. In case of simulation initial values from FMU will be used
.
[INFO][FMUCHK] Printing output file header
"time","eeprom_block_a.lifetime_ms","eeprom_block_a.poweron_count","eeprom_block_b.last_product",
"product"
[INFO][FMUCHK] Model identifier for CoSimulation: Fmu20
[INFO][FMILIB] Loading 'win64' binary with 'default' platform types
[INFO][FMUCHK] Version returned from CS FMU:   2.0
[INFO][FMUCHK] Initialized FMU for simulation starting at time 0
0,20000,2,2.0000000000000000E+000,0.0000000000000000E+000
1,21000,3,2.0000000000000000E+000,2.0000000000000000E+000
2,22000,3,2.0000000000000000E+000,2.0000000000000000E+000
3,23000,3,2.0000000000000000E+000,2.0000000000000000E+000
4,24000,3,2.0000000000000000E+000,2.0000000000000000E+000
5,25000,3,2.0000000000000000E+000,2.0000000000000000E+000
[INFO][FMUCHK] Simulation finished successfully at time 5
FMU check summary:
FMU reported:
        0 warning(s) and error(s)
Checker reported:
        0 Warning(s)
        0 Error(s)
```

Fig. 5-1: FMU Checker output (Windows)

5.1.2 Difference Between Debug and Release vECUs

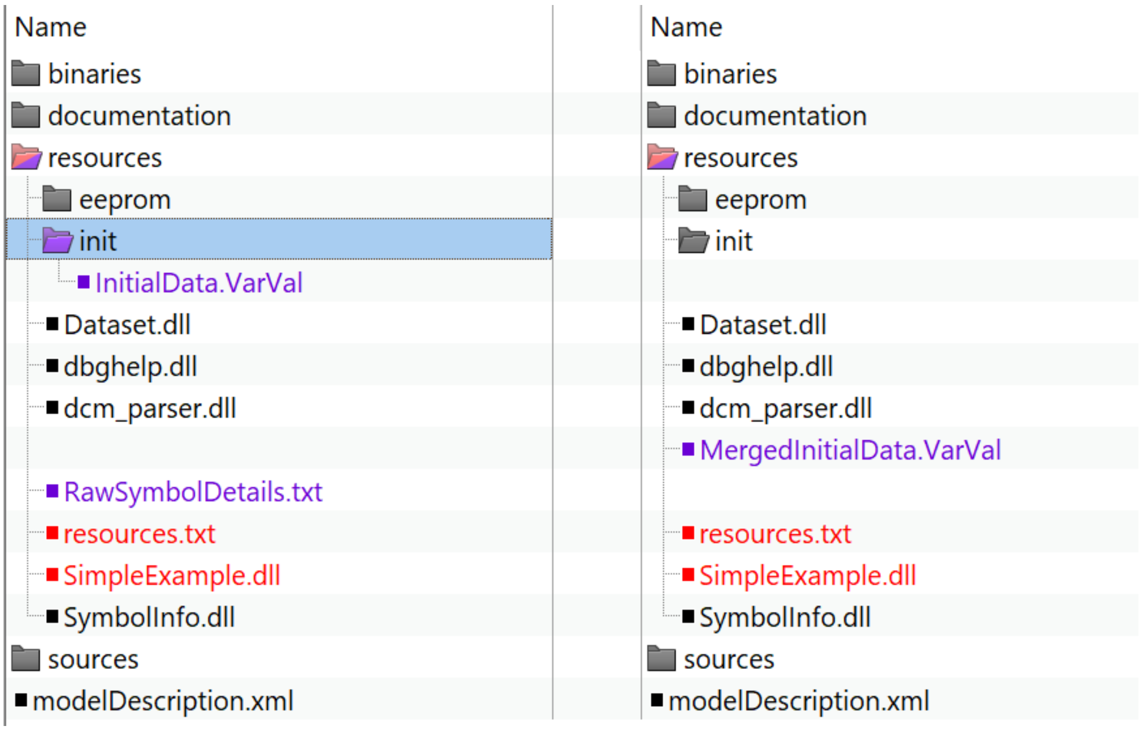
You find two FMUs in this workspace, one named `SimpleExample.fmu` (which will be referred to as 'release vECU') and the other one named `SimpleExample_debug.fmu` (which will be referred to as 'debug vECU').

Extract each of these two FMU archives into its own folder and let's explore what they contain and how they differ.

The functional behavior of both vECUs is identical.

The debug vECU contains symbol information and additional artefacts, e.g., PDB (when build tool is MSVC) or DIE (when build tool is MinGW). Use the debug vECU to debug and step through your code.

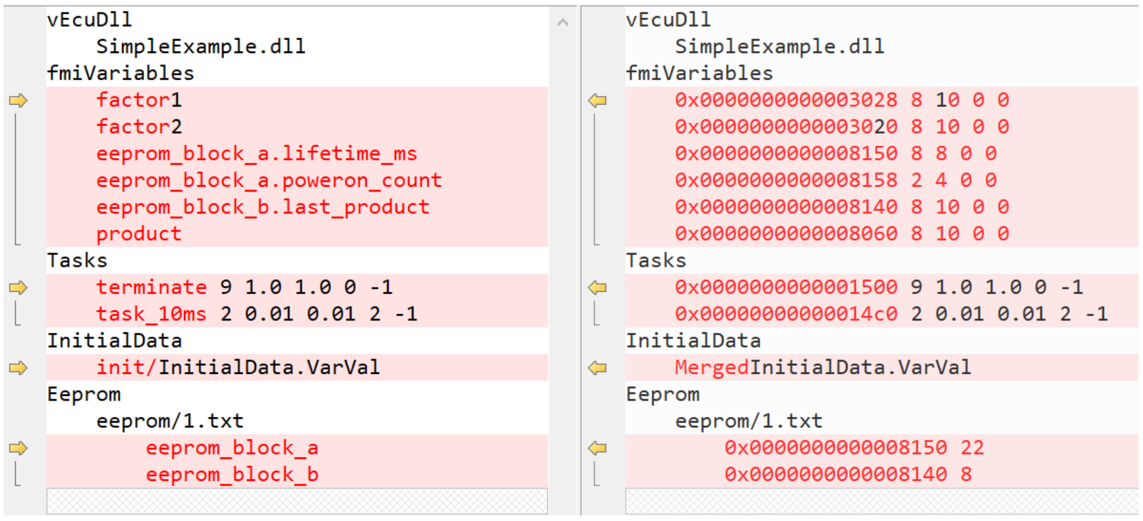
When you compare the two extracted folders, you will notice that the main difference is in the resources folder.



Name	Name
binaries	binaries
documentation	documentation
resources	resources
eeprom	eeprom
init	init
InitialData.VarVal	
Dataset.dll	Dataset.dll
dbghelp.dll	dbghelp.dll
dcm_parser.dll	dcm_parser.dll
	MergedInitialData.VarVal
RawSymbolDetails.txt	
resources.txt	resources.txt
SimpleExample.dll	SimpleExample.dll
SymbolInfo.dll	SymbolInfo.dll
sources	sources
modelDescription.xml	modelDescription.xml

Fig. 5-2: Comparison of debug and release vECU

The release vECU contains only address information, unlike the debug vECU which contains the variables and function names. The release vECU protects the IP contained in the vECU and does not contain symbol information. Use the release vECU if you want to share it with others.



```
vEcuDll                                          vEcuDll
    SimpleExample.dll                                SimpleExample.dll
fmiVariables                                     fmiVariables
    factor1                                          0x0000000000003028 8 10 0 0
    factor2                                          0x0000000000003020 8 10 0 0
    eeprom_block_a.lifetime_ms                       0x0000000000008150 8 8 0 0
    eeprom_block_a.poweron_count                     0x0000000000008158 2 4 0 0
    eeprom_block_b.last_product                      0x0000000000008140 8 10 0 0
    product                                          0x0000000000008060 8 10 0 0
Tasks                                            Tasks
    terminate 9 1.0 1.0 0 -1                         0x0000000000001500 9 1.0 1.0 0 -1
    task_10ms 2 0.01 0.01 2 -1                       0x00000000000014c0 2 0.01 0.01 2 -1
InitialData                                      InitialData
    init/InitialData.VarVal                          MergedInitialData.VarVal
Eeprom                                           Eeprom
    eeprom/1.txt                                     eeprom/1.txt
        eeprom_block_a                                   0x0000000000008150 22
        eeprom_block_b                                   0x0000000000008140 8
```

Fig. 5-3: Comparison of resources.txt

5.1.3 InitialData Functionality

When using `intialData` in the `.yaml` file, the `InitialData.VarVal` will be copied from `Examples/SimpleExample/init` into the FMU that is going to be created. You can also use `intialData.dcm`, it is the same procedure.

`InitialData.VarVal` and `intialData.dcm` includes initial values. To see the output of the two initial values using InitialData.VarVal, see `5.1.1`.

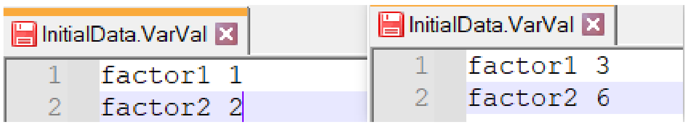
You can change the initial values inside the `InitialData.VarVal` or `intialData.dcm` to the desired values. It is possible to use both files simultaneously.

Note

If `InitialData.VarVal` and `intialData.dcm` are used simultaneously, a variable can only occur in one of the two files, E.g. `factor1` in `.VarVal` and `factor2` in `.dcm`. Otherwise, the variables will overwrite each other.

To change InitialData and build new FMU using InitialData.VarVal

1. Open the `.InitialData.VarVal` in `Examples/SimpleExample/init`.
2. Change the values to the desired values.
3. Save the changes.



