

ETAS VECU-BUILDER V1.8



www.etas.com

Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract.

Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

© Copyright 2024 ETAS GmbH, Stuttgart

The names and designations used in this document are trademarks or brands belonging to the respective owners.

VECU-BUILDER V1.8 | User Guide R11 EN I10.2024

Contents

| 1 | Introduction | | |
|---------------------|--------------------------------------|---|----|
| 1.1 | Intended use | | |
| 1.2 | Target group | | |
| 1.3 | Data protection | | |
| 1.4 | Data and | d information security | 6 |
| | 1.4.1 | Data and storage locations | 6 |
| | 1.4.2 | Technical and organizational measures | 7 |
| 2 | About VECU-BUILDER | | |
| 2.1 | VECU-BUILDER on YouTube | | |
| 2.2 | Warning and error messages | | |
| 2.3 | Basics | | |
| 2.4 | Virtual ECU | | |
| 2.5 | vECU creation process workflow | | |
| 2.6 | Functional Mock-up Interface (FMI)12 | | |
| 3 | Installation | | |
| 3.1 | Hardware requirements | | |
| 3.2 | Preparation | | |
| 3.3 | Installation content | | |
| 3.4 | Licensin | g | 14 |
| 3.5 Installation on | | on on Windows 10 | 15 |
| | 3.5.1 | Software requirements for Windows 10 | 15 |
| | 3.5.2 | Manual installation of VECU-BUILDER | 15 |
| | 3.5.3 | Silent installation of VECU-BUILDER | 16 |
| | 3.5.4 | Uninstalling VECU-BUILDER on Windows 10 | 17 |
| 3.6 | Installati | on on Ubuntu 22.04 LTS | 18 |
| | 3.6.1 | Software requirements for Ubuntu 22.04 LTS | 18 |
| | 3.6.2 | Installing License Manager (LiMa) on Ubuntu 22.04 LTS | 18 |
| | 3.6.3 | Opening ETAS License Manager on Ubuntu 22.04 LTS | 19 |
| | 3.6.4 | Activating the LiMa license | 19 |
| | 3.6.5 | Installing VECU-BUILDER on Ubuntu 22.04 LTS | 20 |
| | 3.6.6 | Uninstalling VECU-BUILDER on Ubuntu 22.04 LTS | 20 |
| 3.7 | Installati | on on Ubuntu 22.04 LTS for WSL | 21 |
| | 3.7.1 | Software requirements for Ubuntu 22.04 LTS on WSL | 21 |
| | 3.7.2 | Installing WSL on Windows | 21 |

| | 3.7.3 | Installing Ubuntu 22.04 LTS on WSL | 21 | |
|-----|---|---|--------|--|
| | 3.7.4 | Installing dependent software packages | 22 | |
| | 3.7.5 | Installing License Manager (LiMa) on Ubuntu 22.04 LTS for WSL | 23 | |
| | 3.7.6 | Installing VECU-BUILDER on Ubuntu 22.04 LTS for WSL | 24 | |
| | | 3.7.6.1 Opening ETAS License Manager on Ubuntu 22.04 LTS for WS | SL .24 | |
| | 3.7.7 | Activating the LiMa license | 25 | |
| | 3.7.8 | Uninstalling VECU-BUILDER on Ubuntu 22.04 LTS for WSL | 25 | |
| 3.8 | Installed | d files and folders | 26 | |
| 4 | VECU-BUILDER without admin credentials2 | | | |
| 4.1 | Use of p | oortable version without admin credentials on Windows | 29 | |
| 4.2 | Use of portable version without admin credentials on Ubuntu 22.04 LTS | | | |
| 5 | Working with VECU-BUILDER | | | |
| 5.1 | Creating | g a new workspace | 33 | |
| | 5.1.1 | Creating a workspace on Windows | 33 | |
| | 5.1.2 | Creating a workspace on Ubuntu 22.04 LTS | 35 | |
| 5.2 | Importir | ng files and folders | 37 | |
| 5.3 | Building the vECU | | | |
| 5.4 | Building the FMU | | | |
| 5.5 | Workspace content | | | |
| 5.6 | Configuration | | | |
| 6 | Explorin | g the examples/templates | 56 | |
| 6.1 | Simple e | example | 56 | |
| | 6.1.1 | fmusim | 56 | |
| | 6.1.2 | Difference between debug and release vECUs | 58 | |
| | | 6.1.2.1 Keeping symbol information in a release FMU | 59 | |
| | 6.1.3 | InitialData functionality | 59 | |
| | 6.1.4 | eeprom functionality | 64 | |
| | 6.1.5 | Usage of link_into_project | 67 | |
| | 6.1.6 | ARXML-defined tasks | 69 | |
| 6.2 | BCU exa | ample (only available for Windows) | 70 | |
| | 6.2.1 | Show symbol information | 71 | |
| | 6.2.2 | A2L file patching | 71 | |
| | 6.2.3 | A2L name mapping | 73 | |
| | 6.2.4 | HEX file generation | 74 | |
| | 6.2.5 | Example of additional scripts - A2L characteristics as parameters | 75 | |
| 6.3 | EventTr | igger example | 76 | |

| | 6.3.1 | Event-triggered tasks | 77 |
|------|--------------------------------------|--|-----|
| | 6.3.2 | Task scheduling with task trigger defined as cyclic | 77 |
| 6.4 | Templat | e for plug-in V1 (FMI2) | 79 |
| | 6.4.1 | Plug-In feature | 79 |
| | 6.4.2 | Plug-In configuration | 81 |
| | 6.4.3 | Plug-In interface | |
| | 6.4.4 | What a plug-in can do with tasks | |
| 6.5 | Templat | e for plug-in V2 (FMI3) | |
| | 6.5.1 | Plug-In feature V2 | |
| | 6.5.Z | Plug-In confuguration v2 | |
| | 654 | What a plug-in cap do with tasks V2 | 96 |
| 7 | Controll | | |
| 71 | Manuali | ntoraction | |
| 7.1 | Common | | |
| /.Z | Comman | | |
| 1.5 | Ubuntu | 22.04 LIS Command Line Interface | |
| 8 | Debuggi | ng vECU | |
| 8.1 | Debugging with Visual Studio 2019103 | | |
| 8.2 | Debuggi | ng with Visual Studio Code | |
| 9 | Troubles | shooting | |
| 9.1 | CMake r | not found | 106 |
| 9.2 | Notepac | I++ does not open during workspace creation | |
| 9.3 | Some br | eakpoints not being hit | |
| 9.4 | (Symbol | Info.dll) the *.die file is too large to load | 108 |
| 9.5 | Windows | s cannot access localhost while using sync attribute in EEPROM | |
| 9.6 | Redirect | ing function calls did not work as expected | 114 |
| 9.7 | License | check failed | 114 |
| 9.8 | Building | sources failed | 114 |
| 9.9 | Indentat | ion errors in YAML file | 115 |
| 9.10 | Failed to | parse symbols | |
| 9.11 | Could no | ot load the vECU binary | 117 |
| 9.12 | Skipping | plug-in | |
| 9.13 | Encoding | g cannot be defined for a VARVAL file | 119 |
| 9.14 | Encoding | g of DCM file is not supported | |
| 10 | Contact | information | 121 |
| | | | |

1 Introduction

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 create a virtual ECU for microcontrollers using existing ECU source code or precompiled binaries. This virtual ECU is intended for simulation, debugging, and pre-calibration of ECU software in a computer-based virtual simulation environmenIn general, virtual ECUs may not be real-time capable. If you use a virtual ECU to control physical devices, the system may react 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. Adhere to the ETAS Safety Advice (see documentation folder).

1.2 Target group

This product is intended for skilled and qualified personnel in development of auto-motive ECU software (e.g., function developer, application engineer, ECU soft-ware integrator, systems 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 Data protection

If the product contains functions that process personal data, legal requirements of data protection and data privacy laws shall be complied with by the customer. As the data controller, the customer usually designs subsequent processing. Therefore, he must check if the protective measures are sufficient.

1.4 Data and information security

To securely handle data in the context of this product, see the next sections about data and storage locations as well as technical and organizational measures.

1.4.1 Data and storage locations

The following sections give information about data and their respective storage locations for various use cases.

License Management

When using the ETAS License Manager in combination with user-based licenses that are managed on the FNP license server within the customer's network, the following data are stored for license management purposes:

Data

- Communication data: IP address
- User data: User ID

Storage location

- FNP license server log files on the customer network

When using the ETAS License Manager in combination with host-based licenses that are provided as FNE machine-based licenses, the following data are stored for license management purposes:

Data

- Activation data: Activation ID
 - Used only for license activation, but not continuously during license usage

Storage location

FNE trusted storage

Windows:
C:\ProgramData\ETAS\FlexNet\fne\license\ts
Linux:
/usr/share/ETAS/LiMa/fne/license/ts/

1.4.2 Technical and organizational measures

We recommend that your IT department takes appropriate technical and organizational measures, such as classic theft protection and access protection to hardware and software.

2 About VECU-BUILDER

VECU-BUILDER is a tool for building virtual ECUs (vECUs) for simulation, debugging, and pre-calibration of ECU software within a computer-based virtual simulation environment.

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

VECU-BUILDER is built on Python and CMake. It can take inputs in the form of C/C++ source codes or binaries, such as object files or shared libraries containing symbol information. Unlike AUTOSAR Classic, the configuration of a vECU is performed in a single YAML file (vEcuConf.yaml). The properties are configured within this text-based file, which 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. These FMUs can be integrated into any FMI-compliant simulation master.

2.1 VECU-BUILDER on YouTube

You can access a playlist on YouTube that features VECU-BUILDER and its functionalities. To open the playlist, click the image below.



2.2 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.



Fig. 2-1: 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.





2.3 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. These files are reflected in the imported folder (vECU\imported) in the vECU build process.

The result is a stand alone FMU:

- The FMU contains the model description (e.g. its variables) as XML file.
- The FMU contains the access to calibration and measurement variables via patched A2L file.



The FMU contains an executable model as DLL/SO file.



2.4 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.

2.5 vECU creation process workflow

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

- A. Prepare sources
 - Fix directives that refer to header files in code
 - 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 completing the first iteration of an vECU it can be used to perform further steps outside of VECU-BUILDER:

- Integration of the vECU with plant models and execution in a Co-Simulation environment
- Running and testing the vECU in an experimental environment
- Measuring and calibrating the vECU
- Debugging using a source code editor

2.6 Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) is a free specification that outlines a container and interface for exchanging dynamic simulation models. VECU-BUILDER supports Co-simulation (CS). The FMU is provided with its own solver in Co-simulation.

For more information about FMI, see FMI standard.

Fig. 2-4 shows the general steps for Co-Simulation in FMI versions 2 and 3.

Steps 4-6 are repeated until fmi2Terminate Or fmi3Terminate function is called.





Fig. 2-4: General steps for Co-Simulation

VECU-BUILDER supports FMI2 and FMI3 by a plug-in concept. For more information about plug-in concept, see Template for plug-in V1 (FMI2) and Template for plug-in V2 (FMI3).

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 22.04 LTS
- Ubuntu 22.04 on WSL for Windows

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 installing, ensure that your computer meets the hardware and software requirements. Ensure that you have the appropriate user rights based on your operating system and network connection.

(i) Note

If you lack appropriate user rights, contact your system administrator.

3.3 Installation content

You can download the installation content from ETAS license and download portal. Log in using your email address, download the installation content, and then proceed with the installation.

The installation content includes information about the open-source software attributions, important information such as Safety Advice or the User Guide as well as the executable installation files.

i) Note

If the download files or download link are not available, contact technical support for assistance.

3.4 Licensing

A valid license is required to use the software. You can obtain a license in one of the following ways:

- from your tool coordinator
- via the self-service portal on the ETAS website at www.etas.com/support/licensing
- via the ETAS License Manager

To activate the license, you must enter the Activation ID that you received from ETAS during the ordering process.

For more information about ETAS license management, see the ETAS License Management FAQ or the ETAS License Manager help.

To open the ETAS License Manager help

The ETAS License Manager is available on your computer after the installation of any ETAS software.

 From the Windows Start menu, select E > ETAS > ETAS License Manager. or

Under Linux, use LiMaQt.sh, which you can find at the following location: ./usr/share/ETAS/LiMa/x32/bin/.

The ETAS License Manager opens.

2. Click in the ETAS License Manager window and press F1.

The ETAS License Manager help opens.

VECU-BUILDER performs the following checks:

- Check of the product license when building FMUs.
- Check of the run time license during run time of the vECU.
- Check of 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 | ETAS License Manager CMake (version ≥3.15) |
|----------------------|---|
| | |
| Recommended Software | |
| Optional Software | Visual Studio 2015, 2017, 2019, 2022 |
| | Visual Studio Code |
| | Python |

3.5.2 Manual installation of VECU-BUILDER

- 1. Navigate to the directory where the installation file is located and execute the VECU_BUILDER_installer_1.8.0.exe file.
- ⇒ The Setup Wizard opens.
- 2. Click Next.
- ⇒ The Safety Advice window opens.
- 3. Read the Safety Advice carefully, then select "I read and accept the Safety Advice".
- 4. Click Next.
- ➡ The Installation Path window opens.
- 5. Accept the default path (click **Next**) or click **Browse** to select a custom location.
- ⇒ The Ready to Install window opens.
- 6. Click Install.
- The installation is performed, its progress is shown via a progress bar.
- 7. Click Next.
- ⇒ The Third-party Software window opens.
- 8. Install CMake (required) and Notepad++ (recommended).
 - See the links below in the installation dialog:

CMake (version 3.15 or higher)

Notepad++

- 9. Click Next.
- ⇒ The Completing VECU-BUILDER Setup window opens.
- 10. Optional: To open the documentation folder ,activate the Open VECU-BUILDER documentation checkbox.
- 11. Click Finish.
- ⇒ The installation is completed, and you can use VECU-BUILDER.

3.5.3 Silent installation of VECU-BUILDER

You can also use silent installation in addition to manual installation. The installation process varies based on whether you use the Command Prompt or the PowerShell.

Silent installation using command prompt

- 1. Open the command prompt.
- Navigate to the directory where the installer (VECU-BUILDER_ installer_1.8.0.exe) is located.
- 3. Install VECU-BUILDER using the following command:

start cmd.exe /c VECU-BUILDER_installer_1.8.0.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.

Wicrosoft Windows [Version 10.0.19045.2965] (c) Microsoft Corporation. All rights reserved. Start cmd.exe /c VECU-BUILDER_installer_<version>.exe/S /INST="path_to_installation_dir" /EULAAccepted="YES" / SafetyHintsAccepted="YES"

A new command prompt window opens and installation starts.

```
■ CWWndowslystem32/cmdexe
User has accepted EULA and Safety Advice.
Silent installation has started.Please wait for completion...
```

Silent installation using PowerShell

- 1. Open the PowerShell.
- Navigate to the directory where the installer (VECU-BUILDER_ installer_1.8.0.exe) is located.
- 3. Install VECU-BUILDER using the following command:

```
Start-Process -FilePath".\VECU-BUILDER_installer_
1.8.0.exe/" -ArgumentList "/c /S /INST= path_to_install-
ation_dir /EULAAccepted=YES /SafetyHintsAccepted=YES" -
Wait
```

```
where path_to_installation_dir contains a path to a directory where the software is to be installed.
```