```
factor1 1
factor2 2
```

```
factor1 3
factor2 6
```

Fig. 5-4: Initial values (left) and changed values (right) inside InitialData.VarVal. (Windows)

4. Execute `2_Build.bat` in the workspace on **Windows**

Or

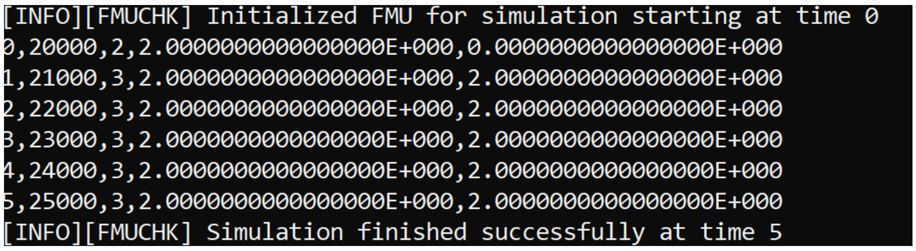
`Execute 2_Build.sh` in the workspace on **Ubuntu 20.04 LTS**.

⇨ The FMU is generated (`SimpleExample.fmu` and `SimpleExample_debug.fmu`).

5. To see the changed output in the FMU, `execute 3a_CheckFMU.bat` on **Windows**

Or

execute `3a_CheckFMU.sh` on **Ubuntu 20.04 LTS**.

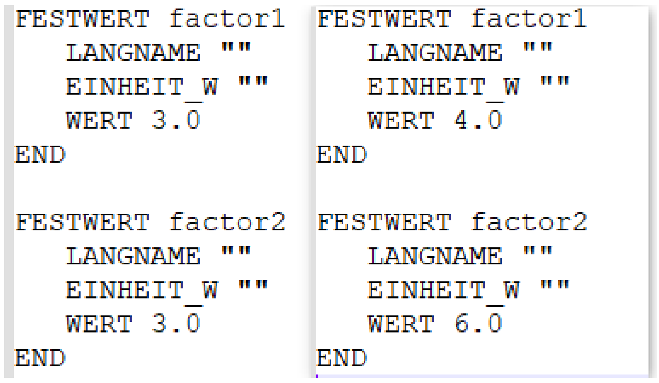


```
[INFO][FMUCHK] Initialized FMU for simulation starting at time 0
0,20000,2,2.0000000000000000E+000,0.0000000000000000E+000
1,21000,3,2.0000000000000000E+000,2.0000000000000000E+000
2,22000,3,2.0000000000000000E+000,2.0000000000000000E+000
3,23000,3,2.0000000000000000E+000,2.0000000000000000E+000
4,24000,3,2.0000000000000000E+000,2.0000000000000000E+000
5,25000,3,2.0000000000000000E+000,2.0000000000000000E+000
[INFO][FMUCHK] Simulation finished successfully at time 5
```

Fig. 5-5: FMU Checker output with new values using InitialData.VarVal. (Windows)

To change InitialData and build new FMU using intialData.dcm

1. Open the `.InitialData.dcm` in `Examples/SimpleExample/init`.
2. Change the values to the desired values.
3. Save the changes.



```
FESTWERT factor1
   LANGNAME ""
   EINHEIT_W ""
   WERT 3.0
END

FESTWERT factor2
   LANGNAME ""
   EINHEIT_W ""
   WERT 3.0
END
```

```
FESTWERT factor1
   LANGNAME ""
   EINHEIT_W ""
   WERT 4.0
END

FESTWERT factor2
   LANGNAME ""
   EINHEIT_W ""
   WERT 6.0
END
```

Fig. 5-6: Initial values (left) and changed values (right) inside InitialData.dcm (Windows)

4. Execute `2_Build.bat` in the workspace on **Windows**

Or

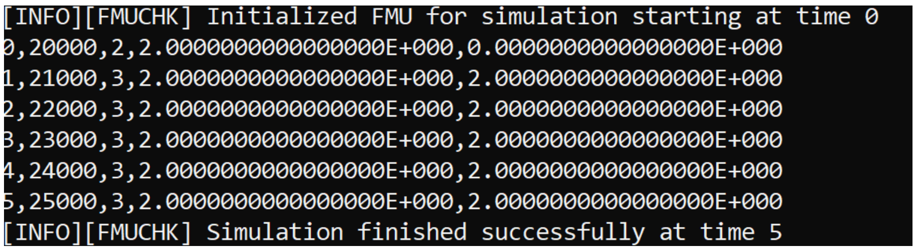
`Execute 2_Build.sh` in the workspace on **Ubuntu 20.04 LTS**.

⇨ The FMU is generated (`SimpleExample.fmu` and `SimpleExample_debug.fmu`).

5. To see the changed output in the FMU, `execute 3a_CheckFMU.bat` on **Windows**

Or

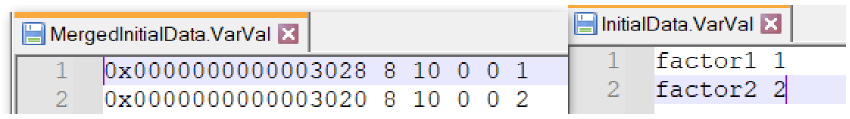
execute `3a_CheckFMU.sh` on **Ubuntu 20.04 LTS**.



```
[INFO][FMUCHK] Initialized FMU for simulation starting at time 0
0,20000,2,2.0000000000000000E+000,0.0000000000000000E+000
1,21000,3,2.0000000000000000E+000,2.0000000000000000E+000
2,22000,3,2.0000000000000000E+000,2.0000000000000000E+000
3,23000,3,2.0000000000000000E+000,2.0000000000000000E+000
4,24000,3,2.0000000000000000E+000,2.0000000000000000E+000
5,25000,3,2.0000000000000000E+000,2.0000000000000000E+000
[INFO][FMUCHK] Simulation finished successfully at time 5
```

Fig. 5-7: FMU Checker output with new values using InitialData.dcm. (Windows)

The initial data (`InitialData`) is processed differently. While in release vECU Initial Data is merged in another `.VarVal` file (`MergedInitialData.VarVal`), this is not the case in the debug vECU. Here the InitialData file is stored in the `init` folder. To get the different folder structures, see 5.1.3.



```
0x0000000000003028 8 10 0 0 1
0x0000000000003020 8 10 0 0 2
```

```
factor1 1
factor2 2
```

Fig. 5-8: Comparison of MergedInitialData.VarVal (release vECU) and InitialData.VarVal (debug vECU)

5.1.4 eeprom Functionality

When using `eeprom` in the `.yaml` file, `eeprom_data.txt` will be copied from `VECUBUILDER_EXAMPLES\SimpleExample\src` into the workspace to `vECU/imported` during the build. The `eeprom_data.txt` will be set as default sync txt file in workspace.

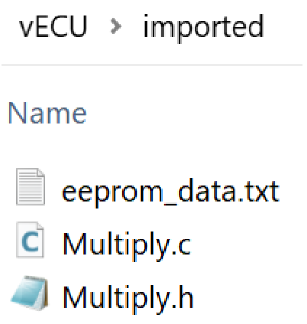


Fig. 5-9: eeprom_data.txt in the workspace

The `.txt` contains the data stream that should be used for the `c_variables`. The order of the data stream to be flashed to the `c_variables` in the `.txt` needs to be the same as in the `.yaml` file. To see the correct order for `c_variables` and the data streams for `c_variables` in the `.yaml` file and in the `.txt` file, see Fig. 5-10 and Fig. 5-11

Supported c_variables for SimpleExample are:

- `eeprom_block_a`: Shows the lifetime of the vECU in ms and counts, how often vECU was powered on.
- `eeprom_block_b`: Shows the last value of product calculated in the previous execution.

```
eeprom:
- source: '${VECUBUILDER_WORKSPACE}\vECU\imported\eeprom_data.txt'
  destination: 'eeprom\1.txt'
#  sync: '\\localhost\c$\TEMP\eeprom_data.txt'
#  sync: 'C:\TEMP\eeprom_data.txt'
  c_variables:
  - eeprom_block_a
  - eeprom_block_b
```

Fig. 5-10: Correct order for c_variables in yaml file

```
# This is a comment
# eeprom_block_a
20 4e 00 00 00 00 00 00 02 00
# eeprom_block_b
00 00 00 00 00 00 00 40
```

Fig. 5-11: Data streams for c_variables in txt file

When using `eeprom` in the .yaml file, the `eeprom_data.txt` will be copied into the FMU that is going to be created.

The path location in the FMU is: `\resources\eeprom\1.txt`

It is also possible, to have another txt file anywhere else by using an UNC path or a regular path name in the sync section as displayed in Fig. 5-11.

5.1.5 Features to Explore in the Simple Example Workspace

Now start experimenting with the following features in this current workspace:

- build tool, inputs, outputs and tasks
- Initial data and eeprom
- redirect function calls
- debug hook

5.2 BCU Example

To create a workspace based on the BCU example, follow the steps described in Creating a New Workspace to the point where the YAML file opens in Notepad++.

1. Replace the entire content of the YAML file with the content of prepared BCU configuration YAML file located in:

   `C:\ProgramData\ETAS\VECU-BUILDER\Examples_1.2.0\BCU` for **Windows**

   Or

   `/opt/ETAS/VECU-BUILDER/Examples_1.2.0/BCU` for **Ubuntu 20.04 LTS**.

2. Continue the process as described in Working with VECU-BUILDER.

5.2.1 Show Symbol Information

To see all the symbols available in your vECU, open the `SymbolDetails` file.

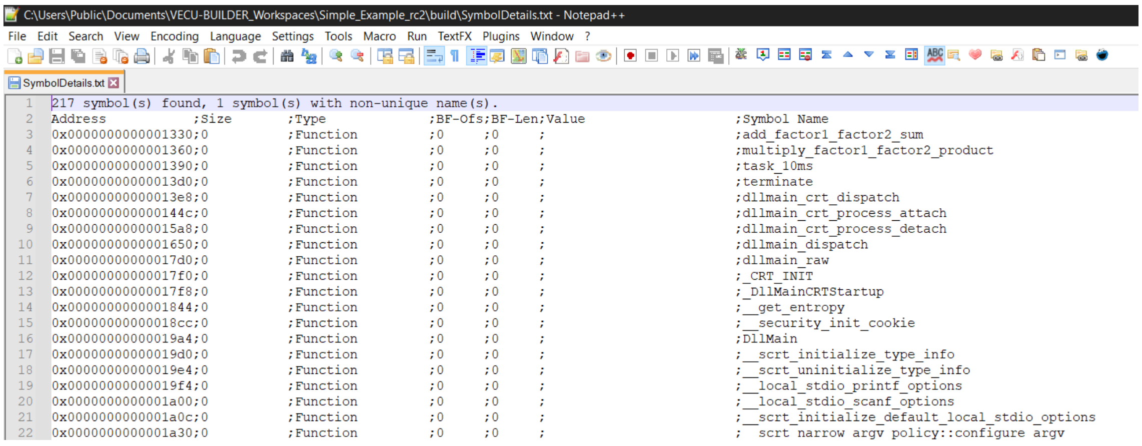
1. Run the command:

   `3c_ShowSymbolDetails.bat` on **Windows**

   Or

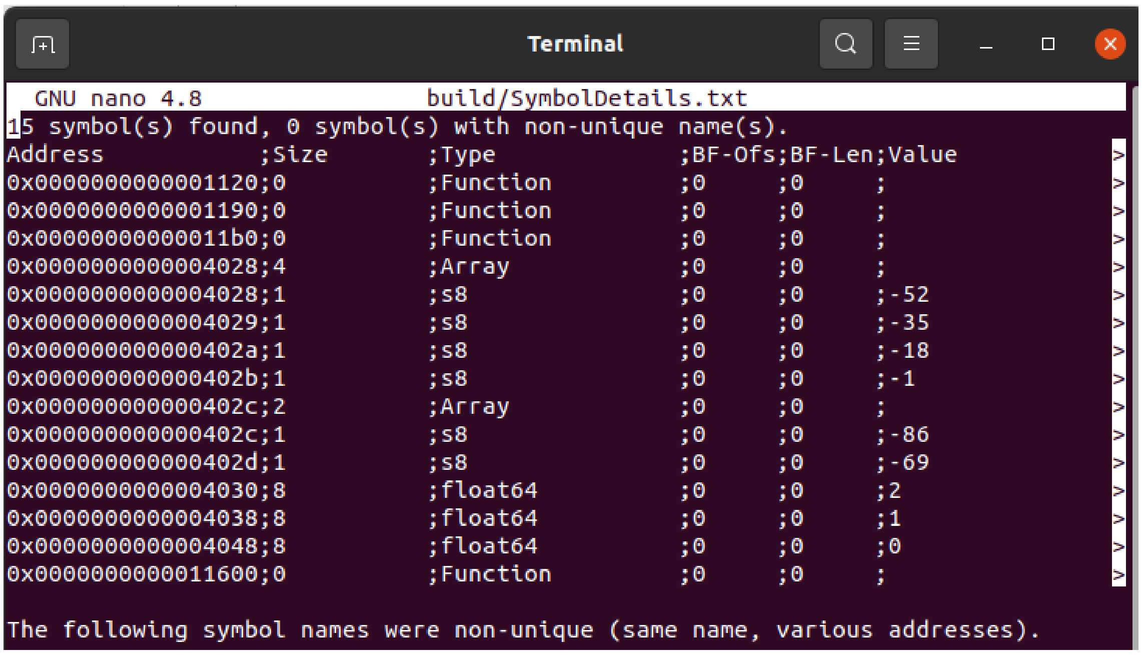
   `3c_ShowSymbolDetails.sh` on **Ubuntu 20.04 LTS**.

⇨ A text editor window (Windows) / a new terminal (Ubuntu 20.04 LTS) opens, and symbol details are shown.


C:\Users\Public\Documents\VECU-BUILDER_Workspaces\Simple_Example_rc2\build\SymbolDetails.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run TextFX Plugins Window ?
SymbolDetails.txt

```
217 symbol(s) found, 1 symbol(s) with non-unique name(s).
Address           ;Size       ;Type             ;BF-Ofs;BF-Len;Value                   ;Symbol Name
0x0000000000001330;0          ;Function         ;0     ;0     ;                        ;add_factor1_factor2_sum
0x0000000000001360;0          ;Function         ;0     ;0     ;                        ;multiply_factor1_factor2_product
0x0000000000001390;0          ;Function         ;0     ;0     ;                        ;task_10ms
0x00000000000013d0;0          ;Function         ;0     ;0     ;                        ;terminate
0x00000000000013e8;0          ;Function         ;0     ;0     ;                        ;dllmain_crt_dispatch
0x000000000000144c;0          ;Function         ;0     ;0     ;                        ;dllmain_crt_process_attach
0x00000000000015a8;0          ;Function         ;0     ;0     ;                        ;dllmain_crt_process_detach
0x0000000000001650;0          ;Function         ;0     ;0     ;                        ;dllmain_dispatch
0x00000000000017d0;0          ;Function         ;0     ;0     ;                        ;dllmain_raw
0x00000000000017f0;0          ;Function         ;0     ;0     ;                        ;_CRT_INIT
0x00000000000017f8;0          ;Function         ;0     ;0     ;                        ;_DllMainCRTStartup
0x0000000000001844;0          ;Function         ;0     ;0     ;                        ;__get_entropy
0x00000000000018cc;0          ;Function         ;0     ;0     ;                        ;__security_init_cookie
0x00000000000019a4;0          ;Function         ;0     ;0     ;                        ;DllMain
0x00000000000019d0;0          ;Function         ;0     ;0     ;                        ;__scrt_initialize_type_info
0x00000000000019e4;0          ;Function         ;0     ;0     ;                        ;__scrt_uninitialize_type_info
0x00000000000019f4;0          ;Function         ;0     ;0     ;                        ;__local_stdio_printf_options
0x0000000000001a00;0          ;Function         ;0     ;0     ;                        ;__local_stdio_scanf_options
0x0000000000001a0c;0          ;Function         ;0     ;0     ;                        ;__scrt_initialize_default_local_stdio_options
0x0000000000001a30;0          ;Function         ;0     ;0     ;                        ;  scrt narrow argv policy::configure argv
```

Fig. 5-12: Symbol Details of BCU example (Windows)



```
  GNU nano 4.8                    build/SymbolDetails.txt
15 symbol(s) found, 0 symbol(s) with non-unique name(s).
Address           ;Size         ;Type              ;BF-Ofs;BF-Len;Value        >
0x0000000000001120;0            ;Function          ;0      ;0      ;            >
0x0000000000001190;0            ;Function          ;0      ;0      ;            >
0x00000000000011b0;0            ;Function          ;0      ;0      ;            >
0x0000000000004028;4            ;Array             ;0      ;0      ;            >
0x0000000000004028;1            ;s8                ;0      ;0      ;-52         >
0x0000000000004029;1            ;s8                ;0      ;0      ;-35         >
0x000000000000402a;1            ;s8                ;0      ;0      ;-18         >
0x000000000000402b;1            ;s8                ;0      ;0      ;-1          >
0x000000000000402c;2            ;Array             ;0      ;0      ;            >
0x000000000000402c;1            ;s8                ;0      ;0      ;-86         >
0x000000000000402d;1            ;s8                ;0      ;0      ;-69         >
0x0000000000004030;8            ;float64           ;0      ;0      ;2           >
0x0000000000004038;8            ;float64           ;0      ;0      ;1           >
0x0000000000004048;8            ;float64           ;0      ;0      ;0           >
0x0000000000011600;0            ;Function          ;0      ;0      ;            >