Or

```
Install VECU-BUILDER using the following command:
```

```
Start-Process -FilePath " path_to \VECU-BUILDER_
installer_1.8.0.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_1.8.0.exe) is located and path_to_installation_dir contains a path to a directory where the software is to be installed.



Installation starts.

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

3.5.4 Uninstalling VECU-BUILDER on Windows 10

- 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 22.04 LTS

3.6.1 Software requirements for Ubuntu 22.04 LTS The following Software Requirements need to be met:

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

3.6.2 Installing License Manager (LiMa) on Ubuntu 22.04 LTS

Prior to installing VECU-BUILDER, you need to manually install the ETAS License Manager (LiMa).

The installation debian packages for LiMa are delivered next to the VECU-BUILDER installation debian package.

LiMa-1.8.11.24-Linux.deb LiMaX64-1.8.11.24-Linux.deb

Fig. 3-1: LiMa installation debian packages

- 1. Navigate to the directory where the LiMa debian Package files are located.
- 2. Install LiMa using the following command:

sudo apt install ./LiMa-1.8.11.24-Linux.deb

3. Install LiMaX64 unsing the following command:

sudo apt install ./LiMaX64-1.8.11.24-Linux.deb

⇒ LiMa and all its components are installed.

3.6.3 Opening ETAS License Manager on Ubuntu 22.04 LTS

- Navigate to the following direction: /usr/share/ETAS/LiMa/x32/bin
- 2. Open a new terminal.
- 3. Enter the following command:
 - ./LiMaQt.sh
- ➡ LiMa was opened.

| FI | /usr/share/ETAS/LiMa/x32/bin Q = _ 🗆 🗙 | | |
|--|---|--|--|
| QOb [.] QOt | :/usr/share/ETAS/LiMa/x32/bin\$./LiMaQt.sh iect::connect: signal_not_found_in_LiMa::Ot::LineEdit ETAS License Manager – © 😣 | | |
| Onl Onl Cas Nun Onl Cas Nun Onl Cas Nun Onl Cas Nun Onl | File License Tools Device Help ion implementation All ion implementation ation ion implementation ion implementation ion implementation | | |
| Case insensitive sorting unsupported in the posix collation implementation Numeric mode unsupported in the posix collation implementation Only C and default locale supported with the posix collation <u>implementation</u> | | | |

3.6.4 Activating the LiMa license

There are several possibilities to activate the license. For more information about ETAS license management, see the ETAS License Management FAQ or the ETAS License Manager help.

3.6.5 Installing VECU-BUILDER on Ubuntu 22.04 LTS

- 1. Navigate to the directory where the Debian Software Package file (VECU-BUILDER_installer_1.8.0.deb) is located.
- 2. Open a new terminal.
- 3. Install VECU-BUILDER using the following command:

sudo apt install ./VECU-BUILDER_installer_1.8.0.deb

i) Note

VECU-BUILDER has dependencies on other software. The dependent software packages will be installed during the installation. An Internet connection is required to install the dependent software packages.

- 4. Accept the installation of dependent packages.
- ⇒ The packages are selected and unpacked.
- 5. Accept the Safety Advice.
- ⇒ The VECU-BUILDER package deployment is completed.
- 6. Log out and log in to enable environment variables to be set.

3.6.6 Uninstalling VECU-BUILDER on Ubuntu 22.04 LTS

- 1. Open a new terminal.
- 2. Uninstall VECU-BUILDER using the following command: sudo apt remove vecu-builder
- Sou are asked if you want to continue uninstalling.
- 3. To continue uninstalling, enter < Y > and press < ENTER >.
- ⇒ The VECU-BUILDER package is removed.

3.7 Installation on Ubuntu 22.04 LTS for WSL

It is possible to create a Linux-vECU from a Windows host. To be able to create a Linux-vECU from a Windows host, the following prerequisites must be met:

- WSL is installed on Windows.
- Ubuntu 22.04 LTS is installed on WSL.
- LiMa is installed on WSL.
- VECU-BUILDER is installed on WSL.

3.7.1 Software requirements for Ubuntu 22.04 LTS on WSL

The following Software Requirements need to be met:

| Required Soft- | ETAS License Manager |
|------------------------|---|
| ware | cmake |
| | build-essential |
| | gcc-multilib |
| | g++-multilib |
| | libssl-dev:i386 |
| | linux-libc-dev:i386 |
| | gnome-terminal (for dialog mode softwares) |
| Optional Soft- ware | Visual Studio Code (for debugging, installed on Windows com- puter host) |
| | Python |
| | nano |
| | gdb (for debugging) |

3.7.2 Installing WSL on Windows

To install WSL on Windows, see Install WSL command.

3.7.3 Installing Ubuntu 22.04 LTS on WSL

- 1. Open PowerShell.
- 2. Check what distributions are available online in PowerShell using the following command:

wsl --list --online

3. Install Ubuntu 22.04 LTS on WSL using the following command:

wsl --install -d ubuntu-22.04

➡ Ubuntu 22.04 LTS is installed.

3.7.4 Installing dependent software packages

In order to install VECU-BUILDER you need to install dependent software packages.

i) Note

The installation of dependent software packages requires unrestricted internet access. If your computer is not permitted to connect to the official package repositories, the sudo apt commands will fail. For further information, contact your system administrator.

i) Note

Downloading dependencies or installing VECU-BUILDER only runs in WSL1, working with VECU-BUILDER only runs in WSL2. Ensure that the WSL version aligns with the specific action. If necessary, you need to change the version.

- For WSL1:
- wsl --set-version Ubuntu-22.041
- For WSL2:
- wsl --set-version Ubuntu-22.04 2
 - 1. In PowerShell, check the WSL Ubuntu version using the following command:

```
wsl -l -v
```

If it is not 1, set the version to 1, using the following command:

```
wsl --set-version Ubuntu-22.04 1
```

- 2. Open Ubuntu 22.04 LTS command line interface.
- Install the i386 architecture using the following command: sudo dpkg --add-architecture i386
- Install libc6-i386 using the following command: sudo apt install -y libc6-i386
- Install lsb using the following command: sudo apt install -y lsb
- Run a package update using the following command: sudo apt update
- Run a package upgrade using the following command: sudo apt upgrade
- Install gnome-terminal using the following command: sudo apt install gnome-terminal

You need gnome-terminal for debugging, ETAS License Manager (LiMa) and the Dialog mode.)

9. Install gdb using the following command:

sudo apt install gdb

You need gdb for debugging.

3.7.5 Installing License Manager (LiMa) on Ubuntu 22.04 LTS for WSL

Prior to installing VECU-BUILDER, you need to manually install the ETAS License Manager (LiMa).

The installation debian packages for LiMa are delivered next to the VECU-BUILDER installation debian package.

LiMa-1.8.11.24-Linux.deb LiMaX64-1.8.11.24-Linux.deb

Fig. 3-2: LiMa installation debian packages

- 1. In order to install LiMa copy the required installation debian packages for LiMa to user home on Ubuntu.
- 2. Open Ubuntu 22.04 LTS command line interface.
- Install LiMa using the following command: sudo apt install ./LiMa-1.8.11.24-Linux.deb
- 4. Install LiMaX64 using the following command:
 - sudo apt install ./LiMaX64-1.8.11.24-Linux.deb
- ⇒ LiMa and all its components are installed.

3.7.6 Installing VECU-BUILDER on Ubuntu 22.04 LTS for WSL

i) Note

The installation of VECU-BUILDER only works if you have unrestricted access to the internet. The sudo apt commands will fail if your computer is not allowed to connect your Linux to the official package repositories. In this case, ask your IT department for help.

1. In order to install VECU-BUILDER copy VECU-BUILDER_installer_1.8.0.deb to user home on Ubuntu.

```
    > Linux > Ubuntu-20.04 > home > <user>
    Name
    VECU-BUILDER_installer_<version>.deb
```

2. Install VECU-BUILDER using the following command:

```
sudo apt install ./VECU-BUILDER installer 1.8.0.deb
```



VECU-BUILDER has dependencies on other software. The dependent software packages will be installed during the installation. An Internet connection is required to install the dependent software packages.

3. Close and restart Ubuntu 22.04 LTS.

To create a workspace using WSL Ubuntu 22.04 LTS, see Ubuntu 22.04 LTS Command Line Interface.

3.7.6.1 Opening ETAS License Manager on Ubuntu 22.04 LTS for WSL <u>Prerequisites:</u>

- Ensure to have the latest Ubuntu 22.04 LTS version and all related packages installed.
- Ensure to have XTerm (Ubuntu-22.04) installed.
- Ensure to have UXTerm (Ubuntu-22.04) installed.

| U |
|-------------------------|
| Ubuntu 22.04.3 LTS |
| Ubuntu 22.04.3 LTS |
| i UXTerm (Ubuntu.22.04) |
| 🙏 UTerm (Ubuntu.22.04) |

Fig. 3-3: UXTerm and XTerm for Ubuntu 22.04 LTS

For more information about running Linux GUI apps on the Windows Subsystem for Linux see Run Linux GUI apps on the Windows Subsystem for Linux.

1. Open PowerShell.

Set WSL Ubuntu 22.04 LTS version to 2 using the following command: wsl --set-version Ubuntu-22.04 2

2. Open Ubuntu 22.04 LTS and change directory using the following command:

cd /usr/share/ETAS/LiMa/x32/bin

- Open LiMaQt.sh using the following command: LiMaQt.sh
- LiMa was opened.

0 /usr/s

| 1 | | ETAS License | Manager | | |
|-----|---------------|---------------------|---------|-----------------|-----|
| ile | License Tools | Device <u>H</u> elp | | | |
| All | ٣ | | | | |
| • | Product | Feature | Status | Feature Version | Sou |
| ٥ | VECU | VECU_BUILDER_RT | Not | 1.0 | Not |
| ٥ | VECU | VECU_BUILDER_GO | Not | 1.0 | Not |
| ٥ | VECU | VECU_BUILDER | Not | 1.0 | Not |
| | | | | | |

3.7.7 Activating the LiMa license

There are several possibilities to activate the license. For more information about ETAS license management, see the ETAS License Management FAQ or the ETAS License Manager help.

3.7.8 Uninstalling VECU-BUILDER on Ubuntu 22.04 LTS for WSL

- 1. Open Ubuntu 22.04 LTS command line interface.
- To uninstall, execute the following command: sudo apt remove vecu-builder
- Sou are asked if you want to continue uninstalling.
- 3. To continue uninstalling, enter <Y> and press <ENTER>.
- ⇒ The VECU-BUILDER package is removed.

3.8 Installed files and folders

VECU-BUILDER software

The default installation location is

```
C:/Program Files/ETAS/VECU-BUILDER/1.8.0 On Windows
```

or

/opt/etas/VECU-BUILDER/1.8.0 on **Ubuntu 22.04 LTS** and **Ubuntu 22.04 LTS on WSL**.

It is recommended not to alter the installation location.

An environment variable of VECUBUILDER_HOME points to this folder.

| << VECU-BUILDER > <version></version> | opt / etas / VECU-BUILDER / <version></version> | | |
|---------------------------------------|---|--|--|
| Name | Name | | |
| Ird_party | 3 rd_party | | |
| 📜 bin | in bin | | |
| 🦲 build | build | | |
| documentation | documentation | | |
| CreateWorkspace.bat | CreateWorkspace.sh | | |

Fig. 3-4: Installation content (left: Windows, right: Ubuntu 22.04 LTS)

This folder contains multiple subfolders and a one command/shell script:

- 3rd party: Contains the third party software of fmusim 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/CreateWorkspace.sh: Creates a new workspace. After executing, you will be guided through the process step by step.

VECU-BUILDER examples/templates

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

```
C:/ProgramData/ETAS/VECU-BUILDER/Examples_1.8.0 ON Windows 
or
```

/opt/etas/VECU-BUILDER/Examples_1.8.0 on Ubuntu 22.04 LTS and Ubuntu 22.04 LTS on WSL.

An environment variable of VECUBUILDER_EXAMPLES points to this folder. The following examples and templates are delivered along with the software:

- additional_scrpts_library

BCU (Body Control Unit) - only for Windows

- EventTriggerExample
- plugin_template_v1_FMl2
- plugin_template_v2_FMI3
- SimpleExample
- SimpleExample_Plugin_v1_FMl2
- SimpleExample_Plugin_v2_FMI3

| << VECU-BUILDER > Examples_ <version></version> | opt / etas / VECU-BUILDER / Examples_ <version></version> |
|---|---|
| Name | Name |
| additional_scripts_library | additional_Scripts_library |
| ECU BCU | EventTriggerExample |
| EventTriggerExample | plugin template v1 FMI2 |
| plugin_template_v1_FMI2 | |
| plugin_template_v2_FMI3 | plugin_template_v2_FMI3 |
| SimpleExample | SimpleExample |
| SimpleExample_Plugin_v1_FMI2 | SimpleExample_v2_FMI3 |
| SimpleExample_Plugin_v2_FMI3 | SimpleExample_v3_FMI3 |



VECU-BUILDER workspaces

The default folder is recommended as the location for all your workspaces, where you can create a dedicated subfolder for each workspace.

The default folder is created during the installation process under

C:/Users/Public/Documents/VECU-BUILDER_Workspaces ON **Windows** or

/opt/etas/VECU-BUILDER_Workspaces On Ubuntu 22.04 LTS and Ubuntu 22.04 LTS on WSL.

Access to artefacts in Windows

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



Fig. 3-6: Start Menu entries

4 VECU-BUILDER without admin credentials

You can use VECU-BUILDER without Admin credentials. If you lack admin credentials, you can use VECU-BUILDER as a portable version without needing an installation.

4.1 Use of portable version without admin credentials on Windows

Prerequisites:

- Ensure that ETAS License Manager (LiMa) is installed and a valid license is available.

To use VECU-BUILDER portable version:

- Extract VECU-BUILDER_portable_1.8.0_windows.zip to some folder, such as C:/PortableTools/.
- 2. Open a new command prompt.
- 3. Set environment variables using the following commands:
 - SET VECUBUILDER_EXAMPLES=<C:/PortableTools/>/VECU-BUILDER_portable_1.8.0_windows\\$LOCALAPPDATA\ETAS\VECU-BUILDER\EXAMPLES 1.8.0
 - SET VECUBUILDER_HOME=<C:/PortableTools/>VECU-BUILDER_ portable_1.8.0_windows\1.8.0
- 4. Log off and log in again.
- ➡ You can now create a new workspace using CreateWorkspace.bat under VECU-BUILDER_portable_1.8.0_windows\1.8.0.



For more information about Working with VECU-BUILDER and VECU-BUILDER Examples/Templates, see VECU-BUILDER software, VECU-BUILDER examples/templatesWorking with VECU-BUILDER and Exploring the examples/templates.

4.2 Use of portable version without admin credentials on Ubuntu 22.04 LTS

Prerequisites:

- Ensure that License Manager (LiMa) is installed and a valid license is available.
- To install LiMa on Ubuntu 22.04 LTS, see Installing License Manager (LiMa) on Ubuntu 22.04 LTS.

To use VECU-BUILDER portable version:

- 1. Navigate to the directory where the Debian Software Package file VECU-BUILDER_installer_1.8.0.deb is located.
- Extract the directory to some folder, such as /home/<your user->/PortableTools.

To extract the portable version, use the following command: dpkg-deb -R ./VECU-BUILDER_installer_1.8.0.deb /home/<y-

our user>/PortableTools

3. Navigate to the following path:

/etc/profile.d

- 4. Create a shellscript file named vecubuilder-conf.sh.
- 5. Enter the following content into the shellscript file:

```
#!/bin/bash
export VECUBUILDER_HOME=/home/<your user-
>/PortableTools/opt/etas/VECU-BUILDER/1.8.0/
```

- 6. Log out and log in again.
- Copy the content of /home/<your user>/PortableTools>/usr to any location in the home(~) folder.

i) Note