The following symbol names were non-unique (same name, various addresses).
```

Fig. 5-13: Symbol Details of BCU example (Ubuntu 20.04 LTS)

5.2.2 A2L File Patching

Most ECU software authoring tools can generate an A2L file for you. It contains the addresses of your labels for a specific target. In addition, it may contain tool-specific statements or even non-standard clauses. The label addresses of a vECU target differ from the addresses of a physical ECU target which means the original A2L file cannot be used for an XCP connection with a vECU target.

Since the generation of A2L files is an intricate task, VECU-BUILDER excludes this functionality completely. Instead, VECU-BUILDER reads, modifies, and writes a given A2L file. This patching procedure preserves most of the original contents of the A2L file but changes all addresses to those of the vECU target. A backup copy of the original A2L file is preserved (named as `*.a2l.bak`).

Note

The A2L patching leads to an A2L file that works in ETAS INCA. This file may not work in Vector CANoe or CANape.

VECU-BUILDER includes its own XCP slave software component. Currently, it supports TCP connections only. The communication parameters for an XCP connection are part of an A2L file. VECU-BUILDER patches in the values for TCP port and IP address, which were specified in the YAML file. For instance:

Original A2L file	Patched A2L file

`/begin XCP_ON_TCP_IP`
` 0x0100 /* XCP on IP 1.0 */`
` <TCPPORT> /* Port */`
` /ADDRESS "<IPADDR>"`
`/end XCP_ON_TCP_IP`
	`/begin XCP_ON_TCP_IP`
	` */0x0100 /* XCP on IP 1.0 */`
	` 12345 /* Port */`
	` ADDRESS "127.0.0.1"`
	`/end XCP_ON_TCP_IP`

If your A2L file contains an "`XCP_ON_UDP_IP`" clause, then VECU-BUILDER rewrites it to an "`XCP_ON_TCP_IP`" clause. The integrated XCP slave supports a limited subset of the commands of the ASAM MCD-1 (XCP) standard version 1.0. It supports a limited subset of the clauses from ASAM MCD-2 (ASAP2 / A2L) standard version 1.7.1.

If your ECU software includes an XCP slave already, you may want to remove this software component from your vECU software stack.

5.2.3 A2L Name Mapping

By default, the A2L file contains the symbol names of characteristics and measurements. Sometimes the symbol names in the A2L file are renamed. Because the addresses in the A2L must refer to the original symbol names, one must map them.

Original A2L file	Mapped and patched A2L file

`/begin CHARACTERISTIC Hys-`
`teresis_LightOffIntensity`
` "unsigned integer 16bit"`
` VALUE`
` 0x00000000`
` RTAA2L_Internal_Scalar_`
`UnsignedWord`
` 0`
` CompuMethods_STEP_100_`
`OFFSET_0`
` 0`
` 100`
` DISPLAY_IDENTIFIER Hys-`
`teresis_LightOffIntensity`
`/end CHARACTERISTIC`
	`/begin CHARACTERISTIC Hys-`
	`tLiOfInt`
	` "unsigned integer 16bit"`
	` VALUE`
	` 00x00003016`
	` RTAA2L_Internal_Scalar_`
	`UnsignedWord`
	` 0`
	` CompuMethods_STEP_100_`
	`OFFSET_0`
	` 0`
	` 100`
	` DISPLAY_IDENTIFIER Hys-`
	`teresis_LightOffIntensity`
	`/end CHARACTERISTIC`

5.2.4 Features to Explore in the BCU Workspace

Now start experimenting with the following features in the current workspace:

- additional include directories
- additional compile and linker flags
- additional scripts
- xcp slave and a2l file patching and mapping

6 Controlling VECU-BUILDER

6.1 Manual Interaction

You can operate VECU-BUILDER via the provided command and batch/shell scripts. For some user inputs, such as selecting a workspace directory, the tool may display dialogs.

6.2 Command-Line Interface

Besides the manual Interaction method, you can also operate VECU-BUILDER via a command-line interface (CLI). VECU-BUILDER is a CLI native application, and the command and batch/shell scripts allow manual interaction. To get more information about the Installation using CLI, see Silent Installation.

The following arguments exist:

`--new-project-path`: Path where the workspace is to be created.

`--no-dialogs`: Suppress all dialogs and always select the default option.

`--stop-on-success`: Prevent automatic forwarding to the next stage (create workspace, import, build).

`--version`: Print the version information.

`-h`: Print list of all optional arguments.

To see all CLI optional arguments and their description

1. Open a command prompt on **Windows**

   or

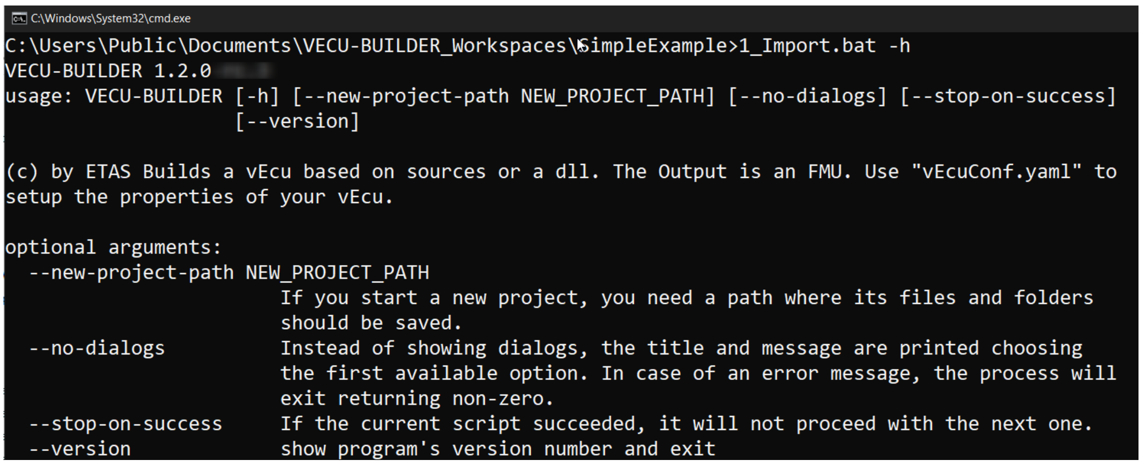
   a terminal on **Ubuntu 20.04 LTS**.

2. Execute the following command:

   `1_Import.bat -h` for **Windows**

   or

   `1_Import.sh -h` for **Ubuntu 20.04 LTS**.



```
C:\Users\Public\Documents\VECU-BUILDER_Workspaces\SimpleExample>1_Import.bat -h
VECU-BUILDER 1.2.0 [redacted]
usage: VECU-BUILDER [-h] [--new-project-path NEW_PROJECT_PATH] [--no-dialogs] [--stop-on-success]
                    [--version]

(c) by ETAS Builds a vEcu based on sources or a dll. The Output is an FMU. Use "vEcuConf.yaml" to
setup the properties of your vEcu.

optional arguments:
  --new-project-path NEW_PROJECT_PATH
                        If you start a new project, you need a path where its files and folders
                        should be saved.
  --no-dialogs          Instead of showing dialogs, the title and message are printed choosing
                        the first available option. In case of an error message, the process will
                        exit returning non-zero.
  --stop-on-success     If the current script succeeded, it will not proceed with the next one.
  --version             show program's version number and exit
```

Fig. 6-1: CLI optional arguments (Windows)

The CLI control method is ideal for integrating VECU-BUILDER into an automation pipeline. The CLI behaviour is the same as running the scripts manually: each script would call the next script to proceed through the stages of create a workspace, import, build. To change this behaviour, use `--stop-on-success`.

The following table gives an overview of which batch file uses which arguments:

argument	CreateWorkspace	1_Import	2_Build
--new-project-path	Used (required)	Ignored	Ignored
--no-dialogs	Used (optional)	Used (optional)	Used (optional)
--stop-on-success	Used (optional	Used (optional)	Ignored
--version	Used (optional)	Used (optional)	Used (optional)
-h	Used (optional)	Used (optional)	Used (optional)

Tab. 6-1: Mapping of CLI arguments to scripts

To build the SimpleExample via two command lines

After creating the workspace, stop the process so that you can copy a specific YAML file into your workspace. Then trigger the import without `stop-on-success` and let it finish the build automatically.

1. Open a command prompt on **Windows**

   or

   a terminal on **Ubuntu 20.04 LTS**.
2. Navigate to the directory where the installer is located executing the following command:

   `cd %VECUBUILDER_HOME%.`
3. Execute the following command:

   `CreateWorkspace.bat` on **Windows**

   or

   `CreateWorkspace.sh` on **Ubuntu 20.04 LTS**.

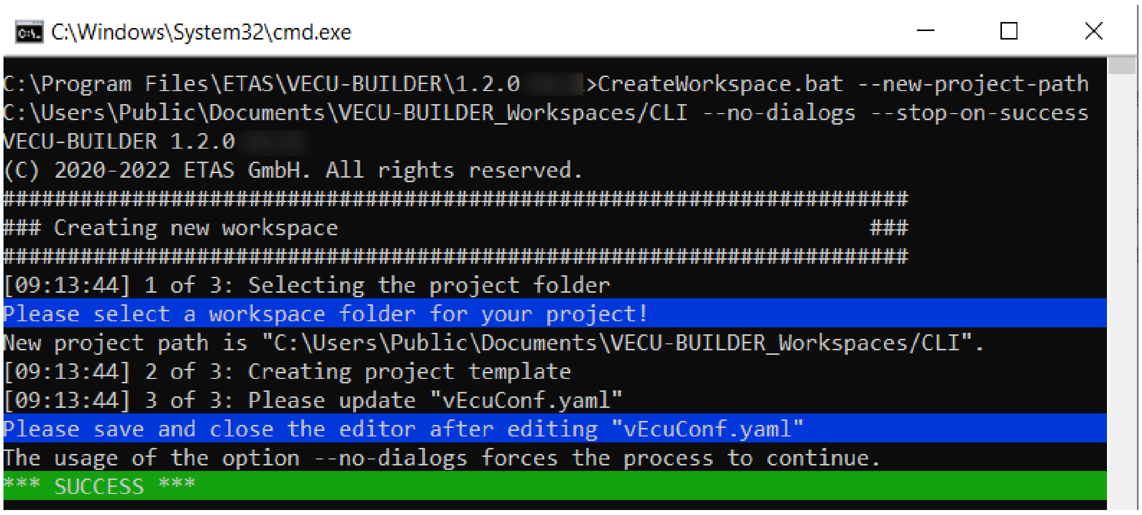
   with the arguments

   `--new-project-path <destination>`

   `--no-dialogs`

   `--stop-on-success`

   where `<destination>` points to your workspace folder.