```
You can find SW and examples under /home/<your user-
>/PortableTools/opt/etas/VECU-BUILDER/.
```

⇒ You can now create a new workspace.

For more information about Working with VECU-BUILDER and VECU-BUILDER Examples/Templates, see VECU-BUILDER software, VECU-BUILDER examples/templates, Working with VECU-BUILDER and Exploring the examples/templates.

5 Working with VECU-BUILDER

The steps outlined in the upcoming chapters guide you in creating your first vECU based on the Simple example. This provides an ideal starting point for your journey into virtualization.

To become familiar with working in VECU-BUILDER, follow the path below:



Fig. 5-1: 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. 5-2: VECU-BUILDER stages

5.1 Creating a new workspace

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

i) Note

Workspaces are designed for parallel use.

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

To create a workspace using Command Line Interface (CLI) and Ubuntu 22.04 LTS CLI, see Controlling VECU-BUILDER.

5.1.1 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.

| 😥 Create new workspace | | | — | | × |
|--|----------|----------------|--------------|------|---|
| (C) 2020-2023 ETAS GmbH. All rights reserved. VECU-BUILDER <version></version> | | | | | Â |
| ###################################### | ***** | ####### ### | | | |
| [07:36:14] 1 of 3: Selecting the project folder | **** | *###### | | | |
| Please select a workspace folder for your project! | | | | | |
| Please select a workspace folder for your project! | | | | × | |
| \leftarrow \rightarrow \checkmark \uparrow \blacksquare « Public > Public Documents > VECU-BUILDER_Workspaces \checkmark \circlearrowright | <u>ب</u> | Search VEO | CU-BUILDER_V | Vork | |
| Organize 🔻 New folder | | | | ? | |
| Name | | | | ~ | |
| Folder: | | | | | |
| | Sele | ct Folder | Cance | l ai | |

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 vEcuConf.yaml configuration file opens in Notepad++.
- Per default, this is the configuration file of the Simple example.



- 3. Keep the configuration file as is and close the Notepad ++ software.
- Your new workspace is now created.

The process will automatically continue with the next stage.

5.1.2 Creating a workspace on Ubuntu 22.04 LTS

i) Note

In Ubuntu 22.04 LTS LTS the folder, that should be used as workspace, needs to exist before the workspace creation is proceeded.

 Navigate to the folder, where the CreateWorkspace.sh is located. The default path is

opt/etas/VECU-BUILDER/1.8.0.

- 2. Open a new terminal. VECU-BUILDER will use the editor found under /usr/bin/editor.
- 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.

| Ē | /opt/etas/VECU-BUILDER/ <version> Q =</version> | _ 0 | × |
|--|--|--------|------|
| ****** VECUBUI VECUBUI ****** | <pre>/opt/etas/VECU-BUILDER/<version> \$./CreateWor ************************************</version></pre> | kspace | s.sh |
| (C) 202 VECU-BU ##### Cre ###### [08:18: Please | 0-2023 ETAS GmbH. All rights reserved. ILDER <version> ####################################</version> | | |
| | Please select a workspace folder for your project! 8 | | |
| Directory: | /opt/etas/VECU-BUILDER_Workspaces/ | | |
| Selection: | /opt/etas/VECU-BUILDER_Workspaces/ | | |

- ⇒ The vEcuConf.yaml configuration file opens.
- Per default, this is the configuration file of the Simple example.



- 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.
5.2 Importing files and folders

During this stage, the sources defined in your vEcuConf.yaml file are copied into vEcu/imported folder in your workspace.



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 22.04 LTS.

After successful completion of this stage, a dialog opens. It asks whether you want to continue with the next stage Building the vECU or inspect the results of this stage.



Fig. 5-3: Proceed with vECU Build dialog or inspect the results (Windows)

| F | ~/opt/etas/VECU-BUILDER/ <version> $Q \equiv - \Box \times$</version> |
|---|---|
| (C) 2 VECU- ##### [09:4 [09:4 [09:4 eepro src" to [09:4 *** S Pleas | 020-2024 ETAS GmbH. All rights reserved. BUILDER <version> ####################################</version> |
| | 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. Yes No Yes No |

Fig. 5-4: Proceed with vECU Build dialog or inspect the results (Ubuntu 22.04 LTS)

- 1. Click Yes.
- ⇒ Your new workspace is now created.

The process will continue with the next stage.

5.3 Building the vECU

During this stage, the sources imported into your workspace are compiled. Also they are 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 are 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 22.04 LTS.



Fig. 5-5: Building vECU completed (Windows)

| F | | /opt/etas/VE | ECU-BUILDER/ <version></version> | Q | | ۵ | × |
|---|---|--|--|---------|-----|---|---|
| (C) 2020-2023 ETAS GmbH. VECU-BUILDER <version> ####################################</version> | All rights | 5 reserved. | | #### | | | |
| ### Duituing sources of #################################### | <i>*************************************</i> | | ####################################### | #### | | | |
| [08:21:57] 1 of 4: Readi [08:21:57] 2 of 4: Creat [08:21:57] 3 of 4: Runni - No | ng config: ing Visual ng scripts script defi | vEcuConf.yaml Studio Code deb triggered throu ned in the vEcu | ug configuration gh "before_build Conf.vaml file | _source | es" | | |
| [08:21:57] 4 of 4: Compi - lit | ling and li SimpleExamp | nking ble.so (Linux 64 | , bit) | | | | |
| Forwarding to building t *** SUCCESS *** Please wait 4 seconds. | he FMUs upo | on completion | | | | | |



The process will automatically continue with the next stage.

If the process will not automatically continue with the next stage and error messages are displayed, see Building sources failed for more information.

5.4 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) 2020-2024 ETAS GmbH. All rights reserved. | ^ |
|--|---|
| VECU-BUILDER <version></version> | |
| | |
| ### Building FMU ### | |
| | |
| [12:47:55] 1 of 7: Reading config: vEcuConf.yaml | |
| <pre>[12:47:56] 2 of 7: Running scripts triggered through "before_build_fmus"</pre> | |
| - No script defined in the vEcuConf.yaml file | |
| [12:47:56] 3 of 7: Building inputs, outputs, parameters, tasks | |
| [12:47:59] 4 of 7: Patching a2l file | |
| - No a2l file defined in the vEcuConf.yaml file | |
| [12:47:59] 5 of 7: Processing initial data | |
| C:/ProgramData/ETAS/VECU-BUILDER/Examples_<version>/SimpleExample/init</version> | |
| /InitialData.VarVal | |
| [12:48:00] 6 of 7: Building fmu archives | |
| [12:48:03] 7 of 7: Running scripts triggered through "after_build_fmus" | |
| - No script defined in the vEcuConf.yaml file | |
| *** SUCCESS *** | |
| Please wait 4 seconds. | |
| | |
| | |

Fig. 5-7: Building FMU completed (Windows)

| bash – | | × |
|--|------|------|
| (C) 2020–2024 ETAS GmbH. All rights reserved. | | |
| VECU-BUILDER <version></version> | | |
| *************************************** | | |
| ### Building FMU ### | | |
| *************************************** | | |
| [16:39:50] 1 of 7: Reading config: vEcuConf.yaml | | |
| [16:39:51] 2 of 7: Running scripts triggered through "before_build_fmus" | | |
| - No script defined in the vEcuConf.yaml file | | |
| [16:39:51] 3 of 7: Building inputs, outputs, parameters, tasks | | |
| [16:39:53] 4 of 7: Patching a2l file | | |
| - No a2l file defined in the vEcuConf.yaml file | | |
| [16:39:53] 5 of 7: Processing initial data | | |
| - /opt/etas/VECU-BUILDER/Examples_version>/SimpleExamp | le/i | init |
| /InitialData.VarVal | | |
| [16:39:53] 6 of 7: Building fmu archives | | |
| [16:39:54] 7 of 7: Running scripts triggered through "after_build_fmus" | | |
| — No script defined in the vEcuConf.yaml file | | |
| *** SUCCESS *** | | |
| Please wait 4 seconds. | | |
| | | |

Fig. 5-8: Building FMU completed (Ubuntu 22.04 LTS)

5.5 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 content for Windows, Ubuntu 22.04 LTS and Ubuntu 22.04 LTS on WSL.



Fig. 5-9: Workspace contents

The content of the workspace consists of several artefacts:

- vscode folder:
 - launch.json file for vECU debugging in VS Code
- build folder:
 - additional_scripts folder: Location for your project specific additional scripts
 - log folder: Log files from executed stages
 - scripts folder: Batch 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 call fmusim and inspect the vECU outputs.
- 3b_StartDebugger.bat/3b_StartDebugger.sh
 File to call MSVC or VS Code as debugger.
- 3c_ShowSymbolDetails.bat/3c_ShowSymbolDetails.sh
 File to call Notepad++/new Terminal 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 information, see Simple example.
- SimpleExample_debug.fmu
 Debug version of your vECU, for information, see Simple example.
- vEcuConf.yaml
 YAML configuration file, for more information, see Configuration.

5.6 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:

| | - 1 | • |
|---|-----|--|
| | 2 | # The version of the .yaml file schema # |
| | 3 | |
| B | 4 | -version: |
| - | 5 | |
| | 6 | # |
| | 7 | # |
| | 8 | # |
| | 9 | # |
| | 10 | # |
| | 11 | # |
| | 12 | # |
| | 13 | # |
| | 14 | Text10 mode fac111 controls |

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_compiled:

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. VECU-BUILDER sets up the inputs, outputs and tasks, patches the A2L file, sets up the XCP slave port, etc.

fmu_name

Enter the name of your vECU.

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

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 vEcu/imported folder in your workspace.

You can use environment variables like this:

"\${VECUBUILDER_EXAMPLES}\SimpleExample\eeprom_data.txt"

- link_into_project

To create a symbolic link; use link_into_project.

A symbolic link is a reference to a file or directory in a file system. A symbolic link is created to the default folder vEcu/imported from another directory. This can be either the source directory or optionally a destination subdirectory in vEcu/imported folder. You can also create nested folders.

source: Source of the folder to be linked.

destination: Name of the linked folder or name of the linked nested folder.

You can use environment variables like this:

\${SomeEnvironmentVariable}

If you do not state a destination, the source will be linked in v ECU/im-ported/NameOfSource.

For more information about link_into_project, see Usage of link_into_project.

- import_external_compiled_vecu

Only needed if you selected import_compiled 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/SOname and the path for updates:

dll_so_name: The name of the DLL/SO. There must exist a corresponding PDB file with the same filename.

get_updates_from: If VECU-BUILDER can find a DLL/SO and the PDB file in this folder, it will update the imported DLL/SO.

You can use environment variables like this: "\${SystemDrive}/Sandbox".

- additional_resources

You can use additional resources to resolve dependencies by integrating DLL/SO libraries that your software relies on into the build and execution process. You can reference the necessary files and folders for the vECU assembly to function properly. When adding resources, copy folders recursively (including their contents) to the root of the resources folder, while copying files directly to the root of the resources folder. There is no limit to the number of additional resources you can inc

i) Note

additional_resources does not support wildcards.

Specify all additional resources that are to be included in the FMU. Additional resources will be copied to the resources folder of the FMU during the Building FMU stage i.e.

\${VECUBUILDER_WORKSPACE}/vECU/imported/additional_

```
DLLs/UsedByVECU.dll for Windows
```

or

\${VECUBUILDER_WORKSPACE}/vECU/imported/additional_

DLLs/UsedByVECU.so for Ubuntu 22.04 LTS.

You can use and include plug-ins as additional resources the same way. For more information, see Template for plug-in V1 (FMI2) and Template for plug-in V2 (FMI3).

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 32-bit or 64-bit architecture, you need to set the attribute to the same.

i) Note

For Ubuntu 22.04 LTS only 64-bit is supported.

_ 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 information, see A2L file patching.

i) Note

You can only use a socket (IP address + port + protocol) for the XCP connection between INCA and XCP slave once.

If a port is busy, you must define another port in the YAML file.

- operating_system

Enter the operating system. Currently Windows and Ubuntu 22.04 LTS supported.

- build_tool

Set up the build tool to be used for your build.

Built tool differs between Windows and Ubuntu 22.04 LTS.

Windows

i Note

VECU-BUILDER configures the build tool for the underlying CMake.

If you select Visual Studio, a Visual Studio solution is generated.

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

These artefacts are stored in vECU/CMake folder in your workspace.

 path_to_mingw: If you define a specific MinGW version, CMake builds the sources using that MinGW version.

Ubuntu 22.04 LTS

You can select Unix Makefiles.

cmake_generator_toolset

Define which toolset should be used by CMake during the build process. For more information, see CMAKE_GENERATOR_TOOLSET.

- inputs, outputs, parameters, locals

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

Inputs, outputs, parameters, and locals refer to the causality of the FMI. You can use wildcards (* and ?) in your expressions. You can add arrays and structures using myArray*. If your wildcard expression breaks the YAML compatibility, enclose it in single apostrophes.

EXAMPLE

*a finds all symbols ending with an a.

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

i) Note

Variables of type enumeration will be interpreted as integers in 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.

- initial_data

To define the initial values of calibration variables, enter the path for source and target destination

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: Location to obtain 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.

encoding: Character encoding for DCM files (optional) specifies the standard used for text content within the file. If the encoding is not defined, the default UTF-8 encoding is used.

Supported formats:

<code>VarVal</code>: List of pairs separated by one space, where the <code>lhs</code> refers to the C variable and the <code>rhs</code> to the value.

DCM: Format containing ASAP2 labels and their values in physical form which are processed according to information in A2L file.

For more information, see InitialData functionality.

- eeprom

Specify the eeprom simulation attributes.

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 path name. When the vECU is initializing, this file is copied to 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.

 ${\tt c_variables}$: The C variable names that store the eeprom data.

Supported format:

TXT: A line starting with *#* denotes a comment. All other lines contain the data stream to be flashed to the C variables. The order of the data stream lines corresponds to the order of the C variables listed. A data stream consists of bytes in HEX format, with each byte separated by a space.

EXAMPLE

01 02 ee 4f.

In the default YAML file the sync is commented out.

For more information about eeprom, see eeprom functionality.

- arxml_tasks

To retrieve tasks from ARXML files in vECU/imported folder, use arxml_tasks. These tasks will be added to the tasks list.

Supported events:

- INIT-EVENT
- TIMING-EVENT

Per default, arxml_tasks are disabled. To enable arxml_tasks and place your ARXML files in the imported folder, use enabled. For more information about arxml_tasks, see ARXML-defined tasks.

tasks

)Note

Task functions must have no arguments.

Define the tasks that are to be executed and their attributes. To simulate the microcontroller behavior with its periodically executed functions of your software, define the functions as tasks. You can define a task only once, duplicated functions will be ignored.

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

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

initial: Functions are called from fmi2DoStep before cyclic tasks.

cyclic: Functions are called from fmi2Dostep before terminate tasks. For more information about task scheduling with cyclic task trigger, see Task scheduling with task trigger defined as cyclic.

i) Note

The period of a cyclic task must not be less than 1 ns.

terminate: Functions are called from fmi2Terminate.

fmi2_enter_init: Functions are called from fmi2EnterInitializationMode.

fmi2_exit_init: Functions are called from fmi2ExitInitializationMode.

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.

i) Note

If two functions run at the same time, the one with the lower priority runs first.

max calls: <number>, -1 means infinite, 0 means no call.

trigger_function: The function is written in Multiply.c. You can use trigger_function only, if trigger is set to event. The trigger function predicts when the next event might occur and returns if the next events needs to be triggered. You can find the defined arguments of the trigger function in Multiply.c trigger_inputs: A list of additional inputs that refer to variables accessible via symbol details. Trigger inputs must be included in symbolDetails.txt.

For more information, see EventTrigger example.

- 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. Also you do not need 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.

i) Note

Sometimes redirect_function_calls does not work as expected. For more information, see additional_compile_flags in this chapter and Redirecting function calls did not work as expected.

- build_include_filters, build_exclude_filters

Only usable if you selected build_sources as build_mode. You can select files and/or folders to 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.

Only the sources listed in a file will be passed to the compiler from the given sources defined by build_include_filters and build_exclude_filters.

If you did not configured assembly list files, all sources are compiled.

additional_include_directories

Only usable if you selected build_sources as build_mode.

Specify the path of the directory to be added with additional_ include_directories. This directory will be included in the list of directories searched for include files. Additional include directories are passed to the preprocessor.

Wildcards * and ? are allowed. The environment variable \${VECUBUILDER_ WORKSPACE} points to the workspace.

- additional_defines

Only usable if you selected build_sources as build_mode.

Specify the preprocessor macro definitions you want to add. These definitions are passed to the preprocessor. This is useful if you need to set or unset some of the definitions to adapt them to the new Windows target.

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

additional_compile_flags

Only usable if you selected build_sources as build_mode.additional_compile_flags will be applied to C and C++.

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

The flags are written into the CMakeLists.

For more information about compling, see MSVC compiler options or gcc compiler options.

A successful use of redirect_function_calls depends on additional_compile_flags. Only if you set additional_compile_flags correctly, redirect_function_calls will work.

To prevent the GNU compiler from using incompatible optimizations when redirect_function_calls feature is enabled, optimizations are disabled by using the following flags:

- '-00' for gcc

For more information, see Options that control optimization and Redirecting function calls did not work as expected.

- additional_c_compile_flags

The same prerequisites as for "additional_compile_flags" above must be met. additional_c_compile_flags will be applied only to C.

A successful use of redirect_function_calls depends on additional_c_compile_flags. Only if additional_c_compile_flags is set correctly, redirect_function_calls will work.

To prevent the GNU compiler from using incompatible optimizations when redirect_function_calls feature is enabled, optimizations are disabled by using the following flags:

- '-fhosted'

- additional_cxx_compile_flags

The same prerequisites as for additional_compile_flags must be met. additional_cxx_compile_flags will be applied only to C++.

A successful use of redirect_function_calls depends on additional_cxx_compile_flags. Only if you set additional_cxx_compile_flags correctly, redirect_function_calls will work.

To prevent the GNU compiler from using incompatible optimizations when redirect_function_calls feature is enabled, optimizations are disabled by using the following flags:

- '-fpermissive'

- additional_static_libraries

 $Only \ usable \ if \ you \ selected \ {\tt build_sources} \ for \ {\tt build_mode}.$

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}

You can configure and modify these variables in one location and you can access them 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.

- import_additional_scripts

To include additional files, use import_additional_scripts.

With import_additional_scripts, additional scripts are imported at the beginning of the import stage into the following target folder: build/additional_scripts

You can define paths to single files and to folders. If you define a path to a folder, all files and subfolders in that folder are imported.

You can use environment variables like this: \${SomeEn-vironmentVariable.

- '\${VECUBUILDER_EXAMPLES}/additional_scripts_library/'

- additional_scripts

Define additional scripts to be executed at various phases of the import and/or the build stage.

Use batch files on Windows/shell scripts on Ubuntu to simplify the execution of your project-specific scripts, such as those implemented in Python or Perl, if they are executable on your system. You can use these scripts for tasks such as file manipulation, adding files to the FMU archive, parsing etc..

command:The script to be executed by the OS, the default search path is
\${VECUBUILDER_WORKSPACE}/build/additional_scripts/(utf-8
only).

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

- before_import
- after_import

- before_build_sources
- before_build_fmus
- after_build_fmus

priority: Specify the priority at which your script should be executed. A lower number indicates a higher priority, with the default value set to 1.

For more information about additional_scripts, see Example of additional scripts - A2L characteristics as parameters.

patch_a2l_file

You require an A2L file to connect an MCD software such as ETAS INCA to the running vECU. The A2L file needs to be located in the folder vEcu/imported.

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

symbol_name_mapping

When you use A2L or DCM files, ASAP2 labels might differ from the symbol names. If both, DCM and A2L, files are provided, then for each DCM entry, an A2L entry of the identical ASAP2 label name must exist.

based_on_csv:

CSV file, where lhs is the symbol name and rhs is the ASAP2 label. The CSV file must follow the following format: SYMBOL_name; ASAP2_label. You need to use a semicolon as a delimiter.

When you use a CSV file, you can use simple string search & replace only. Ensure that there are no header or any comments in the CSV file and every line is treated as a data record.

```
i Note
```

VECU-BUILDER removes all leading and trailing spaces before and after the first character.

EXAMPLE

```
my symbol ; ASAP2 LABEL 5
```

This is a valid entry and will be treated as defined below:

```
my_symbol;ASAP2_LABEL_5
```

based_on_adx:

If a mapping between ASAP2 labels and symbols is available in an ADX file format (a proprietary format of Bosch, used exclusively within Bosch projects), you can include this file in the build to apply the mappings. When using the ADX file format for mappings, a simple string search and replace is applied.

i) Note

The content of an ADX file will be processed as is, no interpretation or validation will be performed by VECU-BUILDER. The file is assumed to be complete and correct.

based on symbol_link (only on Windows):

In A2L files, SYMBOL_LINK information refers to the linkage between symbols or variables defined in the file. It provides information about how different symbols are related or connected to each other.

based_on_assignments:

If the right side includes a dollarsign \$ (like in a reference to a group, e.g. (\$1), then a regular expression search & substitute is applied. Else a simple string search and replace is applied.

One such regular expression allows to map multiple names at once. To see an example, see the following table.

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

The mappings can be verified by examining the JSON files appended to the debug FMU, located in resources \mappings folder.

VECU-BUILDER will update the memory addresses of entries in the provided A2L file. The original A2L file is renamed by appending .bak to its name. For more information, 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 information, see Debugging vECU.

- additional_link_flags

Only usable if you selected build_sources as build_mode.

Specify how the linker should work. You need to write each individual flag in a separate line and needs to be put in single apostrophes, i.e. /DEBUG.

The flags are written into the CMakeLists.txt.

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

simple_file_modifications

Specify file modifications that must be applied to files imported in v ECU/im-ported 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 must be modified.

trigger: Specify when the modification must 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_below: 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 youuse simple_file_modifications, consider the following procedure to ensure that 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.

 \rightarrow 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.

i) Note

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

- include_symbol_details

The use of plug-ins may require that the release FMU contains symbol information.

EXAMPLE

If you want to change the cycle time of "task_10ms" using a plug-in, then the symbol name "task_10ms" must be disclosed in the release vECU.

For more information about release FMU, see Difference between debug and release vECUs and Keeping symbol information in a release FMU.

fmi_<type>: disabled (default) or enabled. When enabled, all symbol
details of that fmi type will be included.

symbol_names: All symbol details matching the regular expressions will be included.

— fmi

You can use the functional mockup interface. Default version is 2.0. The use of version 3.0 is also possible. Follow the convention below: #fmi: '2.0'

For more information about FMI, see FMI standard.

6 Exploring the examples/templates

This chapter contains details on examples/templates that are designed to help you familiarize with the features of VECU-BUILDER.

6.1 Simple example

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

6.1.1 fmusim

To conduct a quick smoke test of the created vECU, fmusim is delivered along with VECU-BUILDER. You can invoke this software via the 3a_CheckFMU.bat on **Windows** or 3a_CheckFMU.sh on **Ubuntu 22.04 LTS**. Execute this file to run the release vECU. Alternatively, you can drag-and-drop the debug vECU into this batch/shell script file to run the debug vECU.

fmusim opens a terminal and prints the simulation outputs.