```
C:\Windows\System32\cmd.exe
C:\Program Files\ETAS\VECU-BUILDER\1.2.0[redacted]>CreateWorkspace.bat --new-project-path
C:\Users\Public\Documents\VECU-BUILDER_Workspaces/CLI --no-dialogs --stop-on-success
VECU-BUILDER 1.2.0 [redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
######################################################################
### Creating new workspace                                         ###
######################################################################
[09:13:44] 1 of 3: Selecting the project folder
Please select a workspace folder for your project!
New project path is "C:\Users\Public\Documents\VECU-BUILDER_Workspaces/CLI".
[09:13:44] 2 of 3: Creating project template
[09:13:44] 3 of 3: Please update "vEcuConf.yaml"
Please save and close the editor after editing "vEcuConf.yaml"
The usage of the option --no-dialogs forces the process to continue.
*** SUCCESS ***
```

Fig. 6-2: Workspace creation via CLI (Windows)

Note

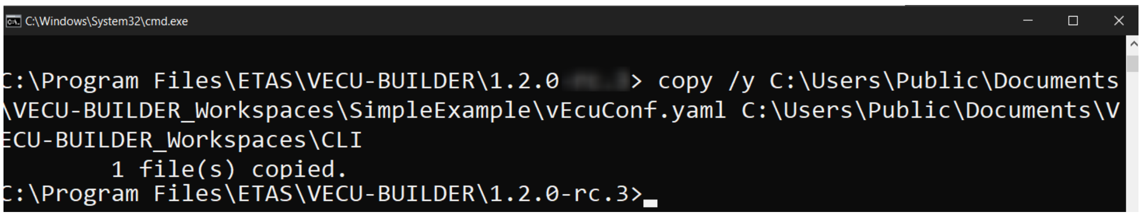
A default YAML file is used in all newly created workspaces.

Project specific YAML file can be either prepared manually or in the previous step of your automation pipeline.

<u>To use your project specific YAML file in this newly created workspace:</u>

1. Execute the following command:

   `copy /y <source> <destination>`.



```
C:\Windows\System32\cmd.exe

C:\Program Files\ETAS\VECU-BUILDER\1.2.0[redacted]> copy /y C:\Users\Public\Documents
\VECU-BUILDER_Workspaces\SimpleExample\vEcuConf.yaml C:\Users\Public\Documents\V
ECU-BUILDER_Workspaces\CLI
        1 file(s) copied.
C:\Program Files\ETAS\VECU-BUILDER\1.2.0-rc.3>_
```

Fig. 6-3: Copy your project specific YAML file (Windows)

Note

The argument `/y` suppresses the prompt and thus overwrites the destination file.

<u>To continue building your workspace:</u>

1. Navigate to this new workspace by executing the following command:

   `cd <destination>`.

2. Run the command:

   `1_Import.bat --no-dialogs` **for Windows**

   or

   `1_Import.sh --no-dialogs` for **Ubuntu 20.04 LTS**.

7 Debugging vECU

VECU-BUILDER provides useful functionalities to debug your vECU. It is possible to debug the vECU by using an Integrated Development Environment (IDE), such as Visual Studio Code or Visual Studio.

As the folder `<workspace>/vECU` is a CMake project, any IDE that can import CMake projects can be used for debugging.

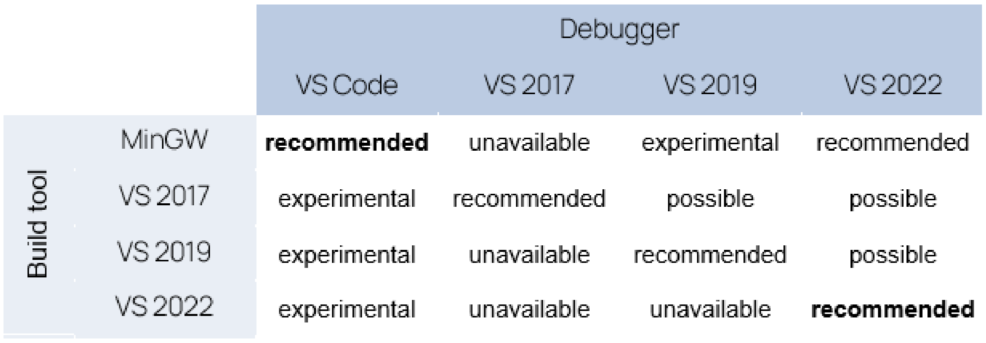
During the Build stage, the debugging environment and batch/shell script files are created enabling you to enter a debugging session in just a few clicks.

You can use the `debug_hook` attribute, which can be enabled in the YAML file. vECUs built with this attribute enabled enter their instantiation and wait for a debugger to be attached by the user before continuing.

Note

The VECU-BUILDER debugging functionality is intended to be used for debugging of a single vECU within its workspace. If your vECU is integrated into a simulation, the `debug_hook` might be the best option for debugging,

The below table summarizes the possible combinations of build tool and debugger:



		Debugger			
		VS Code	VS 2017	VS 2019	VS 2022
Build tool	MinGW	**recommended**	unavailable	experimental	recommended
	VS 2017	experimental	recommended	possible	possible
	VS 2019	experimental	unavailable	recommended	possible
	VS 2022	experimental	unavailable	unavailable	**recommended**

Tab. 7-1: Debugging possibilities

Combinations marked as experimental, are neither tested nor supported and their use is solely your responsibility.

Among the recommended combinations, two are particularly recommended for use and are described in detail in the following chapters.

7.1 Debugging with Visual Studio 2019

This chapter describes how to debug vECU built with Visual Studio 2019 using Visual Studio 2019 as the debugger.

More information about Visual Studio 2019 can be found here.

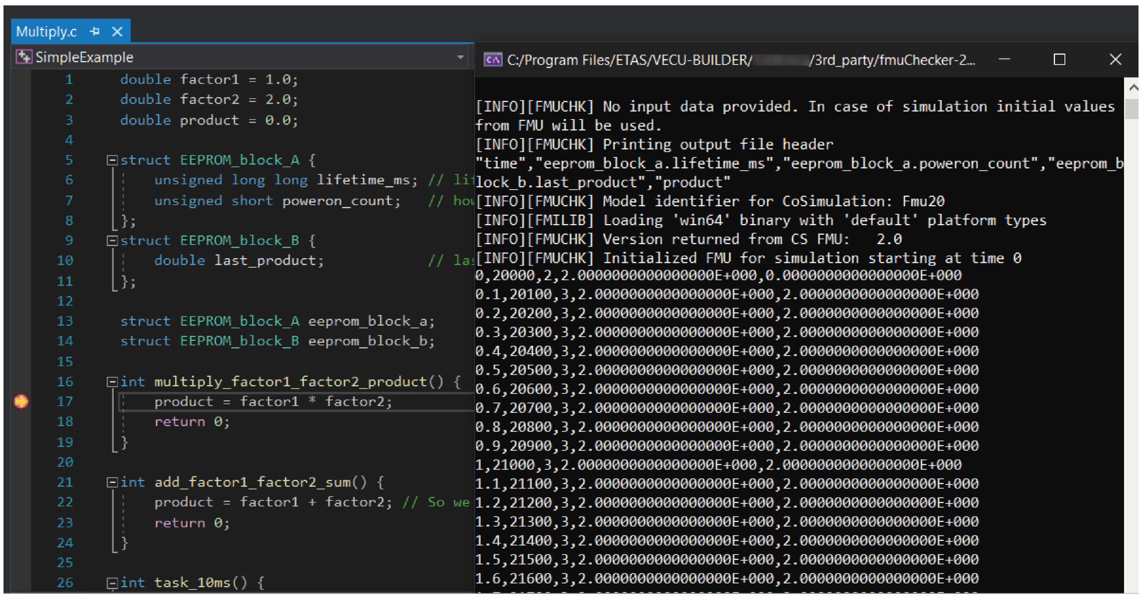
To debug with Visual Studio 2019

1. Navigate to your workspace.
2. Execute the `3b_StartDebugger.bat` file on **Windows** or `3b_StartDebugger.sh` on **Ubuntu 20.04 LTS**.

⇨ The VS2019 debugger is invoked and loads the CMake project.

3. Navigate to where you want to start debugging and place a breakpoint there.
4. In the "Menu" tab **click Debug › Start Debugging (F5)**.

⇨ FMU Checker is invoked, and the debugger is attached.



```
double factor1 = 1.0;
double factor2 = 2.0;
double product = 0.0;

struct EEPROM_block_A {
    unsigned long long lifetime_ms; // lif
    unsigned short poweron_count;   // how
};
struct EEPROM_block_B {
    double last_product;            // las
};

struct EEPROM_block_A eeprom_block_a;
struct EEPROM_block_B eeprom_block_b;

int multiply_factor1_factor2_product() {
    product = factor1 * factor2;
    return 0;
}

int add_factor1_factor2_sum() {
    product = factor1 + factor2; // So we
    return 0;
}

int task_10ms() {
```

C:/Program Files/ETAS/VECU-BUILDER/[redacted]/3rd_party/fmuChecker-2...