```
fmi2DoStep(currentCommunicationPoint=9.890000000000001, communicationStepSize=0.01, noSetF
MUStatePriorToCurrentPoint=1) -> OK
Fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
mi2DoStep(currentCommunicationPoint=9.9, communicationStepSize=0.01, noSetFMUStatePriorTo
CurrentPoint=1) -> OK
Fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
fmi2DoStep(currentCommunicationPoint=9.91, communicationStepSize=0.01, noSetFMUStatePriorT
oCurrentPoint=1) -> OK
Fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
fmi2DoStep(currentCommunicationPoint=9.92, communicationStepSize=0.01, noSetFMUStatePriorT
CurrentPoint=1) -> OK
fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
fmi2DoStep(currentCommunicationPoint=9.93, communicationStepSize=0.01, noSetFMUStatePriorT
CurrentPoint=1) -> OK
Fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
Fmi2DoStep(currentCommunicationPoint=9.94, communicationStepSize=0.01, noSetFMUStatePriorT
oCurrentPoint=1) -> OK
Fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
mi2DoStep(currentCommunicationPoint=9.9500000000000001, communicationStepSize=0.01, noSetF
MUStatePriorToCurrentPoint=1) -> OK
Fmi2GetReal(vr={3}, nvr=1, value={8}) -> OK
```

Fig. 6-1: fmusim output

| Result.csv | 1 | time,"product" | |
|-----------------------|---|----------------|--|
| Completiversple. Inc. | 2 | 0 | |
| | 3 | 0.01,2 | |
| | 4 | 0.02,2 | |
| | 5 | 0.03,2 | |
| | 6 | 0.04,2 | |
| | 7 | 0.05,2 | |

Fig. 6-2: Cutout of result.csv

An FMU, that is built by VECU-BUILDER will set the environment variable VECUBUILDER_FMURESOURCES. The environment variable is set for the process that runs the FMU. It is not set on system-level or user-level.

This environment variable stores the absolute path to the resources folder of the FMU. You can find the environment variable when inspecting the process properties in a process monitor software.

6.1.2 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 explore their contents and differences.

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). To debug and step through your code, use the debug vECU.



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

Fig. 6-3: Comparison of debug and release vECU (GCC compiler)

The release vECU exclusively includes address information, while the debug vECU contains both variables and function names. The release vECU protects the IP contained in the vECU and does not contain symbol information. To share your vECU, use the release vECU.



Fig. 6-4: Comparison of resources.txt

6.1.2.1 Keeping symbol information in a release FMU

When using a plug-in in your vECU, variables and functions are accessed by name. This is possible in a debug FMU, but not in a release FMU.

Searching for symbols in the plug-in it is done by name. In the release FMU all symbols names are replaced by their memory addresses. When FMU Runner runs in release mode, it accesses the symbols directly by addresses instead of symbol names. As a result, SymbolDetails file like RawSymbolDetails.txt file has to be kept in release.fmu file

j) Note

The release FMU does not contain the PDB/DWARF file. Thus, debugging the release FMU is not possible. RawSymboldetails.txt is a subset of SymbolDetails.txt and is used by the plug-in.

For more information, see include_symbol_details in configuration chapter. For more information about the plug-in Feature, see Template for plug-in V1 (FMI2) and "Template for plug-in V2 (FMI3)" on page 88.

6.1.3 InitialData functionality

Usually, software function and its data are separated. While the logic of the software function is defined in the source files, the data is stored in separate files in various formats. Common formats for such calibration data are DCM and CDF.

VECU-BUILDER provides support of DCM format. For more information about DCM Format, see DCM file format. A DCM file stores the data in their physical form which typically need to be processed into ECU-internal form. This processing is done based on COMPU_METHOD and RECORD_LAYOUT entries in an A2L file. You need to provide the A2L file in the patch_a21_file attribute in the YAML file.

In case the Symbols do not match the ASAP2 labels (entries in the DCM and A2L files), you can resolve this by applying mappings. These mappings are then used to map ASAP2 labels to their respective symbols and can be defined in one of these three ways:

- direct definition in the YAML file making use of regular expressions
- via ADX file
- via CSV file
- via symbol_link (only available for Windows)

For more information about these options, see symbol_name_mapping section in configuration chapter.

You also can define the initial data in the VARVAL format. These initial data will not be processed based on entries in the A2L file neither will any mapping be applied. Thus the VARVAL file must contain the symbols and the ECU-internal values.

Simple example contains sample files of both supported formats. You can find the files in this folder:

```
C:/ProgramData/ETAS/VECU-BUILDER/Examples_1.8.0/Sim-
```

pleExample/init ON Windows

or

/opt/etas/VECU-BUILDER/Examples_1.8.0 On Ubuntu 22.04 LTS.

The vEcuConf.yaml file is preconfigured and uses the InitialData.VarVal.

To experiment with VARVAL functionality

1. Open Multiply.c file located within your workspace in vECU/imported folder.

The Multiply.c file contains the variable definitions. The variables factor1 and factor2 serve as the inputs, with assigned values of 1 and 2. The variable product serves as the output and is calculated as the product of factor1 and factor2.

As InitialData.VarVal file is already activated in the YAML file, it is already used in the default Simple Example vECU.

| 📙 Multiply | .c 🗙 | | | | | 📙 Initial | Data.VarVal | × |
|------------|------|--------|---------|---|------|-----------|-------------|---|
| 1 | C | louble | factor1 | = | 1.0; | 1 | factor1 | 1 |
| 2 | c | louble | factor2 | = | 2.0; | 2 | factor2 | 2 |
| 3 | c | louble | product | = | 0.0; | | | |

- 2. Close the source file and navigate back to the workspace.
- 3. Check the output using

 $3a_CheckFMU.bat On Windows$

or

- 3a_CheckFMU.sh On Ubuntu 22.04 LTS, described in fmusim.
- ➡ The output for SimpleExample is 2.
- 4. Change the value for factor 1 to 4.

| 📙 InitialData.VarVal 🔀 | | | | | | |
|------------------------|---------|---|--|--|--|--|
| 1 | factor1 | 4 | | | | |
| 2 | factor2 | 2 | | | | |

- 5. Save the change.
- 6. Rebuild the vECU using

2 Build.bat in the workspace on Windows

or

- 2_Build.sh in the workspace on Ubuntu 22.04 LTS.
- 7. To show the changed output in the FMU, execute

 $3a_CheckFMU.bat On Windows$

or

- 3a_CheckFMU.sh ON Ubuntu 22.04 LTS.
- ⇒ The new output is now 8.

| | А | | В |
|----|-------------|-------|---------|
| 1 | time,"produ | ct" | |
| 2 | | 0 | |
| 3 | 0.01,8 | | |
| 4 | 0.02,8 | | |
| 5 | 0.03,8 | | |
| 6 | 0.04,8 | | |
| 7 | 0.05,8 | | |
| 8 | 0.06,8 | | |
| 9 | 0.07000000 | 0000 | 00001,8 |
| 10 | 0.08,8 | | |
| 11 | 0.09,8 | | |
| 12 | 0.1,8 | | |
| 13 | 0.11,8 | | |
| 14 | 0.12,8 | | |
| 15 | 0.13,8 | | |
| | · · · · | esult | t (+) |

- \Rightarrow The initial data set in the VARVAL file are thus correctly used in the vECU.
- The sources in Multiply.c stay the same. The variables are overwritten at run time by the values of the InitialData.VarVal.

To experiment with intialData.dcm

- 1. Open the .InitialData.dcm in Examples/SimpleExample/init.
- 2. Change the value for factor 1 to 4.
- 3. Change the value for factor 2 to 4.
- 4. Save the changes.

```
FESTWERT factor1
LANGNAME ""
EINHEIT_W ""
WERT 4.0
END
FESTWERT factor2
LANGNAME ""
EINHEIT_W ""
WERT 4.0
```

- 5. Open the YAML file and navigate to the initial_data section.
- 6. Uncomment the source and destination for InitialData.VarVal.
- 7. Comment the source and destination for InitialData.dcm.

```
initial_data:
#- source: '${VECUBUILDER_EXAMPLES}\SimpleExample\init\InitialData.VarVal'
# destination: 'init/InitialData.VarVal'
- source: '${VECUBUILDER_EXAMPLES}\SimpleExample\init\InitialData.dcm'
destination: 'init/InitialData.dcm'
```

- 8. Save the changes.
- 9. Rebuild the vECU using
 - 2_Build.bat in the workspace on Windows

or

- 2_Build.sh in the workspace on Ubuntu 22.04 LTS.
- 10. To show the changed output in the FMU, execute

3a_CheckFMU.bat On Windows

or

3a_CheckFMU.sh On Ubuntu 22.04 LTS.

⇒ The new output is now 16.

| 5 | | А | | | В |
|---|----|-----------|------|-------|-------|
| 7 | 1 | time,"pro | duct | .11 | |
| 3 | 2 | | 0 | | |
| | 3 | 0.01,16 | | | |
| 1 | 4 | 0.02,16 | | | |
| 2 | 5 | 0.03,16 | | | |
| 3 | 6 | 0.04,16 | | | |
| 4 | 7 | 0.05,16 | | | |
| 5 | 8 | 0.06,16 | | | |
| 5 | 9 | 0.070000 | 0000 | 00000 | 01,16 |
| 7 | 10 | 0.08,16 | | | |
| 3 | 11 | 0.09,16 | | | |
| 9 | 12 | 0.1,16 | | | |
| 2 | 13 | 0.11,16 | | | |
| | 14 | 0.12,16 | | | |
| 1 | 15 | 0.13,16 | | | |
| ŋ | | | res | ult | (+) |

- \Rightarrow The initial data set in the DCM file are thus correctly used in the vECU.
- The sources in Multiply.c stay the same. The variables are overwritten at run time by the values of the InitialData.dcm.

()Note

You can define several files and formats. If one variable is set in multiple files, the value of the last file is used.

For release vECU all initial data is merged into MergedInitialData.VarVal. This VARVAL file protects the IP. Release and debug vECU behave the same. To get the different folder structures, see Difference between debug and release vECUs.

6.1.4 eeprom functionality

During vECU initialization, the EEPROM data is loaded from a file to RAM. The data is saved to the file before running terminate tasks and when unloading the vECU. This can be utilized to simulate a soft reset behavior, ensuring that EEPROM-stored data is preserved and not lost once the vECU simulation terminates. A typical usage of this feature is the storage of total mileage information in the ESP controller.

1. Open vEcuConf.yaml file of SimpleExample and navigate to eeprom section.

The eeprom_data.txt file is initially copied from VECUBUILDER_ EXAMPLES/SimpleExample/src to the workspace in vECU/imported during the import process with standard configuration. During SimpleExample.fmu build, eeprom_data.txt file is integrated into the FMU as 1.txt in resources/eeprom folder. This happens because the optional destination attribute is enabled. You can change the destination path and file name accordingly. This file serves as the working copy.

| vECU > imported | SimpleExample.fmu > resources > eeprom |
|-----------------|--|
| Name | Name |
| sources | 1.txt |
| eeprom_data.txt | |
| | |

| / vECU / imported | SimpleExample.fmu / resources / eeprom | | |
|-------------------|--|--|--|
| Name | Name | | |
| 🌄 sources | bources | | |
| eeprom_data.txt | 1.txt | | |

(i) Note

If destination attribute is deactivated, eeprom_data.txt is integrated into the FMU (release and debug FMu) in resources folder.

2. Open eeprom_data.txt in imported folder and check the content.



 $\tt eeprom_data.txt$ must include the relevant data in expected HEX format.

3. Go back to YAML file.

In the standard configuration, the following **c_variables** are used:

- 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.

 $eeprom_data.txt$ contains the data stream that should be used for the c_variables.



matches with the size defined in SymbolDetails.txt.

To open SymbolDetails.txt, run

```
3c ShowSymbolDetails.bat On Windows
```

or

```
3c ShowSymbolDetails.sh On Ubuntu 22.04 LTS.
```

i.e. eeprom_block_b has a size of 8 bytes and comprises eeprom_block_ b.last_product which also has a size of 8 bytes.

| 0x000000000008140;8 0x000000000008140;8 | ;User-Defined Type;0 ;float64 ;0 | ;0 ;0 | ; ;0 | ;eeprom_block_b ;eeprom_block_b.last_product |
|--|---|----------|---------|---|
| # eeprom block b 00 00 00 00 00 00 | 00 40 | | | |
| | 8 Bytes | | | |

5. Delete the comment under sync and use the following:

```
sync:'C:/TEMP/eeprom data.txt'On Windows
```

or

```
sync: '//localhost/c$/TEMP/eeprom_data.txt' on Ubuntu 22.04
LTS.
```

()Note

Data from the imported eeprom file is used as initial data for the first simulation. After this step, the data will be always written back to the sync path at the end of each simulation and used by the next one.

- If the file is not existing in sync location, it will be created.
- If the file is already existing in sync location, this file will be taken by the first simulation run.

- 6. Save the changes.
- 7. Rebuild the workspace using

```
2_Build.bat in the workspace on Windows
```

or

2_Build.sh in the workspace on Ubuntu 22.04 LTS.

8. To start the simulation, execute

3a_CheckFMU.bat ON Windows

or

3a CheckFMU.sh On Ubuntu 22.04 LTS.

9. Navigate to

C:/TEMP on Windows

or

//localhost/c\$/TEMP on Ubuntu 22.04 LTS.

⇒ eeprom_data.txt was added to sync location.