```
[INFO][FMUCHK] No input data provided. In case of simulation initial values
from FMU will be used.
[INFO][FMUCHK] Printing output file header
"time","eeprom_block_a.lifetime_ms","eeprom_block_a.poweron_count","eeprom_b
lock_b.last_product","product"
[INFO][FMUCHK] Model identifier for CoSimulation: Fmu20
[INFO][FMILIB] Loading 'win64' binary with 'default' platform types
[INFO][FMUCHK] Version returned from CS FMU:   2.0
[INFO][FMUCHK] Initialized FMU for simulation starting at time 0
0,20000,2,2.0000000000000000E+000,0.0000000000000000E+000
0.1,20100,3,2.0000000000000000E+000,2.0000000000000000E+000
0.2,20200,3,2.0000000000000000E+000,2.0000000000000000E+000
0.3,20300,3,2.0000000000000000E+000,2.0000000000000000E+000
0.4,20400,3,2.0000000000000000E+000,2.0000000000000000E+000
0.5,20500,3,2.0000000000000000E+000,2.0000000000000000E+000
0.6,20600,3,2.0000000000000000E+000,2.0000000000000000E+000
0.7,20700,3,2.0000000000000000E+000,2.0000000000000000E+000
0.8,20800,3,2.0000000000000000E+000,2.0000000000000000E+000
0.9,20900,3,2.0000000000000000E+000,2.0000000000000000E+000
1,21000,3,2.0000000000000000E+000,2.0000000000000000E+000
1.1,21100,3,2.0000000000000000E+000,2.0000000000000000E+000
1.2,21200,3,2.0000000000000000E+000,2.0000000000000000E+000
1.3,21300,3,2.0000000000000000E+000,2.0000000000000000E+000
1.4,21400,3,2.0000000000000000E+000,2.0000000000000000E+000
1.5,21500,3,2.0000000000000000E+000,2.0000000000000000E+000
1.6,21600,3,2.0000000000000000E+000,2.0000000000000000E+000
```

Fig. 7-1: VS 2019 Debugger attached

7.2 Debugging with Visual Studio Code

This chapter describes how to debug vECU built with MinGW using Visual Studio Code as the debugger.

Prerequisites For Debugging with Visual Studio Code

Visual Studio Code needs some extensions to be installed and by default will ask you to install them. If you accept the suggestions of Visual Studio Code, you should be ready to go.

More information about Visual Studio Code can be found here.

To debug with Visual Studio Code

1. Navigate to your workspace.
2. Right-click in your workspace and select **Open with Code**.

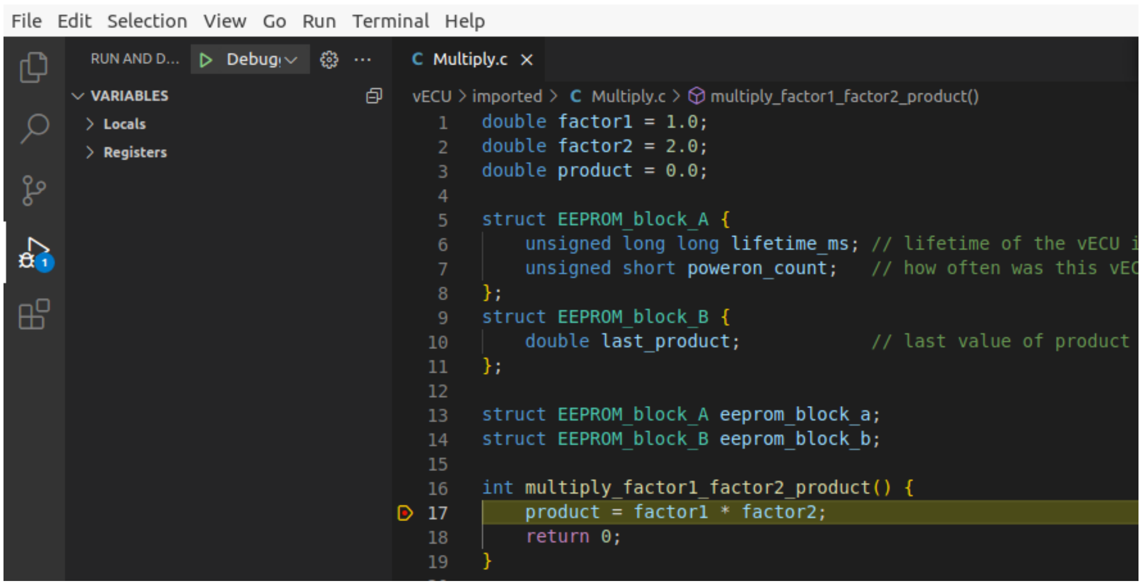
⇨ Visual Studio Code opens.

3. Navigate to where you want to start the debugging and place a breakpoint there.

   4. In the menu panel on the left click **Run and Debug**.

   Click **Start Debugging (F5)**.

⇨ FMU Checker is invoked, and the debugger is attached.


File Edit Selection View Go Run Terminal Help
RUN AND D...
Debug
Multiply.c
VARIABLES
Locals
Registers
vECU > imported > Multiply.c > multiply_factor1_factor2_product()

```
double factor1 = 1.0;
double factor2 = 2.0;
double product = 0.0;

struct EEPROM_block_A {
    unsigned long long lifetime_ms; // lifetime of the vECU i
    unsigned short poweron_count;   // how often was this vEC
};
struct EEPROM_block_B {
    double last_product;            // last value of product
};

struct EEPROM_block_A eeprom_block_a;
struct EEPROM_block_B eeprom_block_b;

int multiply_factor1_factor2_product() {
    product = factor1 * factor2;
    return 0;
}
```

Fig. 7-2: VS Code Debugger attached

8 Workspace Migration

If you have created a workspace with an older VECU-BUILDER version, this existing workspace needs to be migrated so that you can use the latest VECU-BUILDER version.

To migrate workspaces

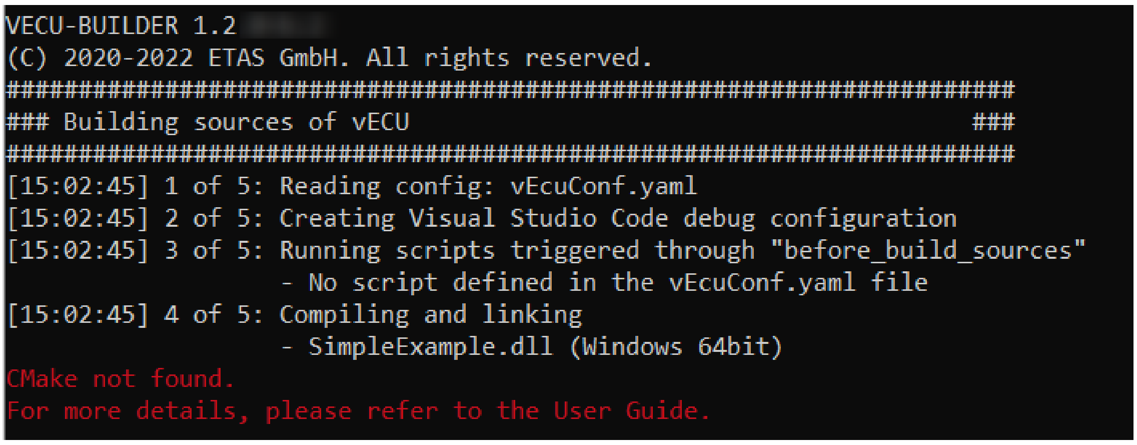
1. Create a new workspace with the default template configuration, see Creating a New Workspace.
2. Stop the process after 'Importing files and folders' stage.
   Select **Cancel** when asked whether to proceed with building sources.
3. Replace the content of `imported` folder in the new workspace with the content of your existing workspace.
4. Replace the content of `additional_scripts` folder in the new workspace with the content of your existing workspace.
5. Transfer all configuration attributes from the existing workspace `.yaml` file to the new workspace `.yaml` file.
   - TIP: The use of a comparison tool (e.g., Beyond Compare) is the most efficient way.

Your new workspace created with the latest VECU-BUILDER version is now ready and you can continue your work on an existing project in this workspace.

9 Troubleshooting

This chapter lists possible warning or error messages, their possible reasons and a possible solution to fix the issue.

9.1 CMake not found



```
VECU-BUILDER 1.2 [redacted]
(C) 2020-2022 ETAS GmbH. All rights reserved.
#######################################################################
### Building sources of vECU                                        ###
#######################################################################
[15:02:45] 1 of 5: Reading config: vEcuConf.yaml
[15:02:45] 2 of 5: Creating Visual Studio Code debug configuration
[15:02:45] 3 of 5: Running scripts triggered through "before_build_sources"
                   - No script defined in the vEcuConf.yaml file
[15:02:45] 4 of 5: Compiling and linking
                   - SimpleExample.dll (Windows 64bit)
CMake not found.
For more details, please refer to the User Guide.
```

Fig. 9-1: CMake not found error

Possible Reason

A CMake installation is required and must be registered properly. (Software Requirements for Windows 10). This registry entry is used to locate the CMake installation, if it does not exist, the build fails.

It appears as if CMake was not installed or is not properly registered on your PC.

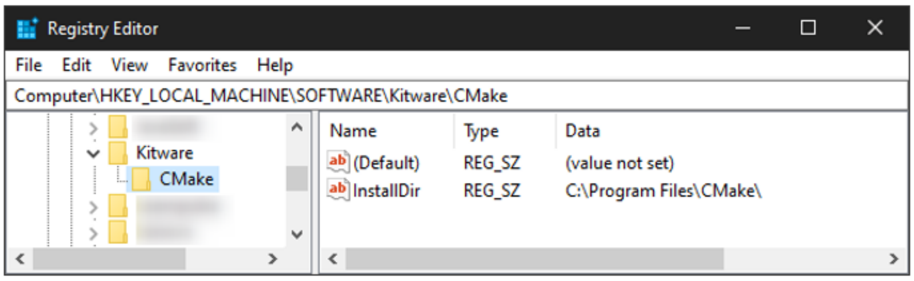
Possible Solution

Ensure the following:

- CMake is installed (version 3.15 or higher).
- Kitware and CMake keys exist in the Windows Registry.
- The CMake registry key

  `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Kitware\CMake`

  contains the string value `InstallDir` pointing to the CMake installation path:



Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Kitware\CMake

Name	Type	Data
(Default)	REG_SZ	(value not set)
InstallDir	REG_SZ	C:\Program Files\CMake\

Fig. 9-2: Windows Registry with Kitware\CMake registry key

9.2 Notepad++ Does Not Open During Workspace Creation

Notepad++ is the recommended text editor to be used along with VECU-BUILDER. For it to work as intended, it must be installed and registered properly.

If Notepad++ does not open during the Workspace Creation stage, but Windows Notepad opens instead, it is either not installed at all or is not properly registered on your PC.

<u>Possible Solution</u>

Ensure the following:

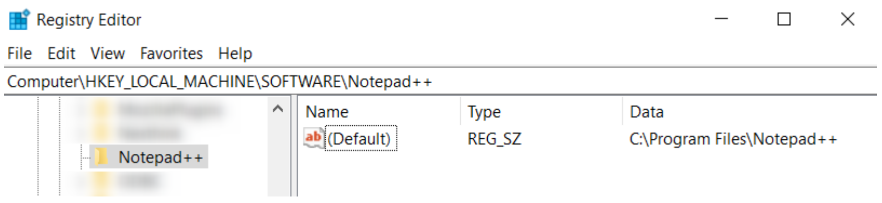
- Notepad++ is installed.
- Notepad++ key exists in the Windows Registry.

A. **For 64-bit version:**

- The Notepad++ registry key

  `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Notepad++`

  contains the string value (Default) pointing to the Notepad++ installation path:


Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Notepad++
Notepad++

Name	Type	Data
(Default)	REG_SZ	C:\Program Files\Notepad++

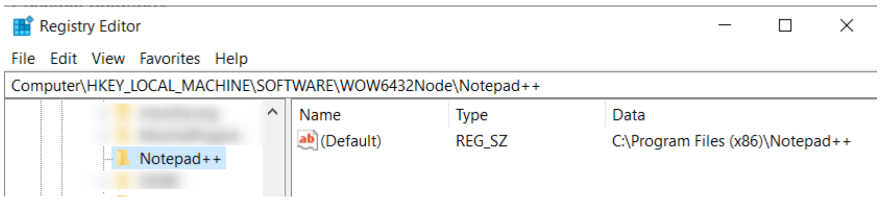
Fig. 9-3: Windows Registry with Notepad++ registry key for 64-bit version

B. **For 32-bit version:**

- The Notepad++ registry key

  `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Notepad++`

  contains the string value (`Default`) pointing to the Notepad++ installation path:


Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Notepad++
Notepad++

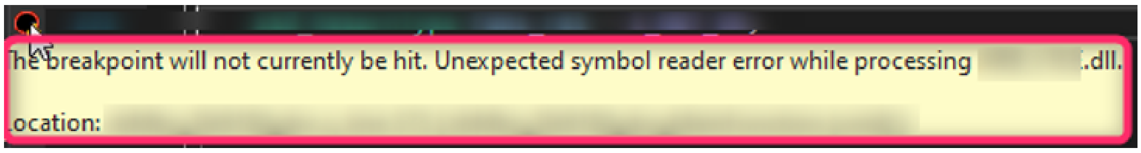
Name	Type	Data
(Default)	REG_SZ	C:\Program Files (x86)\Notepad++

Fig. 9-4: Windows Registry with Notepad++ registry key for 32-bit version

9.3 Some Breakpoints Not Being Hit

Possible Reason

Depending on your compiler configurations, the resulting vECU may be built so that some debugging information is not available. This may result in the debugger not being able to hit some breakpoints.



he breakpoint will not currently be hit. Unexpected symbol reader error while processing [redacted].dll.