```
# This is a comment
# eeprom block a
20 4e 00 00 00 00 00 00 00 02 00
# eeprom block b
00 00 00 00 00 00 00 40
```

6.1.5 Usage of link_into_project

You can link folders into your project. When you copy a file or folder, its contents are duplicated. Changes made to one copy do not affect the other.

In contrast, creating a link generates a file or folder that references the same content. Changes to one linked item are reflected in both. A link acts as an additional name for the same file or folder. It exists only once on the file system and occupies space just once. This method is useful for conserving disk space.

To use link_into_project

- 1. Open vEcuConf.yaml file of SimpleExample.
- 2. Navigate to link_into_project section.

With the standard configuration, the content of VECUBUILDER_ EXAMPLES/SimpleExample/src was linked into imported/sources.

| vECU > imported | imported > sources |
|------------------------------------|---|
| Name | Name |
| <pre>sources eeprom_data.txt</pre> | Multiply.cMultiply.h |
| / vECU / imported | / vECU / sources |
| Name | Name |
| sources | c Multiply.c |
| eeprom_data.txt | h Multiply.h |

- 3. To add a new linked folder, add the respective source and destination path.
- 4. Save your changes.
- 5. Import the files and folders using

1_Import.bat in the workspace on Windows

or

1_Import.sh in the workspace on Ubuntu 22.04 LTS.

 As there are already sources in the workspace, they will be overwritten. Agree to the deletion by clicking Yes. The files and folders are imported.

- 7. Proceed as described in Creating a new workspace.
 - vECU > imported... / vECU / importedNameNameNewFolderImportedsourcesImportedeeprom_data.txteeprom_data.txt
- ⇒ The folder is linked into vEcu/imported.

6.1.6 ARXML-defined tasks

To retrieve tasks from ARXML files, you can use <code>arxml_tasks</code> in the YAML file. To retrieve tasks from ARXML files, enable <code>arxml_tasks</code> and ensure the ARXML files are located in the <code>vECU/imported</code> folder. VECU-BUILDER will read all the ARXML files and will take all the timing and init tasks.

To use arxml_tasks by using import_into_project

- 1. Open vEcuConf.yaml file in your workspace.
- 2. Navigate to arxml_tasks section.
- 3. Uncommentarxml_tasks.
- 4. Change the value to enabled.
- 5. Navigate to import_into_project section.
- 6. Enter the paths to the ARXML files or folders containing the ARXML files to be imported.
- 7. Save your changes.
- 8. Rebuild the workspace using
 - 1_Import.bat in the workspace on Windows

or

2 Import.sh in the workspace on Ubuntu 22.04 LTS.

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

⇒ arxml_tasks.json file was created in the build folder.

To use arxml_tasks by manually copying ARXML files

- 1. Open vEcuConf.yaml file in your workspace.
- 2. Navigate to arxml tasks section.
- 3. Uncommentarxml_tasks.
- 4. Change the value to enabled.
- 5. Save your changes.
- 6. Place your ARXML files into vECU/imported folder.
- 7. Rebuild the workspace using

2_Build.bat in the workspace on Windows

or

2_Build.sh in the workspace on Ubuntu 22.04 LTS.

⇒ arxml tasks.json file was created in the build folder.

6.2 BCU example (only available for Windows)

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_1.8.0/BCU ON **Win-dows**.

A2L file patching is enabled In the YAML file.

- 2. Continue the process as described in Working with VECU-BUILDER.
- When the A2L file patching is enabled, a HEX file is generated during the build process and is available in the workspace after the build. For more information see patch_a2l_file and HEX file generation.

i Note

The HEX file will only be part of the workspace if the A2L file patching is activated.

- .vscode
- build
- vECU
- 1_Import.bat
- 🧕 2_Build.bat
- 3a_CheckFMU.bat
- 3b_StartDebugger.bat
- 3c_ShowSymbolDetails.bat
- 3d_RemoveGoLicense.bat
- BCU.fmu
- BCU.hex
- BCU_debug.fmu
- \min vEcuConf.yaml

6.2.1 Show symbol information

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

1. To see the details, run the command:

3c_ShowSymbolDetails.bat On Windows.

A text editor window 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 | e Settings Tools Macro | Run TextFX | (Plugins | Window ? | |
| ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; | | | | | | |
| 🖶 SymbolDetails 5tt 🔀 | | | | | | |
| 1 | 217 symbol(s) found, 1 symb | ool(s) with non-un | ique name | (s). | | |
| 2 | Address ;Size | ;Type | ;BF-0 | fs;BF-1 | Len;Value | ;Symbol Name |
| 3 | 0x000000000001330;0 | ;Function | ;0 | ;0 | ; | ;add_factor1_factor2_sum |
| 4 | 0x000000000001360;0 | ;Function | ;0 | ;0 | ; | ;multiply_factor1_factor2_product |
| 5 | 0x000000000001390;0 | ;Function | ;0 | ;0 | ; | ;task_10ms |
| 6 | 0x0000000000013d0;0 | ;Function | ;0 | ;0 | ; | ;terminate |
| 7 | 0x0000000000013e8;0 | ;Function | ;0 | ;0 | ; | ;dllmain_crt_dispatch |
| 8 | 0x00000000000144c;0 | ;Function | ;0 | ;0 | ; | ;dllmain_crt_process_attach |
| 9 | 0x0000000000015a8;0 | ;Function | ;0 | ;0 | ; | dllmain_crt_process_detach; |
| 10 | 0x000000000001650;0 | ;Function | ;0 | ;0 | ; | ;dllmain_dispatch |
| 11 | 0x0000000000017d0;0 | ;Function | ;0 | ;0 | ; | ;dllmain_raw |
| 12 | 0x0000000000017f0;0 | ;Function | ;0 | ;0 | ; | ; CRT INIT |
| 13 | 0x0000000000017f8;0 | ;Function | ;0 | ;0 | ; | ; DllMainCRTStartup |
| 14 | 0x000000000001844;0 | ;Function | ;0 | ;0 | ; | ; get entropy |
| 15 | 0x0000000000018cc;0 | ;Function | ;0 | ;0 | ; | ; security init cookie |
| 16 | 0x0000000000019a4;0 | ;Function | ;0 | ;0 | ; | ;DllMain |
| 17 | 0x0000000000019d0;0 | ;Function | ;0 | ;0 | ; | ;scrt_initialize_type_info |
| 18 | 0x0000000000019e4;0 | ;Function | ;0 | ;0 | ; | ; scrt uninitialize type info |
| 19 | 0x0000000000019f4;0 | ;Function | ;0 | ;0 | ; | ; local stdio printf options |
| 20 | 0x000000000001a00;0 | ;Function | ;0 | ;0 | ; | ; local_stdio_scanf_options |
| 21 | 0x0000000000001a0c;0 | ;Function | ;0 | ;0 | ; | ; scrt initialize default local stdio options |
| 22 | 0x000000000001a30;0 | ;Function | ;0 | ;0 | ; | ; scrt narrow argv policy::configure argv |

Fig. 6-5: Symbol Details of BCU example

6.2.2 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 can contain software-specific statements or even non-standard clauses. The label addresses of a vECU target differ from the addresses of a physical ECU target. This means that the original A2L file cannot be used for an XCP connection with a vECU target.

The generation of A2L files is an intricate task. VECU-BUILDER does not include this functionality. 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 *.a21.bak).

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 are specified in the YAML file. For instance:

| Original A2L file | Patched A2L file |
|--|--|
| /begin XCP_ON_TCP_IP 0x0100 /* XCP on IP 1.0 */ | /begin XCP_ON_TCP_IP */0x0100 /* XCP on IP 1.0 */ |
| <tcpport> /* Port */</tcpport> | 12345 /* Port */ |
| /ADDRESS "< IPADDR>" | ADDRESS "127.0.0.1" |
| /end XCP_ON_TCP_IP | /end XCP_ON_TCP_IP |

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 already includes an XCP slave, it is possible to remove this software component from the vECU software stack.
6.2.3 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, you must map them.

| Original A2L file | Mapped and patched A2L file |
|----------------------------|-----------------------------|
| /begin CHARACTERISTIC Hys- | /begin CHARACTERISTIC Hys- |
| teresis_LightOffIntensity | tLiOfInt |
| "unsigned integer 16bit" | "unsigned integer 16bit" |
| VALUE | VALUE |
| 0x0000000 | 00x00003016 |
| RTAA2L_Internal_Scalar_ | RTAA2L_Internal_Scalar_ |
| UnsignedWord | UnsignedWord |
| 0 | 0 |
| CompuMethods_STEP_100_ | CompuMethods_STEP_100_ |
| OFFSET_0 | OFFSET_0 |
| 0 | 0 |
| 100 | 100 |
| DISPLAY_IDENTIFIER Hys- | DISPLAY_IDENTIFIER Hys- |
| teresis_LightOffIntensity | teresis_LightOffIntensity |
| /end CHARACTERISTIC | /end CHARACTERISTIC |

6.2.4 HEX file generation

When the A2L patching mechanism is activated, VECU-BUILDER creates a HEX file. The HEX file (BCU.hex) is located in the corresponding workspace. You can use the HEX file for working with ETAS INCA. The HEX file contains the data INCA considers as the so-called reference page. INCA uses these data to calculate CRC check.

You can upload this generated HEX file while creating an INCA experiment. After uploading the file, workflows within INCA will be enabled. For more information about INCA and HEX file upload with INCA, see the corresponding INCA User Guides available in the ETAS download center.

To enable settings in INCA for a successful HEX file upload:

- 1. Open User Options.
- 2. Click General tab.
- 3. Go to Check dataset code.
- 4. Ensure that the value is set to No without warning.
- 5. Click OK.

| Calibration Session Log 🛛 Path 🔹 Data Es | change Limited EMU RAM ASAM-2MC Edito |
|---|---|
| Autostart 🕮 Experiment 🤝 | Hardware HEA Files Autocommit Import / Expo |
| Option | Value |
| Steve data collection information | - |
| Report con carro | 14 |
| Check dataset code | No without warning |
| the digits deviction | 100 |
| One freeding other reading the deceiption | |
| ACARS MUT THE INSTRUMENT | ADAPT - Lood TYLE A |
| MEA | MEAULA COMPANY PARTY ADDRESS |
| Web investor could be WECA | -unitere default income |
| | |
| | ~ |
| < | > |
| Check dataset code | ^ |
| | |
| | |

i) Note

If the settings are not as described, INCA will display an error and the HEX file will not be accepted.

6.2.5 Example of additional scripts - A2L characteristics as parameters

This is an example of how additional scripts can be used. With the following script, it is possible to use A2L characteristics as parameters of a vECU.

To run the script

- 1. To create a BCU workspace, follow the steps described in BCU example (only available for Windows)
- 2. Copy from \${VECUBUILDER_EXAMPLES}/BCU/additional_scripts
 - 6_get_characteristics.bat
 - 6_get_characteristics.py

t0 <My_BCU_Workspace>/build/additional_scripts.

- 3. Add the path of your Python interpreter directory to the Path environmental variable.
- 4. In vEcuConf.yaml file uncomment the following lines in the additional scripts section.

| additional_scripts: |
|---|
| #command:l before import.bat |
| #···trigger:·····before import |
| #priority:5 |
| #·-·command:·······2_after_import.bat |
| #···trigger:······after_import |
| #priority:5 |
| #·-·command:·······3_before_build_sources.bat |
| #···trigger:·····before_build_sources |
| #···priority:······5 |
| #·-·command:······4_before_build_fmus.bat |
| #···trigger:·····before_build_fmus |
| #···priority:······5 |
| #·-·command:······5_after_build_fmus.bat |
| #···trigger:······after_build_fmus |
| #···priority:·····5 |
| - command: |
| •• trigger: •••••••••••• after_import |
| • priority: • • • • • • • • • • • 5 |

- 5. Start building your workspace using 1 Import.bat on Windows.
- Characteristics were added in parameters section of vEcuConf.yaml file and are also available as parameters in the built BCU.fmu.

parameters:

- Hysteresis_LightOffIntensity
- Hysteresis_LightOffTime
- Hysteresis_LightOnIntensity
- Hysteresis LightOnTime
- Hysteresis WiperOffIntensity
- Hysteresis_WiperOffTime
- Hysteresis_WiperOnIntensity
- Hysteresis_WiperOnTime

6.3 EventTrigger example

This example shows the possibility of using event-triggered tasks. Therefore you need a function and a trigger function. The use of trigger inputs is optional. You can add up to 16 trigger inputs. This trigger function predicts when the next event might occur and returns if the next events needs to be triggered. You can find the event-triggered task in the task section in the vEcuConf.yaml file.

The function name is teeth_count and the trigger function is tooth_event. The function and the trigger function are defined in the Multiply.c file in vECU folder of the corresponding workspace. Optional inputs of trigger_function must be included in SymbolDetails.txt.

| C:\Use | s\Public\Documents\VECU-BUILDER_Workspaces\ | | | | | |
|-----------|--|--------------------------------|----------|---------|-------------------------|-----------------------------------|
| File Edit | Search View Encoding Language Settings Tools Man | ro Run TextFX Plugins Window ? | | | | + ¥ × |
| 🕞 📄 🗄 |] 🛍 🗟 🐚 🎒 🖌 🍿 🖺 ⊃ 🗲 🗰 🛬 🔍 🍕 🕻 | 8 🖓 🖏 1 🔚 🖉 🐘 🖓 🛍 | ۱ ا ا | 🕨 🔛 🛎 🗵 | 🖽 🐺 🛎 🔺 🗶 🖽 🎊 🔍 🤎 🖳 🖉 🗈 | a 🔶 |
| 🔚 Symbo | IDetails.txt 🗵 | | | | | |
| 1 | 19 symbol(s) found, 0 symbol | ol(s) with non-uni | que name | (s). | | |
| 2 | Address ;Size | ;Type | ;BF-0 | Ofs;BF- | Len;Value | ;Symbol Name |
| 3 | 0x000000000001470;0 | ;Function | ;0 | ;0 | ; | |
| | ;dummy_49d6c308_5c4d_4e35_a | ae01_3a51318a08c8 | | | | |
| 4 | 0x000000000001480;0 | ;Function | ;0 | ;0 | ; | ;multiply_factor1_factor2_product |
| 5 | 0x0000000000014a7;0 | ;Function | ;0 | ;0 | ; | ;add_factor1_factor2_sum |
| 6 | 0x0000000000014ce;0 | ;Function | ;0 | ;0 | ; | ;task_10ms |
| 7 | 0x000000000001529;0 | ;Function | ;0 | ;0 | ; | ;tooth event |
| 8 | 0x000000000001632;0 | ;Function | ;0 | ;0 | ; | ;teeth count |
| 9 | 0x000000000001648;0 | ;Function | ;0 | ;0 | ; | ;terminate |
| 10 | 0x000000000003020;8 | ;float64 | ;0 | ;0 | ;1 | ;factor1 |
| 11 | 0x000000000003028;8 | ;float64 | ;0 | ;0 | ;2 | ;factor2 |
| 12 | 0x000000000003030;8 | ;float64 | ;0 | ;0 | ;0.00400000000000000001 | ;eventTimeCheck |
| 13 | 0x000000000003038;1 | ;u8 | ;0 | ;0 | ;1 | ; ignition |
| 14 | 0x000000000003040;8 | ;float64 | ;0 | ;0 | ;357.14285714285711 | ;rpm |
| 15 | 0x000000000008060;8 | ;float64 | ;0 | ;0 | ;0 | ;product |
| 16 | 0x000000000008068;4 | ;s32 | ;0 | ;0 | ;0 | ;teeth counter |
| 17 | 0x000000000008140;8 | ;User-Defined | Type;0 | ;0 | ; | ;eeprom block b |
| 18 | 0x000000000008140;8 | ;float64 | ;0 | ;0 | ;0 | ;eeprom block b.last product |
| 19 | 0x000000000008150;16 | ;User-Defined | Type;0 | ;0 | ; | ;eeprom block a |
| 20 | 0x000000000008150;8 | ;u64 | ;0 | ;0 | ;0 | ;eeprom block a.lifetime ms |
| 21 | 0x000000000008158;2 | ;u16 | ;0 | ;0 | ;0 | ;eeprom block a.poweron count |
| 22 | | | | | | |
| 22 | mb - f - 11 | | | | | |

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

Fig. 6-6: Symbol Details of EventTrigger Example(Windows)

It is checked if the defined conditions of the trigger function are met. The default check time is 4 ms. If the conditions are met, the function is called. E.g. if the conditions of trigger function tooth_event are met, the function teeth_count is called.

6.3.1 Event-triggered tasks

To create a workspace based on the EventTrigger 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 EventTrigger configuration YAML file located in:
 - C:/ProgramData/ETAS/VECU-BUILDER/Examples_
 - 1.8.0/EventTriggerExample On Windows
- or

/opt/etas/VECU-BUILDER/Examples_1.8.0/EventTriggerExample ON Ubuntu 22.04 LTS.

- 2. Continue the process as described in Working with VECU-BUILDER.
- ⇒ The workspace was created.

6.3.2 Task scheduling with task trigger defined as cyclic

VECU-BUILDER does not have a built-in task scheduler. Instead, the task scheduling process is handled by the simulation environment that runs the FMU. VECU-BUILDER creates an FMU file that contains information about vECU tasks and their properties, which the simulation environment reads and uses to schedule the tasks.

In the tasks section of vEcuConf.yaml, you define the tasks that are supposed to execute when the FMU is run by the simulation environment.

When the task trigger is defined as cyclic (executed periodically), the following task properties become relevant:

- period: Specifies, how long the simulation environment has to wait between task executions
- first_call: Denotes the duration before the task is initially called.
- priority: Specifies, which task the simulation environment executes first, in case two or more tasks are supposed to be executed.
- max_calls: Enables the setting of a maximum limit for the total number of task executions during a simulation run.

The first_call property determines the initial delay before the task is first executed. Typically, it is set to zero. Nonetheless, there are instances where setting a non-zero value for first_call is necessary to avoid running the same task twice during the initial simulation step.

The repeated execution of some tasks in the very first step is by design. The convention adopted by VECU-BUILDER is that the time internal used by fmi2DoStep () is treated as right-closed, meaning the endpoint of the time interval is included in the interval itself. This sometimes results in duplicate execution of a task within the first simulation interval.

One way to address this issue is to disregard the initial simulation interval (or multiple intervals). If this is not a viable option, the alternative is to adjust the first_ call parameter to match the period for the impacted tasks in the vEcuConf.yaml file.

6.4 Template for plug-in V1 (FMI2)

6.4.1 Plug-In feature

With VECU-BUILDER plug-in feature it is possible to implement an own logic at run time into the following phases of FMU Runner:

- Instantiation
- Initialization
- Step execution
- Task execution

The phases follow the rules of FMI2 standard. For more information about FMI, see FMI standard.

Your configured plug-in implementation occurs in form of a CMake project. This CMake project needs to be used to implement the functionality of plug-in interface. For more information, see Plug-In interface.

Within plugin_template_v1_FMI2 folder (see Installed files and folders for installation path), the following files and folders are installed:

- include folder: Includes the header files containing VECU-BUILDER type definitions which you are not allowed to modify.
- build.bat/build.sh: Script, that contains a small list of commands which builds plugin_template shared object (DLL/SO).
- CMakeLists.txt: File, which contains CMake configuration for the plugin_template project.
- plugin_template.cpp and plugin_template.h: Main files dedicated for plug-in implementation.



Fig. 6-7: Installed files and folders for plug-in

The plug-in project uses two header files that defines VECU-BUILDER types. They are located in the include folder of the project template:

- plugininterface.h
- vecubTypes.h



Fig. 6-8: plugin_template folder

Once you implemented your own logic, the plug-in can be included in VECU-BUILDER as an additional resource.

As plug-in feature is supported for FMI2 and FMI3, the functions for each plug-in are grouped in pluginInterface.h.

For plug-in version V1 (FMI2) look for #define FMI_2_VERS_1 and #ifdef FMI_2_VERS_1 (obsolete:#define FMI2 and #ifdef FMI2.)



When using the plug-in, you must use include_symbol_details.

EXAMPLE

If you want to change the cycle time of "task_10ms" using a plug-in, then the symbol name "task_10ms" must be disclosed in the release vECU.

6.4.2 Plug-In configuration

You can include plug-ins as additional ressources, i.e.:

- additional_resources:
 - \${VECUBUILDER_EXAMPLES}\plugin_tem-

 $\verb|plate\CMake\Debug\plugin_template.dll\,On\,Windows$

or

- additional_resources:
 - \${VECUBUILDER_EXAMPLES}/Linux/plugin_tem-

plate/CMake/libplugin_template.so for Ubuntu 22.04 LTS.

i) Note

The plug-in can reside in a different location than \${VECUBUILDER_ EXAMPLES}\... and in vEcuconf.yaml you need to give the absolute path to the location of the build plug-in.

i) Note

You can rename a plug-in. However it is mandatory to follow the conventions mentioned below in any case:

- A *.dll file (for Windows) must start with plugin. It then will be managed as plug-in by VECU-BUILDER.
- A *.so file (for Linux) must start with libplugin. It then will be managed as plug-in by VECU-BUILDER.

If the plug-in does not follow this naming convention, VECU-BUILDER will not consider the *.dll or *.so as plug-in.

The files are saved in .fmu file in resources folder. From here VECU-BUILDER will load them as plug-ins.

6.4.3 Plug-In interface

To see and use the plug-in interface, open pluginInterface.h in include folder.

Plug-In functions

You can use several plug-in functions. There are optional and mandatory plug-in functions and therefore not all functions from plug-in interface have to be implemented.

Pointer and function pointers

For Visual Studio compiler on Windows and GNU complier from MinGW on Ubuntu:

All plug-in interface functions have one argument as a pointer of type struct VecubCallbacks1.

For Visual Studio compiler on Windows and GNU compiler on Ubuntu and GNU compiler from MinGW on Windows and Ubuntu

All plug-in interface functions have one argument as a pointer of type struct VecubCallbacks1. Furthermore, the structure also contains conditional compilation instructions that only define certain function pointers if the PLUGIN_ EXTENSION macro is defined.

The following plug-in callback functions are available. The trigger for each plug-in function call is related to the FMI protocol.

| Callback Function Name | Trigger | Priority |
|------------------------|--|-----------|
| vecubPluginVersion | During fmi2Instantiate and before vecubInstantiate1 | mandatory |
| vecublnstantiate1 | During fmi2Instantiate and after vec- ubPluginVersion | mandatory |
| vecubFmi2EnterInit1 | During fmi2EnterInitializationMode | optional |
| vecubFmi2ExitInit1 | During fmi2ExitInitializationMode | optional |
| vecubPreDoStep1 | At the beginning of fmi2DoStep | optional |
| vecubPostDoStep1 | At the end of fmi2DoStep | optional |
| vecubPreTask1 | Before calling a task | optional |
| vecubPostTask1 | After calling a task | optional |
| Terminate1 | During fmi2Terminate | mandatory |

VecubCallbacks1 pointer keeps a list of function pointers which could be used in order for plug-in to grab information from VECU-BUILDER. Available functionalities (callbacks) are:

- logging
- accessing symbol information like: address, size, symbol type, bitfield_offset, bitfield_length
- managing task objects
- handling symbols data like read and write their values

| typedef void (cdecl* v typedef void* (cdecl* v typedef int (cdecl* v typedef PtrSymbolAccess& (cdecl* v | VecubCallbackLog1) VecubCallbackGetSymbolInfo1) VecubCallbackGetTask1) VecubCallbackGetSymbolAccessor1) | <pre>(vecubString); (vecubString); // returns SymbolInfo* (vecubString, Task*&); (vecubString);</pre> | | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux |
|--|---|---|----|--|
| <pre>#if defined(PLUGIN_EXTENSION) typedef IntFloat64 (cdecl* v typedef void (cdecl* v #endif</pre> | VecubCallbackGetSymbolValue1) (co VecubCallbackSetSymbolValue1) (co | nst char*); nst char*, IntFloat64 val); | B | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux + GNU Compiler from MinGW on Windows |
| using VecubCallbacks1 = struct VecubCa VecubCallback.og1 VecubCallbacksetsymbolInfo1 VecubCallbackGetTask1 VecubCallbackGetSymbolAccessor1 | <pre>llbacks1 { log1; getSymbolInfo1; getTask1; getSymbolAccessor1;</pre> | | | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux |
| <pre>#if defined(PLUGIN_EXTENSION)</pre> | getSymbolValue1; setSymbolValue1; | | -0 | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux + GNU Compiler from MinGW on Windows |
| VecubCallbackSetDoubleValue VecubCallbackSetDoubleValue VecubCallbackSetDoubleValue VecubCallbackSetUnsignedLongValue VecubCallbackSetLongLongValue VecubCallbackSetTongLongValue VecubCallbackGetTaskTriggarValue VecubCallbackGetTaskTriggerValue VecubCallbackGetTaskDoubleValue VecubCallbackGetTaskDoubleValue | <pre>setTaskPeriod; setTaskFirstCall; setTaskNextCall; setTaskNumberMaxCalls; setTaskNumberMaxCalls; getTaskNume; getTaskTagger; getTaskTrigger; getTaskFeriod; getTaskFirstCall; us_astTackPaigneit;</pre> | | | |
| VecubCallbackGetTaskLongLongValue VecubCallbackGetTaskLongLongValue VecubCallbackGetTaskLongLongValue VecubCallbackGetTaskDoubleValue Ltendif | <pre>lue getTaskPriority; getTaskMaxCalls; getTaskCalls; getTaskNextCall;</pre> | | | |

Fig. 6-9: function pointers

Example for usage of callbacks for Visual Studio compiler on Windows and GNU compiler on Ubuntu

The following code visualizes how you can use callbacks for plug-in version 1.