ocation: [redacted]

Fig. 9-5: Breakpoint not being hit

Possible Solution

In order to prevent such compiler optimization, include the following pragma statements:

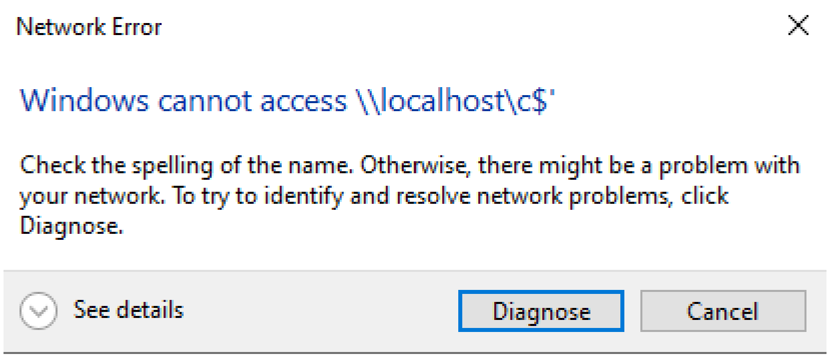
- For MSVC compiler: `#pragma optimize("", off)`
- For MinGW compiler: `#pragma GCC optimize ("O0")`

9.4 Windows Cannot Access Localhost While Using Sync Attribute in EEPROM

Possible Reason

EEPROM simulation feature requires entering the value of sync sub-attribute as UNC path.

If the defined location (e.g., C:\ drive of your localhost) cannot be accessed during the vECU execution, the data defined by the sync sub-attribute cannot be used.



Network Error

Windows cannot access \\localhost\c$'

Check the spelling of the name. Otherwise, there might be a problem with your network. To try to identify and resolve network problems, click Diagnose.

Fig. 9-6: Network Error - Localhost cannot be accessed

Possible Solution

Setup the local share and obtain the UNC pathname.

To setup a local share:

1. Navigate to the drive, you want to share. (e.g., (`C:\ drive`)
2. Right-click in the drive and click on **Properties**.
3. Click on the **Sharing tab**.

4. Click on **Advanced Sharing**. You will need Admin Rights to proceed.

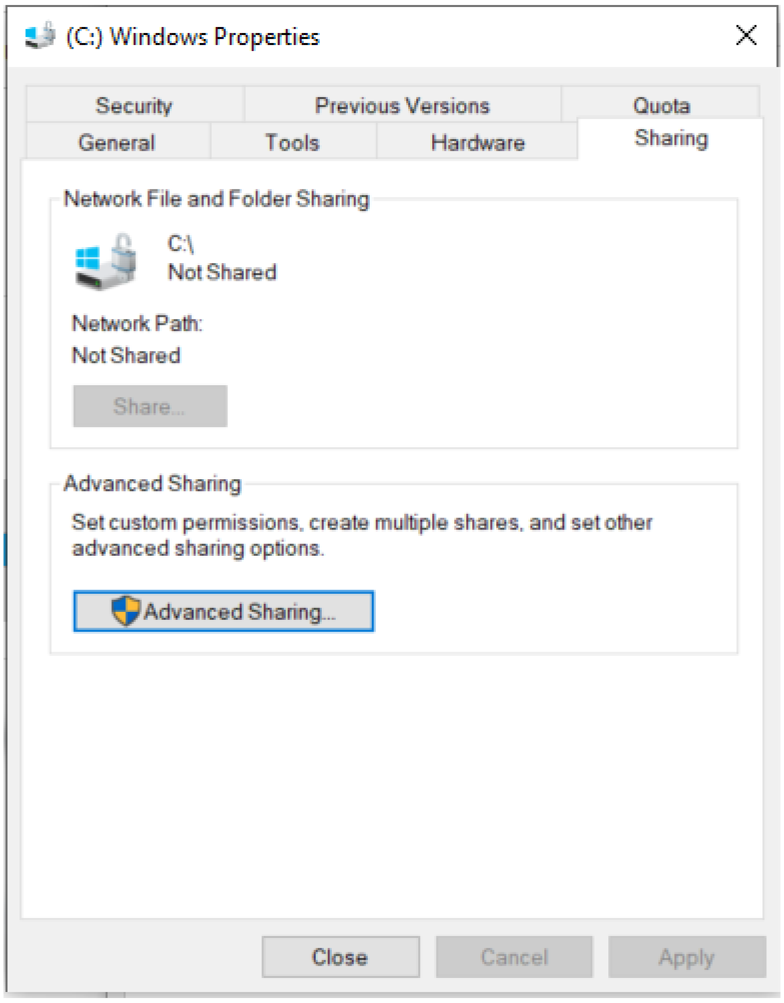
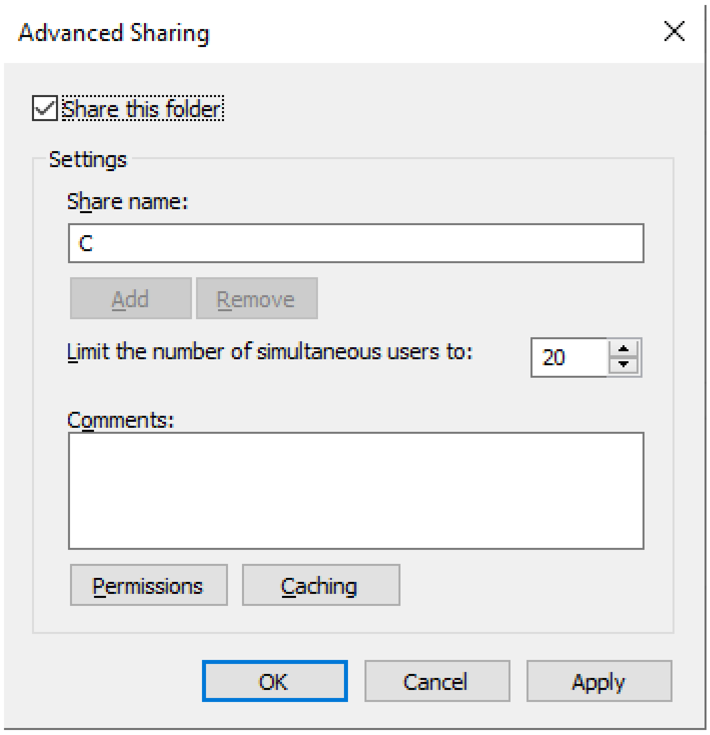


Fig. 9-7: Properties of C:\ drive dialog

5. Activate **Share this folder**.

   Click **OK**.



[✓] Share this folder

Settings

Share name: C

Limit the number of simultaneous users to: 20

Comments:

Fig. 9-8: Advanced Sharing settings dialog

⇨ The drive is now shared and the Network Path is displayed.

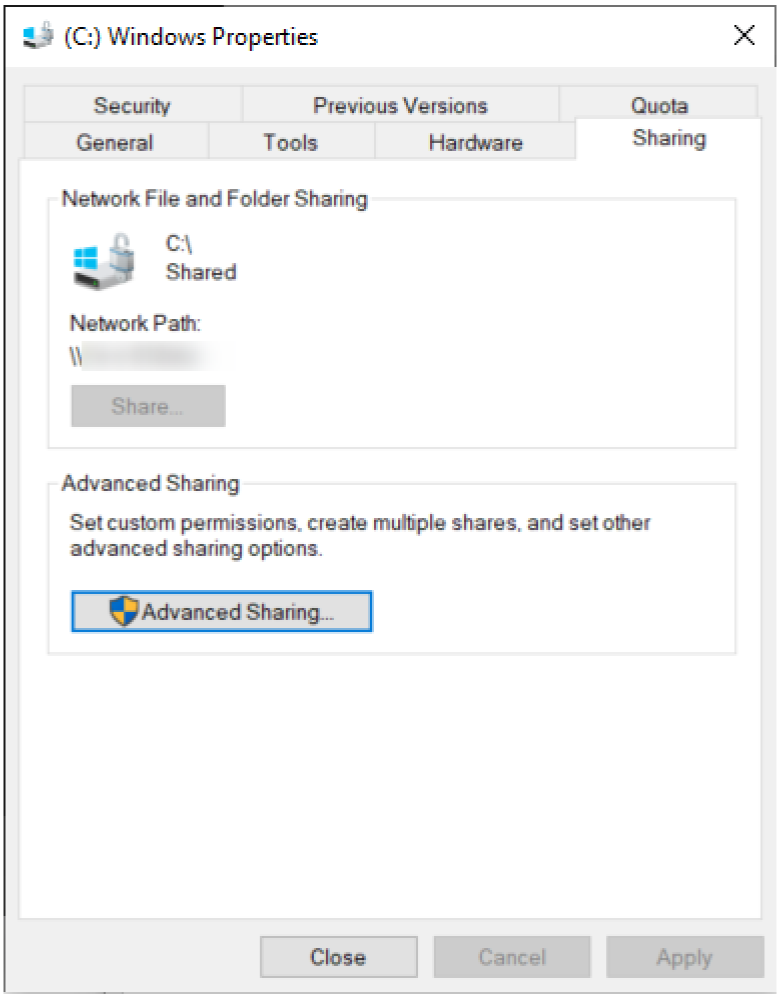


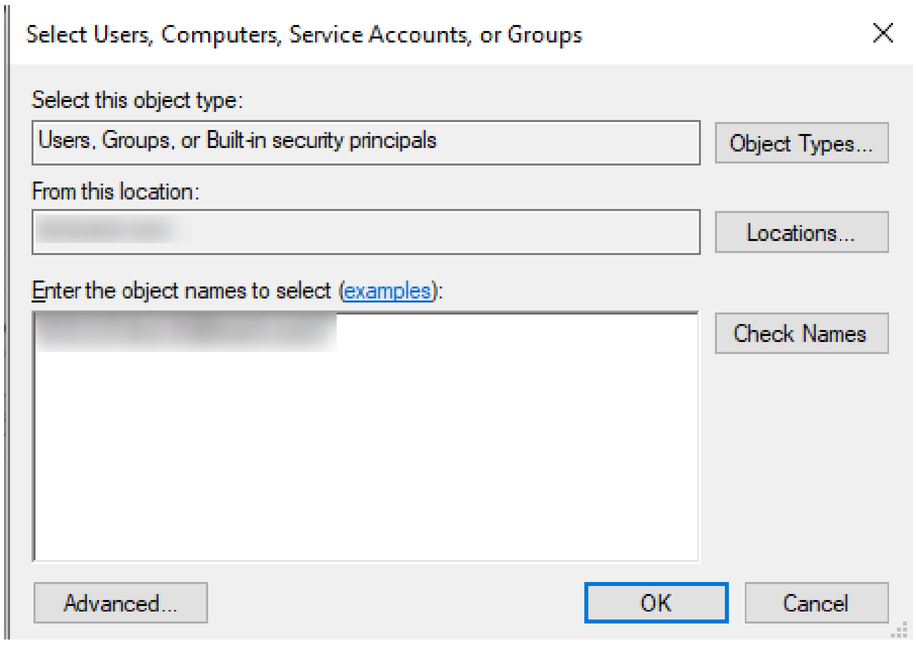
Fig. 9-9: Properties of C:\ drive dialog

The user that will be logged in during the execution of the vECU, needs to be given full control permissions to the shared location.

Per default, Windows will provide permissions to 'everyone'. The permissions should only be provided to the user, that will be logged in during the execution of the vECU. Therefore, the permissions shall to be changed for security reasons.

To change the permissions

1. Click on **Advanced Sharing**. You might need Admin Rights to proceed.
2. Click on **Permissions**.
3. Click on **Add**.
4. Enter the object name (username) to be selected.
5. Click on **Check Names**.
6. Chose the displayed name.
7. Click **OK**.


Select Users, Computers, Service Accounts, or Groups

Select this object type:
Users, Groups, or Built-in security principals

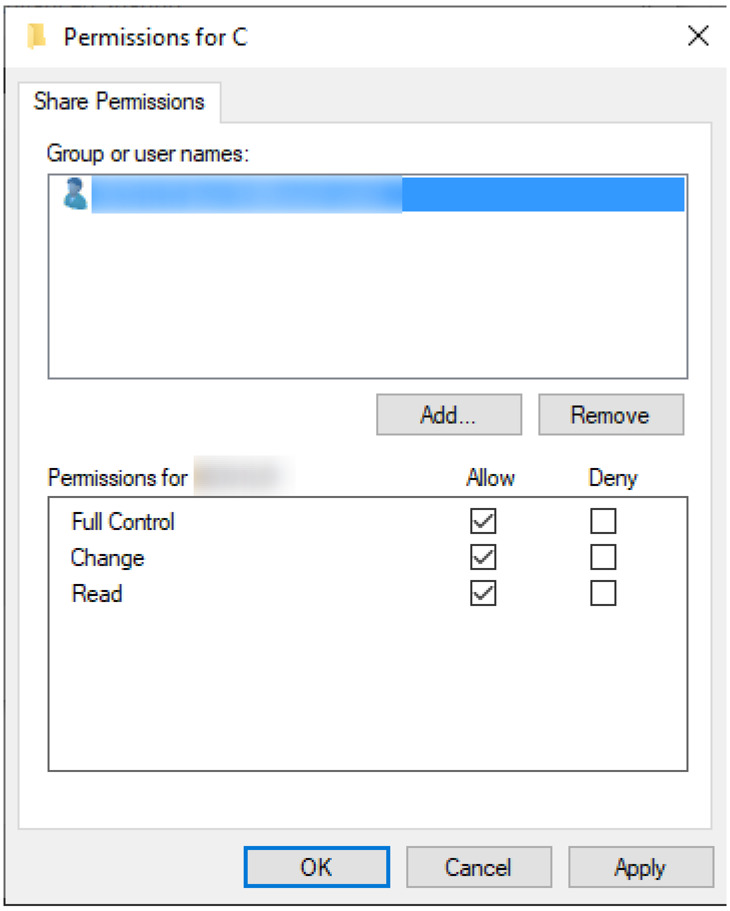
From this location:

Enter the object names to select (examples):

Fig. 9-10: Selecting the user

8. Click on the username to mark the entry.
9. Activate the permissions **Full Control** and **Change**.
10. To mark the entry, click on **Everyone**.
11. To remove the permission for 'everyone', click on **Remove**.

⇨ The group Everyone is removed and the selected user has now full permissions.



Permissions for	Allow	Deny
Full Control	✓	☐
Change	✓	☐
Read	✓	☐

Fig. 9-11: Permissions for selected user

12. To confirm the **User Selection**, click **OK**.
13. To confirm the updated **Advanced Sharing properties**, click **OK**.
14. Close the Properties window.

10 Contact Information

Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

www.etas.com/hotlines

ETAS Headquarters

ETAS GmbH

Borsigstraße 24	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	Internet:	www.etas.com