```
// ===== plug-in vers. 01 =====
DllExport vecub1Status __cdecl vecubFmi2EnterInit1(const VecubCall-
backs1* vecubCallbacks)
{
    if (!vecubCallbacks)
       return vecub1Status::vecubError;
```

The code defines the function vecubFmi2EnterInit1 that takes a pointer to the structure VecubCallbacks1 as input. It checks if the passed pointer vecubCallbacks is valid. If not, it returns vecub1Status::vecubError.

```
// logging
if (vecubCallbacks->log1)
  {
   vecubCallbacks->log1("user message");
  }
```

It checks if the log1 function is defined in vecubCallbacks structure. If yes, it calls the function with the argument user message.

```
// get symbol info
SymbolInfo* symbol = static_cast<SymbolInfo*>(vecubCallbacks->getSym-
bolInfo1("factor1"));
std::cout << "The symbol has address: " << std::hex << symbol->addr <<
std::endl;</pre>
```

It calls getSymbolInfol function to obtain information about a symbol named factor1. The returned symbol is stored in the variable symbol. The address of the symbol is printed using std::cout.

It calls getSymbolAccessor1 function to obtain an accessor for the factor1 symbol. The accessor is stored in the reference variable symbolAccessor. It reads a value of type IntFloat64 from the symbol and assigns the value 5 to it. Then, it writes the value back to the symbol.

i) Note

PtrSymbolAccess and symbolAccessor with its own methods became obsolete but are still supported. They are replaced by methods like:

vecubCallbacks->getSymbolValue1

vecubCallbacks->setSymbolValue1

```
// get task object
Task* task{nullptr};
vecubCallbacks->getTask1("task_10ms", task);
task->setTNextCall(123);
if (task) {
   std::cout << "FOUND! task: " << " ~ " << task->getName() <<
std::endl;
   }
}</pre>
```

It calls getTask1 function to obtain a task object with the name task_Oms. The task object is stored in the pointer variable task.setTNextCall function is called on the task object with the value 123 as the argument. If the task object is valid, a message is printed with the task's name.

return vecub1Status::vecub0K;
}

lt returns vecub1Status::vecub0K.

Example for usage of callbacks for Visual Studio compiler on Windows and GNU compiler on Ubuntu and GNU compiler from MinGW on Windows and Ubuntu

The following code visualizes how you can use callbacks for plug-in version 1.

```
// ===== plug-in vers. 01 =====
DllExport vecub1Status __cdecl vecubFmi2EnterInit1(const VecubCall-
backs1* vecubCallbacks)
{
    if (!vecubCallbacks)
        return vecub1Status::vecubError;
```

The code defines the function vecubFmi2EnterInit1 that takes a pointer to the structure VecubCallbacks1 as input. It checks if the passed pointer vecubCallbacks is valid. If not, it returns vecub1Status::vecubError.

```
// logging
if (vecubCallbacks->log1)
  {
    vecubCallbacks->log1("user message");
    }
```

It checks if the log1 function is defined in vecubCallbacks structure. If yes, it calls the function with the argument user message.

```
// get symbol info
SymbolInfo* symbol = static_cast<SymbolInfo*>(vecubCallbacks->getSym-
bolInfo1("factor1"));
std::cout << "The symbol has address: " << std::hex << symbol->addr <<
std::endl;</pre>
```

It calls getSymbolInfol function to obtain information about a symbol named factor1. The returned symbol is stored in the variable symbol. The address of the symbol is printed using std::cout.

```
// get symbol accessor
IntFloat64 symbolValue = vecubCallbacks->getSymbolValue1("factor1");
symbolValue.f = 10.00;
vecubCallbacks->setSymbolValue1("factor1", symbolValue);
```

It is using getSymbolValue1 function pointer to retrieve the value of a symbol named "factor1" and stores it in a variable called symbolValue. It is assumed that symbolValue is a structure containing an integer and a float value, and the code is accessing the float value using the .f notation and setting it to 10.00. Secondly, it is using setSymbolValue1 function pointer to update the value of the factor1 symbol with the modified symbolValue.

```
// get task object
Task* tasknullptr{};
vecubCallbacks->getTask1("task_10ms", task);
std::cout << "Task getTaskNextCall: " << vecubCallbacks->getTaskNex-
tCall(&task) << std::endl;
std::cout << "Task setTaskNextCall: " << vecubCallbacks->setTaskNex-
tCall(&task, 1) << std::endl;
std::cout << "Task getTaskNextCall: " << vecubCallbacks->getTaskNex-
tCall(&task, 1) << std::endl;
std::cout << "Task getTaskNextCall: " << vecubCallbacks->getTaskNex-
tCall(&task, 1) << std::endl;</pre>
```

A variable task of type task* is created and initialized with the value nullptr. Then the getTask1 method of the vecubCallbacks object is called to retrieve the task named task_10ms and store it in the task variable. The methods getTaskNextCall and setTaskNextCall of the vecubCallbacks object are then called to get and set the next call time of the task. The results of these method calls are then output to the console using std::cout.

return vecub1Status::vecub0K;
}

It returns vecub1Status::vecub0K.

6.4.4 What a plug-in can do with tasks

A plug-in can access a task defined in VECU-BUILDER and can change different properties of a task at run time.

As described in Plug-In functions, you can use several functions. One of these functions is used to get a task. The used function pointer is getTask1, which returns a task type. For more information, see Fig. 6-9

Once the plug-in access a task, it can manage through the task interface and the behavior of a task during run time.

The interface of the task is available in plug-in template project through the definition of task class in include/vecubTypes.h file.

In this class, you can see specific functions of task class which manage i.e the task name, task period, or task priority.

6.5 Template for plug-in V2 (FMI3)

6.5.1 Plug-In feature V2

With VECU-BUILDER plug-in feature it is possible to implement an own logic at run time into the following phases of FMU Runner:

- Instantiation
- Initialization
- Step execution
- Task execution

The phases follow the rules of FMI3 standard. For more information about FMI, see FMI standard.

Your configured plug-in implementation occurs in form of a CMake project. This CMake project needs to be used to implement the functionality of plug-in interface. For more information, see Plug-In interface V2.

Within $plugin_template_v2_FMI3$ folder (see Installed files and folders for installation path), the following files and folders are installed:

- include folder: Includes the header files containing VECU-BUILDER type definitions which you are not allowed to modify.
- build.bat/build.sh: Script, that contains a small list of commands which builds plugin_template shared object (DLL/SO).
- CMakeLists.txt: File, which contains CMake configuration for the plugin_template project.
- plugin_template.cpp and plugin_template.h: Main files dedicated for plug-in implementation.



Fig. 6-10: Installed files and folders for plug-in

The plug-in project uses two header files that defines VECU-BUILDER types. They are located in the include folder of the project template:

- plugininterface.h
- vecubTypes.h





Once you implemented your own logic, the plug-in can be included in VECU-BUILDER as an additional resource.

As plug-in feature is supported for FMI2 and FMI3, the functions for each plug-in are grouped in pluginInterface.h.

For plug-in version V2 (FMI3) look for #define FMI_3_VERS_2 and #ifdef FMI_3_VERS_2 (obsolete:#define FMI3 and #ifdef FMI3.)



When using the plug-in, you must use include_symbol_details.

EXAMPLE

If you want to change the cycle time of "task_10ms" using a plug-in, then the symbol name "task_10ms" must be disclosed in the release vECU.

6.5.2 Plug-In confuguration V2

You can include plug-ins as additional ressources:

- additional_resources:
 - \${VECUBUILDER_EXAMPLES}\plugin_tem-
 - $\verb|plate\CMake\Debug\plugin_template.dll\,On\,Windows$

or

- additional_resources:
 - \${VECUBUILDER_EXAMPLES}/Linux/plugin_tem-
 - plate/CMake/libplugin_template.so On Ubuntu 22.04 LTS.

i) Note

The plug-in can reside in a different location than \${VECUBUILDER_ EXAMPLES}\... and in vEcuconf.yaml you need to give the absolute path to the location of the build plug-in.

i) Note

You can rename a plug-in. However it is mandatory to follow the conventions mentioned below in any case:

- A *.dll file (for Windows) must start with plugin. It then will be managed as plug-in by VECU-BUILDER.
- A *.so file (for Ubuntu) must start with libplugin. It then will be managed as plug-in by VECU-BUILDER.

If the plug-in does not follow this naming convention, VECU-BUILDER will not consider the *.dll or *.so as plug-in.

The files are saved in .fmu file in resources folder. From here VECU-BUILDER will load them as plug-ins.

6.5.3 Plug-In interface V2

To see and use the plug-in interface, open pluginInterface.h in include folder.

Plug-In functions

You can use several plug-in functions. There are optional and mandatory plug-in functions and therefore not all functions from plug-in interface have to be implemented.

Pointer and function pointers

For Visual Studio Compiler on Windows and GNU complier from MinGW on Ubuntu:

All plug-in interface functions have one argument as a pointer of type struct VecubCallbacks2.

For Visual Studio Compiler on Windows and GNU compiler on Ubuntu and GNU compiler from MinGW on Windows and Ubuntu

All plug-in interface functions have one argument as a pointer of type struct VecubCallbacks2. Furthermore, the structure also contains conditional compilation instructions that only define certain function pointers if the PLUGIN_ EXTENSION macro is defined.

| Callback | Trigger | Priority |
|---------------------|--|-----------|
| vecubPluginVersion | During fmi3Instantiate and before vec- ubInstantiate2 | mandatory |
| vecublnstantiate2 | During "fmi3Instantiate" and after "vec- ubPluginVersion" | mandatory |
| vecubFmi3EnterInit2 | During fmi3EnterInitializationMode | optional |
| vecubFmi3ExitInit2 | During fmi3ExitInitializationMode | optional |
| vecubPreDoStep1 | At the beginning of fmi3DoStep | optional |
| vecubPostDoStep2 | At the end of fmi3DoStep | optional |
| vecubPreTask2 | Before calling a task | optional |
| vecubPostTask2 | After calling a task | optional |
| Terminate2 | During fmi3Terminate | mandatory |

This VecubCallbacks2 pointer keeps a list of function pointers which could be used in order for plug-in to grab information from VECU-BUILDER.

Available functionalities (callbacks) are:

- logging
- accessing symbol information like: address, size, symbol type, bitfield_offset, bitfield_length
- managing task objects
- handling symbols data like read and write their values

| typedef void (cdecl* \ typedef void* (cdecl* \ typedef int (cdecl* \ typedef PtrSymbolAccess& (cdecl* \ | VecubCallbackLog2) VecubCallbackGetSymbolInfo2) VecubCallbackGetTask2) VecubCallbackGetSymbolAccessor2) | <pre>(vecubString); (vecubString); // returns SymbolInfo* (vecubString, Task*&); (vecubString);</pre> | | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux |
|--|--|---|---|--|
| <pre>#if defined(PLUGIN_EXTENSION) typedef VarTypes (cdecl* \ typedef void (cdecl* \ #endif</pre> | VecubCallbackGetSymbolValue2) VecubCallbackSetSymbolValue2) | (const char*); (const char*, VarTypes val); | B | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux + GNU Compiler from MinGW on Windows |
| using VecubCallbacks2 = struct VecubCal VecubCallbackLog2 VecubCallbackGetSymbolInfo2 VecubCallbackGetTask2 VecubCallbackGetSymbolAccessor2 | <pre>llbacks2 { log2; getSymbolInfo2; getTask2; getSymbolAccessor2;</pre> | | | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux |
| #if defined(PLUGIN_EXTENSION) VecubCallbackGetSymbolValue2 VecubCallbackSetSymbolValue2 | getSymbolValue2; setSymbolValue2; | | | Visual Studio compiler on Windows and GNU Compiler from MinGW on Linux + GNU Compiler from MinGW on Windows |
| VecubcallbackSetDoubleValue VecubcallbackSetDoubleValue VecubcallbackSetDoubleValue VecubcallbackSetDoubleValue VecubcallbackSetLongLongValue VecubcallbackSetLongLongValue VecubcallbackSetTaskStringValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDoubleValue VecubcallbackSetTaskDongValue VecubcallbackSetTaskDongValue VecubcallbackSetTaskDongValue VecubcallbackSetTaskDoubleValue | <pre>setTaskPeriod; setTaskFirstCall; setTaskFirstCall; setTaskNextCall; setTaskNumberMaxCalls; setTaskNumberCalls; getTaskTrigger; getTaskTrigger; getTaskFirstCall; lue getTaskPriority; getTaskAxCalls; getTaskCall; getTaskCall;</pre> | | | |
| }; | | | - | |

Fig. 6-12: function pointers

Example for Usage of callbacks for Visual Studio compiler on Windows and GNU compiler on Ubuntu

The following code visualizes how you can use callbacks for plug-in version 2.

```
// ===== plug-in vers. 02 =====
DllExport vecub2Status __cdecl vecubFmi3EnterInit2(const VecubCall-
backs2* vecubCallbacks)
{
    if (!vecubCallbacks)
        return vecub2Status::vecubError;
```

The code defines the function vecubFmi3EnterInit2 that takes a pointer to the structure VecubCallbacks2 as input. It checks if the passed pointer vecubCallbacks is valid. If not, it returns vecub1Status::vecubError.

```
// logging
if (vecubCallbacks->log2)
{
  vecubCallbacks->log2("user message");
}
```

It checks if the log2 function is defined in vecubCallbacks structure. If yes, it calls the function with the argument user message.

```
// get symbol info
SymbolInfo* symbol = static_cast<SymbolInfo*>(vecubCallbacks->getSym-
bolInfo2("factor1"));
std::cout << "The symbol has address: " << std::hex << symbol->addr <<
std::endl;</pre>
```

It calls getSymbolInfo2 function to obtain information about a symbol named factor1. The returned symbol is stored in the variable symbol. The address of the symbol is printed using std::cout.

```
// get symbol accessor
PtrSymbolAccess& symbolAccessor = vecubCallbacks->getSymbolAccessor2
("factor1");
VarTypes val = (*symbolAccessor).get();
val.f64 = 5;
(*symbolAccessor).set(val);
```

It calls getSymbolAccessor2 function to obtain an accessor for the factor1 symbol. The accessor is stored in the reference variable symbolAccessor. It reads a value of type IntFloat64 from the symbol and assigns the value 5 to it. Then, it writes the value back to the symbol.

i) Note

PtrSymbolAccess and symbolAccessor with its own methods became obsolete but are still supported. They are replaced by methods like:

vecubCallbacks->getSymbolValue2

vecubCallbacks->setSymbolValue2

```
// get task object
Task* task{nullptr};
vecubCallbacks->getTask2("task_10ms", task);
task->setTNextCall(123);
if (task) {
   std::cout << "FOUND! task: " << " ~ " << task->getName() <<
std::endl;
   }
}</pre>
```

It calls getTask2 function to obtain a task object with the name task_Oms. The task object is stored in the pointer variable task.setTNextCall function is called on the task object with the value 123 as the argument. If the task object is valid, a message is printed with the task's name.

return vecub2Status::vecubOK;
}

lt returns vecub1Status::vecub0K.

Example for usage of callbacks for Visual Studio compiler on Windows and GNU compiler on Ubuntu and GNU compiler from MinGW on Windows and Ubuntu

The following code visualizes how you can use callbacks for plug-in version 2.

```
// ===== plug-in vers. 02 =====
DllExport vecub2Status __cdecl vecubFmi3EnterInit2(const VecubCall-
backs2* vecubCallbacks)
{
    if (!vecubCallbacks)
        return vecub2Status::vecubError;
```

The code defines the function vecubFmi3EnterInit2 that takes a pointer to the structure VecubCallbacks2 as input. It checks if the passed pointer vecubCallbacks is valid. If not, it returns vecub1Status::vecubError.

```
// logging
if (vecubCallbacks->log2)
  {
    vecubCallbacks->log2("user message");
    }
```

It checks if the log2 function is defined in vecubCallbacks structure. If yes, it calls the function with the argument user message.

```
// get symbol info
SymbolInfo* symbol = static_cast<SymbolInfo*>(vecubCallbacks->getSym-
bolInfo2("factor1"));
std::cout << "The symbol has address: " << std::hex << symbol->addr <<
std::endl;</pre>
```

It calls getSymbolInfo2 function to obtain information about a symbol named factor1. The returned symbol is stored in the variable symbol. The address of the symbol is printed using std::cout.

```
// get symbol accessor
VarTypes val2 = vecubCallbacks->getSymbolValue2("factor1");
std::cout << "symbol has value: " << val2.f64 << std::endl;
val2.f64 = 123.45;
vecubCallbacks->setSymbolValue2("factor1", val2);
```

It retrieves the symbol value associated with the key factor1 using getSymbolValue2 method and stores it in a variable called val2 of type VarTypes. It then prints the value of the symbol to the console using std::cout. It modifies the value of the symbol by assigning a new value (123.45) to the f64 field of the val2 variable. Finally, it uses the setSymbolValue2 method to update the value of the symbol associated with the key factor1 to the new value stored in the val2 variable.

```
// get task object
Task* tasknullptr{};
vecubCallbacks->getTask1("task_10ms", task);
std::cout << "Task getTaskNextCall: " << vecubCallbacks->getTaskNex-
tCall(&task) << std::endl;
std::cout << "Task setTaskNextCall: " << vecubCallbacks->setTaskNex-
tCall(&task, 1) << std::endl;
std::cout << "Task getTaskNextCall: " << vecubCallbacks->getTaskNex-
tCall(&task, 1) << std::endl;
std::cout << "Task getTaskNextCall: " << vecubCallbacks->getTaskNex-
tCall(&task, 1) << std::endl;</pre>
```

A variable task of type task* is created and initialized with the value nullptr. Then the getTask1 method of the vecubCallbacks object is called to retrieve the task named task_10ms and store it in the task variable. The methods getTaskNextCall and setTaskNextCall of the vecubCallbacks object are then called to get and set the next call time of the task. The results of these method calls are then output to the console using std::cout.

return vecub2Status::vecubOK;
}

It returns vecub1Status::vecub0K.

6.5.4 What a plug-in can do with tasks V2

A plug-in can access a task defined in VECU-BUILDER and can change different properties of a task at run time.

As described in Plug-In functions, you can use several functions. One of these functions is used to get a task. The used function pointer is getTask2, which returns a task type. For more information, see Fig. 6-12

Once the plug-in access a task, it can manage through the task interface and the behavior of a task during run time.

The interface of the task is available in plug-in template project through the definition of task class in include/vecubTypes.h file.

In this class, you can see specific functions of task class which manage i.e the task name, task period, or task priority.

7 Controlling VECU-BUILDER

7.1 Manual interaction

You can operate VECU-BUILDER via the provided batch/shell scripts. For some user inputs, such as selecting a workspace directory, the software displays dialogs.

7.2 Command Line Interface (CLI)

Besides the manual interaction method you can also operate VECU-BUILDER via a Command Line Interface (CLI). VECU-BUILDER is a CLI native software, and the batch/shell scripts allow manual interaction. For more information about the Installation using CLI, see Silent installation of VECU-BUILDER.

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: Prevents automatic forwarding to the next stage (create workspace, import, build).

--version: Prints the version information.

-h: Prints list of all optional arguments.

To see all CLI optional arguments and their description

1. Open a command prompt on Windows

or

Open a terminal on Ubuntu 22.04 LTS.

2. Execute the following command:

1_Import.bat -h for Windows

or

./1 Import.sh -h for Ubuntu 22.04 LTS.

```
SimpleExample>1_Import.bat -h
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.
                         Instead of showing dialogs, the title and message are printed choosing the first available option. In case of an error
 --no-dialogs
                         message, the process will exit returning non-zero.
 --stop-on-success
                         If the current script succeeded, it will not proceed with the
                         next one.
                         show program's version number and exit
 --version
```

Fig. 7-1: CLI optional arguments on Windows 10

| F | /opt/etas/VECU-BUILDER_Work | Q | ≡ | | | × | | | |
|---|--|---------------------------------|--------------------------------|-------------------------------------|--------------------------------|---|--|--|--|
| <pre>/opt/etas/VECU-BUILDER_Workspaces/CLI\$./1_Import.sh -h usage: VECU-BUILDER [-h] [new-project-path NEW_PROJECT_PATH] [no-dialogs] [stop-on-success] [version]</pre> | | | | | | | | | |
| (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. | | | | | | | | | |
| new-project-path NFW | PROJECT PATH | | | | | | | | |
| no-dialogs | If you start a new project, you ne files and folders should be saved. Instead of showing dialogs, the ti printed choosing the first availab of an error message, the process w | ed a tle a le op ill e | path and m otion exit | where essage . In c returr | e its e are case ning | | | | |
| stop-on-success version | non-zero. If the current script succeeded, i with the next one. show program's version number and | t wil exit | .l no | t prod | ceed | | | | |

Fig. 7-2: CLI optional arguments on Windows 10

import, build. To change this behavior, use --stop-on-success.

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

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

Tab. 7-1: Mapping of CLI arguments to scripts

To build the SimpleExample via two command lines

After creating the workspace, stop the process so you can copy a specific YAML file into your workspace. Then trigger the import without

 ${\tt stop-on-success}$ and let it finish the build automatically.

1. Open a command prompt on Windows

or

Open a terminal on Ubuntu 22.04 LTS.

2. Navigate to the directory where the installer is located executing the following command:

```
cd %VECUBUILDER_HOME%.
```

3. Execute the following command:

```
\texttt{CreateWorkspace.bat} \text{ On } \textbf{Windows}
```

or

./CreateWorkspace.sh On Ubuntu 22.04 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\ <version> >CreateWorkspace.batnew-proj C:\Users\Public\Documents\VECU-BUILDER_Workspaces\CLIno-dialogsstop-on (C) 2020-2023 ETAS GmbH. All rights reserved. VECU-BUILDER <version></version></version> | ect-p -succ | ath ess | |
| ***** | | | |
| ### Creating new workspace ### | | | |
| *************************************** | | | |
| [09:05:36] 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". | | | |
| [09:05:36] 2 of 3: Creating project template | | | |
| [09:05:36] 3 of 3: Please update "vEcuConf.yaml" | | | |
| Please save and close the editor after editing "vEcuConf.yaml" | | | |
| The usage of the optionno-dialogs forces the process to continue. | | | |
| *** SUCCESS *** | | | |
| | | | |

Fig. 7-3: Workspace creation via CLI on Windows

| ſ | 1 | /opt/etas/VECU-BUII | LDER/ | Q | Ξ | | | × |
|-------------------|---|--|---|-------------------------------|-------------------------|-------------------|--------------|-----------|
| [SU(*** | :/opt new-project-path /opt/etas/ ccess | VECU-BUILDER_Worl | ER/ <version> s kspaces/CLI -</version> | 5 ./Cr -no-d | eateW ialog ***** | orksp ss ** | ace. top- | sh on- |
| VE(VE(**; | UBUILDER_HOME: /opt/et UBUILDER_EXAMPLES: /opt/et | as/VECU-BUILDER/ as/VECU-BUILDER/ ****** | <version> / Examples_<ver *****</ver </version> | sion> ***** | / ***** | ** | | |
| (C) VE(##1 | 2020-2024 ETAS GmbH. All U-BUILDER <version></version> | rights reserved. | *** | *##### | ###### | ## | | |
| ### ### [1(| Creating new workspace | ###################################### | ###################################### | *##### | # ##### | ## ## | | |
| Ple Nev [10 | ease select a workspace fol project path is "/opt/eta 5:53:31] 2 of 3: Creating p | der for your pro s/VECU-BUILDER_W project template | ject! orkspaces/CL1 ml" | :". | | | | |
| Ple The | ease save and close the edi usage of the optionno- SUCCESS *** | tor after editing dialogs forces t | g "vEcuConf.y he process to | / <mark>aml"</mark> > cont | inue. | | | |
| | | | | | | | | |

Fig. 7-4: Workspace creation via CLI on Ubuntu 22.04 LTS

()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.

To use your project specific YAML file in this newly created workspace:

1. Execute the following command:

copy /y <source> <destination> ON Windows



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

or

```
cp -i <source> <destination> On Ubuntu 22.04 LTS.
```

You are asked if you want to overwrite the file.



Fig. 7-5: Copy your project specific YAML file on Windows



Fig. 7-6: Copy your project specific YAML file on Ubuntu 22.04 LTS

To continue building your workspace:

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

cd <destination>.

2. Run the command:

1 Import.bat --no-dialogs ON Windows

or

./1_Import.sh --no-dialogs ON Ubuntu 22.04 LTS.

7.3 Ubuntu 22.04 LTS Command Line Interface

You can create a new workspace on WSL Ubuntu 22.04 LTS using the Ubuntu Command Line Interface.

)Note

Downloading dependencies or installing VECU-BUILDER only runs in WSL1, working with VECU-BUILDER only runs in WSL2. Ensure that the WSL version aligns with the specific action. If necessary, you need to change the version.

- For WSL1:
- wsl --set-version Ubuntu-22.041
- For WSL2:
- wsl --set-version Ubuntu-22.04 2
 - 1. Open PowerShell and set WSL Ubuntu 22.04 LTS version to 2 using the following command:

```
wsl --set-version Ubuntu-22.04 2
```

2. Open Ubuntu 22.04 LTS and ensure that the environment variables are set correctly using the following command:

```
env | grep -i vecu
```

~\$ env | grep -i vecu
VECUBUILDER_EXAMPLES=/opt/ETAS/VECU-BUILDER/Examples_<version>/
VECUBUILDER_HOME=/opt/ETAS/VECU-BUILDER/<version>/

- Change directory using the following command: cd \$VECUBUILDER HOME
- 4. Create a new workspace using the following command:

```
./CreateWorkspace.sh --no-dialogs --new-project-path
/opt/etas/VECU-BUILDER Workspaces/SimpleExample/
```

or

With installed gnome terminal workspace creation also works in dialog mode using the following command:

```
./CreateWorkspace.sh --new-project-path /opt/etas/VECU-
BUILDER_Workspaces/SimpleExample/
```

A new workspace was created. For more information about workspace content, see Workspace content.

8 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. This enables 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 you need to attach before continuing.

i) 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. 8-1: 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.

8.1 Debugging with Visual Studio 2019

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

For more information about Visual Studio 2019, see Visual Studio documentation.

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 22.04 LTS.
- ⇒ The VS2019 debugger is invoked and loads the CMake project.
- 3. Navigate to where you want to start debugging and set a breakpoint there.
- 4. In the Menu tab click Debug > Start Debugging (F5).
- s fmusim is invoked and the debugger is attached.



Fig. 8-1: VS 2019 Debugger attached

8.2 Debugging with Visual Studio Code

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

Prerequisites for debugging with Visual Studio Code

It is obligatory to install the following packages in Visual Studio Code:

- Microsoft C/C++ Extension Pack

For Debugging in WSL Ubuntu with Visual Studio Code additionally install the following packages:

- C/C++ extensions for Visual Studio Code and WSL Ubuntu in Windows computer Host
- CMake extensions for Visual Studio Code and WSL Ubuntu in Windows computer Host
- CMake Tools extensions for Visual Studio Code and WSL Ubuntu in Windows computer Host
- gdb in Ubuntu WSL (see Installing dependent software packages.)

Debugging is not possible without installing these packages.

Visual Studio Code requires some further extensions and will prompt you to install them by default.

For more Information about Visual Studio Code, see Visual Studio Code documentation.

To debug with Visual Studio Code in Windows

- 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 set a breakpoint there.
- 4. Click Start Debugging (F5).
- 5. In the **menu** panel on the left click **Run and Debug**.
- fmusim is invoked and the debugger is attached.

To debug with Visual Studio Code in Ubuntu 22.04 LTS

- 1. Navigate to your workspace.
- 2. Start debugging using the following command:
 - \$./3b_StartDebugger.sh
- ➡ Visual Studio Code opens.
- 3. Navigate to where you want to start the debugging and set a breakpoint there.
- 4. In the menu panel on the left click Run and Debug.
- 5. Click Start Debugging (F5).
- ⇒ fmusim is invoked and the debugger is attached.

To debug with Visual Studio Code in WSL

- 1. Check if gnome-terminal and gdb are installed. If not installed, see Installing VECU-BUILDER on Ubuntu 22.04 LTS for WSL.
- 2. Navigate to your workspace.
- 3. Start debugging using the following command:
 - ./3b_StartDebugger.sh
- ➡ Visual Studio Code opens.
- 4. Navigate to where you want to start the debugging and set a breakpoint there.
- 5. Click Start Debugging (F5).
- 6. In the menu panel on the left click Run and Debug.
- ⇒ fmusim is invoked and the debugger is attached.

File Edit Selection View Go Run Terminal Help

| Ð | RUN AND D 🕨 Deb | ugi∨ ⇔ C | Multiply.c × |
|----------|------------------|----------|--|
| | \sim VARIABLES | ē v | ECU > imported > C Multiply.c > 🛇 multiply_factor1_factor2_product() |
| | > Locals | | <pre>1 double factor1 = 1.0;</pre> |
| | > Registers | | 2 double factor2 = 2.0; |
| የ_ዖ | | | 3 double product = 0.0; |
| 6 | | | 4 5 struct EEPROM block A { |
| | | | 6 unsigned long long lifetime ms: // lifetime of the vECU i |
| Ú | | | 7 unsigned short poweron_count; // how often was this vEC |
| | | | 8 }; |
| В | | | 9 struct EEPROM_block_B { |
| | | | 10 double last_product; // last value of product |
| | | | 11 }; |
| | | | 12 |
| | | | <pre>13 struct EEPROM_block_A eeprom_block_a;</pre> |
| | | | 14 struct EEPROM_block_B eeprom_block_b; |
| | | | 15 |
| | | 0 | 16 Int multiply_factor1_factor2_product() { |
| | | D. | 1/ product = factor1 * factor2; |
| | | | |
| | | | |

Fig. 8-2: VS Code Debugger attached

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

| (C) 2020-2023 ETAS GmbH. All rights reserved. VECU-BUILDER <version></version> | |
|--|-----|
| | |
| ### Building sources of vECU ### | |
| ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, | |
| [09:15:24] 1 of 4: Reading config: vEcuConf.yaml | |
| [09:15:24] 2 of 4: Creating Visual Studio Code debug configuration | |
| [09:15:24] 3 of 4: Running scripts triggered through "before_build_source - No script defined in the vEcuConf.yaml file | es" |
| [09:15:24] 4 of 4: Compiling and linking | |
| - SimpleExample.dll (Windows 64bit) | |
| CMake not found. | |
| For more details, please refer to the User Guide. *** FAILURE *** | |

Fig. 9-1: CMake not found error

Possible reason

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

It seems that CMake is either not installed or not properly registered on your computer.

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:

| 📑 Registry Editor | | | | | — | | \times |
|--|---|---------------|--------|---------------------|--------|--|----------|
| File Edit View Favorites Help | | | | | | | |
| Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Kitware\CMake | | | | | | | |
| 🗙 📙 Kitware | ^ | Name | Туре | Data | | | |
| | | (Default) | REG_SZ | (value not set) | | | |
| | ~ | ab InstallDir | REG_SZ | C:\Program Files\CM | ∕lake∖ | | |

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, you need to install and register it 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 computer.

Possible solution

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 | | | | | | | | \times |
|-------------------|--|------|------|----------------|-------------------|-----------|----------|----------|
| File Edit | View Favorites | Help | | | | | | |
| Computer | Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Notepad++ | | | | | | | |
| | - Notepad++ | ^ | Name | Type REG_SZ | Data C:\Progra | am Files\ | Notepad+ | ·+ |

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 | | | | | — | | \times |
|------------------------------|-------|---------------|----------------|------------|--------------|-----------|----------|
| File Edit View Favorites Hel | р | | | | | | |
| Computer\HKEY_LOCAL_MACHIN | E\SOF | TWARE\WOW6432 | Node\Notepad++ | | | | |
| | ^ | Name | Туре | Data | | | |
| - Notepad++ | | 💩 (Default) | REG_SZ | C:\Program | n Files (x86 | 5)\Notepa | ad++ |

9.3 Some breakpoints not being hit

Possible reason

Depending on your compiler configurations, the resulting vECU may be built without some debugging information. This may result in the debugger not being able to hit some breakpoints.

ocation:

Fig. 9-3: 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 ("00")

9.4 (SymbolInfo.dll) the *.die file is too large to load

Possible reason

The operating system does not provide sufficient amount of memory required to load the *.die file.

Possible solution

Use a computer with sufficient amount of memory.

9.5 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 your network. To try to identify Diagnose. | e. Otherwise, there might be and resolve network proble | e a problem with ems, click |
| See details | Diagnose | Cancel |

Fig. 9-4: Network Error - Localhost cannot be accessed
Possible solution

Setup the local share and obtain the UNC path name.

To setup a local share:

- 1. Navigate to the drive, you want to share. (e.g. C:/drive)
- 2. Right-click the drive and click **Properties**.
- 3. Click the **Sharing** tab.
- 4. Click Advanced Sharing. You will need Admin Rights to proceed.

| General Tools Hardware Sharing Network File and Folder Sharing C:\ Not Shared C:\ Not Shared Sharing Network Path: Not Shared Share Share Share Advanced Sharing Set custom permissions, create multiple shares, and set other advanced sharing options. Shares, and set other | General Tools Hardware Sharing Network File and Folder Sharing C:\ Not Shared C:\ Not Shared Sharing Network Path: Not Shared Share Share Share Share Share Set custom permissions, create multiple shares, and set other advanced sharing options. Image: C:\ Not Shared Set custom permissions, create multiple shares, and set other advanced sharing options. | Security | Previo | Previous Versions | | |
|---|---|---|---|----------------------|-----------|--|
| Network File and Folder Sharing C:\ Not Shared Network Path: Not Shared Share Advanced Sharing Set custom permissions, create multiple shares, and set other advanced sharing options. | Network File and Folder Sharing C:\ Not Shared Network Path: Not Shared Share Advanced Sharing Set custom permissions, create multiple shares, and set other advanced sharing options. Method Shares and set other Set custom permissions area to the shares and set other Set custom permissions area to the shares and set other Set custom permissions area to the shares and set other Set custom permissions area to the shares and set other Set custom permissions area to the shares area to the | General | Tools | Hardware | Sharing | |
| Advanced Sharing Set custom permissions, create multiple shares, and set other advanced sharing options. | Advanced Sharing Set custom permissions, create multiple shares, and set other advanced sharing options. | Network File an C:\ Not S Network Path: Not Shared Share | d Folder Sharing Shared | | | |
| | Auvanced Shaning | | | | | |
| | | Advanced Shar Set custom per advanced shar Advanc | ing missions, create ing options. ed Sharing | multiple shares, and | set other | |
| | | Advanced Shar Set custom per advanced shar Advanc | ing missions, create ring options. red Sharing | multiple shares, and | set other | |

- 5. Activate the checkbox Share this folder.
- 6. Click OK.

| Advanced Sharing | \times | | | |
|--|----------|--|--|--|
| ✓ Share this folder | | | | |
| Settings | | | | |
| S <u>h</u> are name: | | | | |
| c ~ | | | | |
| Add Remove | | | | |
| Limit the number of simultaneous users to: | | | | |
| Comments: | | | | |
| Permissions Caching | | | | |
| OK Cancel Apply | | | | |

⇒ The drive is now shared and the network path is displayed.

| Security | Previo | Quota | |
|---|---|------------------------|-----------|
| General | Tools | Hardware | Sharing |
| Network File and C:\ Share Network Path: \\ Share Advanced Sharin Set custom perm advanced sharin | Folder Sharing d g nissions, create r ng options. | nultiple shares, and s | set other |
| Advance | d Sharing | | |

When you are logged in during the execution of the vECU, you need 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 must to be changed for security reasons.

To change the permissions

- 1. Click Advanced Sharing. You might need Admin Rights to proceed.
- 2. Click Permissions.
- 3. Click Add.

| Permissions for C | | × |
|--|-------|--------|
| Share Permissions | | |
| Group or user names: | | |
| Section Everyone | | |
| | | |
| | | |
| | | |
| | | |
| | Add | Remove |
| | | |
| Permissions for Everyone | Allow | Deny |
| Permissions for Everyone Full Control | Allow | Deny |
| Permissions for Everyone Full Control Change | Allow | Deny |
| Permissions for Everyone Full Control Change Read | Allow | Deny |
| Permissions for Everyone Full Control Change Read | Allow | Deny |
| Permissions for Everyone Full Control Change Read | Allow | Deny |
| Permissions for Everyone Full Control Change Read | Allow | Deny |
| Permissions for Everyone Full Control Change Read | Allow | Deny |
| Permissions for Everyone Full Control Change Read | Allow | Deny |

- 4. Enter the object name (username) to be selected.
- 5. Click 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 | Object Types |
| From this location: | |
| | Locations |
| Enter the object names to select (<u>examples</u>): | |
| Manual Rev (Million) and | Check Names |
| | |
| Advanced OK | Cancel |

- 8. To mark the entry, click the user name.
- 9. Activate the checkboxes Full Control and Change.
- 10. To mark the entry, click **Everyone**.
- 11. To remove the permission for **everyone**, click **Remove**.

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

| Share Permissions Group or user names: Add Remove Permissions for BCN1LR Allow Deny Full Control | | | \times |
|---|------------------------|------------|----------|
| Group or user names: Add Remove Permissions for BCN1LR Allow Deny Full Control Image: Control Image: Control T Image: Control Image: Control | re Permissions | | |
| Add Remove Add Remove Permissions for BCN1LR Allow Deny Full Control Image: Control Image: Control T Image: Control Image: Control | oup or user names: | | |
| Add Remove Permissions for BCN1LR Allow Deny Full Control | | | |
| Permissions for BCN1LR Allow Deny Full Control Image: Control Image: Control | | Add Remove | |
| Full Control | missions for BCN1LR | Allow Deny | |
| Change ⊻ □ Read ✓ □ | Full Control Change | | |
| OK Cancel Apply | Read | | |

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

9.6 Redirecting function calls did not work as expected

Possible reason

The GNU compiler optimization level 2 (-O2) includes inline-small-functions which is incompatible with redirect_function_calls.

By default VECU-BUILDER uses the compile settings RelWithDebInf, which includes some optimizations. For gcc this uses the setting **-O2**, which includes inline-small-functions.

Possible solution

Change the settings in additional_compile_flags to enable redirect_function_calls.

There are 3 ways to deactivate the optimization:

- A. -OO: Completely deactivates optimization. This has the advantage that the compiler time of user workspace decreases.
- B. -O1: Reduces the level of optimization from default 2 to 1.
- C. -O2 -f-no-inline-small-functions: Keeps optimization to level 2 but only disables the special optimization with -f-no-inline-small-functions.

For more information, see Options that control optimization.

9.7 License check failed

Possible reason

- The LiMa installation is corrupt.
- LiMa can not reach the license server.

Possible solution

- Reinstall VECU-BUILDER described in Installation on Windows 10 and Installation on Ubuntu 22.04 LTS or contact technical support.
- Check network settings to get a connection to the license server.

9.8 Building sources failed

Possible reason

In some cases, building sources fails with various error messages. To save time, CMake uses caches, e.g. a link to the build tool is stored.

Possible solution

To fix a broken CMake cache, delete the cache and rebuild the sources.

- 1. Navigate to the v ECU folder.
- 2. Delete everything except the imported folder.
- 3. Rebuild the sources using 2_Build.bat on Windows or 2_Build.sh on Ubuntu 22.04 LTS.

If the build fails due to CMake reason, you can find more details in build/- log/build cmake.log file.

9.9 Indentation errors in YAML file

Indentation errors can occur and they are difficult to detect.

Possible solution

To check for indentation errors in YAML file, use a YAML Checker. There are online softwares available. You can search for YAML checker in any search engine. Follow the instructions given by the selected YAML Checker.

9.10 Failed to parse symbols

It is possible, that symbol/debug information is missing in the binary. The error message below treats the missing debug information during build process when build_mode is import_compiled.



Possible reason

The error occurs due to some mishandling of DLL/SO when <code>build_mode</code> in YAML file is set to <code>import_compiled</code>.

Possible solution

- If the used build_tool is one of the Visual Studio compile versions (like 16 2016 or 17 2022), ensure that besides the DLL loaded, there must also exist a mandatory PDB file. The source location for DLL & PDB is given by YAML settings import_external_compiled_vecu and get_updates_from.
- If the build_tool used is MinGW Makefiles for Windows or Unix Makefiles for Ubuntu, only the DLL/SO is required. Ensure that the DLL/SO compulsory contains debug information. A DIE file is created locally when building FMUs runs. Afterwards also symbolDetails.txt is created.
- Ensure that the build_tool YAML settings match the compiler used to build the DLL:
 - If the imported DLL was built with the GNU compiler from MinGW, ensure that build_tool is MinGW Makefiles (build_tool: MinGW Makefiles).
 - If the imported DLL was built with a Visual Studio compiler, ensure that build_tool is **Visual Studio xx xxxx** (build_tool: Visual Studio xx xxxx).

For more information see import_external_compiled_vecu in configuration chapter.

9.11 Could not load the vECU binary

During the vECU FMU build the <code>SymbolDetails.txt</code> variables are updated with the initial values by loading of the <code>vECU.dll/so</code> binary and reading the values from RAM. In cases where the <code>vECU.dll/so</code> cannot be loaded, the <code>Sym-bolDetails.txt</code> variables receive the default value 0. You are informed with an appropriate message in the <code>build_fmu.log</code> file. In this case, the FMU build is possible, but the generated FMU execution will not work due to the same <code>vECU.dll/so</code> loading problem as during the build.

Possible reason

One or more dynamic libraries, required for the vECU.dll/so execution are missing / could not be found by the OS.

Possible solution for Windows:

Check by loading the vECU.dll, which dynamic libraries are missing using the open source SW Dependencies.

| Dependencies (x64) | | | | | - | _ | | × |
|--|----|---------|-------------|------|------------|------|-----------|-----|
| File View Options Help | | | | | | | | |
| Sil_vECU 😣 vECU 😣 | | | | | | | | |
| C:\Temp\vECU.dll C:\WINDOWS\SysWOW64\kernel32.dll C:\WINDOWS\SysWOW64\kernel32.dll | PI | Ordinal | | Hint | | Func | tion | |
| libwinpthread-1.dll | E | Ordinal | | Hint | | Func | tion | |
| Module | | Machine | Туре | | File Size | | Image Bas | e |
| C:\WINDOWS\SysWOW64\kernel32.dll | | i386 | Dll; Execut | able | 0x0009db88 | В | 0x6b800 | 000 |
| C:\WINDOWS\SysWOW64\MSVCRT.dll | | i386 | Dll; Execut | able | 0x000bd458 | В | 0x10100 | 000 |
| ? | | N/A | N/A | | N/A | | N/A | |
| < | | | | | | | | > |
| Loading PE file "C:\Temp\vECU.dll" successful. | | | | | | | | |

Provide the missing info about the dynamic library to the system using the vECUConf.yaml options:

Option 1: (most recommended): Add the library to the resources folder of the vECU.fmu USing additional resources.

Option 2: Provide the corresponding path to the missing library using environment_variables.

Possible solution for Ubuntu 22.04 LTS

To check the dependencies in Linux run 1dd command on a *.so file.

Example: ldd libQt5Gui.so

| Я | | /usr/share/ETAS/LiMa | Q = | | | × |
|--|--|---|---|--------|-----|---|
| <pre>linux-gate.s libQt5Core.s libpng16.so. libz.so.1 => libstdc++.so libm.so.6 => libgcc_s.so. libc.so.6 => libpthread.s libdl.so.2 = /lib/ld-linu</pre> | <pre>:/usr/share/ETAS/LiMa\$ l o.1 (0xf7ed8000) o.5 => /usr/share/ETAS/LiMa/ 16 => /lib/i386-linux-gnu/lib/i386-linux-gnu/lib/i386-linux-gnu/lib/i386-linux-gnu/lib/lib/i386-linux-gnu/libc.sc 0.0 => /lib/i386-linux-gnu/libcl.sc 0.0 => /lib/i386-linux-gnu/libdl. x.so.2 (0xf7eda000)</pre> | dd ./x32/bin/libQt50 /./x32/bin/libQt5Cord bpng16.so.16 (0xf700) .bstdc++.so.6 (0xf6e) .6 (0xf6d83000) ogcc_s.so.1 (0xf6d640) .6 (0xf6d575000) Libpthread.so.0 (0xf0 .so.2 (0xf6b4c000) | Gui.so e.so.5 (0x) 85000) 88000) 000) 6b52000) | f70d70 | 00) | |

Fig. 9-5: Check dependencies in Linux

This command will give you the list of the dependencies. If one file is missing, this command will give a **not found** message.



Fig. 9-6: Not found message

One way to tell to a software where to search for its SO dependencies is to set LD_LIBRARY_PATH to the location where these SO resides.

9.12 Skipping plug-in

[WARNING] Skipping plugin plugin_template_vers.01.dll [OK] Loading plugin plugin_template_vers.02.dll [FMI] fmi3InstantiateCoSimulation(instanceName="Fmu30", instantiation]

Possible reason

The FMI version set in YAML file configuration does not fit to the plug-in version.

Possible solution

Check the selected FMI version in the YAML file and ensure that it corresponds to the correct and required plug-in version.

- If FMI version is set to 2 in YAML file, then the plug-in version must be 1.
- If FMI version is set to 3 in YAML file, then the plug-in version must be 2.

If there is no correspondence between FMI version and plug-in version, change it accordingly.

9.13 Encoding cannot be defined for a VARVAL file

Possible reason

The encoding was used for VARVAL file and not for DCM file.

Possible solution

Ensure that the encoding is defined for DCM file. For more information, see initial_data in configuration chapter.

9.14 Encoding of DCM file is not supported

| (C) 2020-2024 ETAS GmbH. All rights reserved. | |
|--|---|
| VECU-BUILDER <version></version> | |
| | *## |
| ### Building FMU | *## |
| | +## |
| | *** |
| [09:22:01] 1 of 7: Reading config: vEcuConf.yaml | |
| [09:22:02] 2 of 7: Running scripts triggered through "before_build_1 | fmus" |
| No script defined in the vEcuConf.yaml file | |
| [09:22:02] 3 of 7: Building inputs, outputs, parameters, tasks | |
| [09:22:03] 4 of 7: Patching a2l file | |
| - No a2l file defined in the vEcuConf.yaml file | |
| [09:22:03] 5 of 7: Processing initial data | |
| C:/ProgramData/ETAS/VECU-BUILDER/Examples_1.7.6 | 0-a3/SimpleExample/init/InitialData.dcm |
| Encoding ' of DCM file is not supported. | |
| File 'C:/Users/Public/Documents/VECU-BUILDER_Workspaces/ | <pre>/build/fmu/resources/init/InitialData.</pre> |
| dcm' will not be processed. | |

Possible reason

The used encoding is not supported or there is a misspelling in the encoding.

Possible solution

Ensure that the encoding is supported and there is no misspelling in the encoding. For more information, see initial_data in configuration chapter.

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 offers trainings for its products:

www.etas.com/academy

ETAS headquarters

ETAS GmbH

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

