DRIVING EMBEDDED EXCELLENCE

ETAS VECU-BUILDER V1.1

User Guide

www.etas.com

ETAS VECU-BUILDER V1.1 – User Guide R04 EN – 10.2022

Contents

1 Safety and Privacy Information

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

1.1 Intended Use

The product is designed to produce a virtual ECU for microcontrollers from existing ECU source codes or from precompiled binaries. The virtual ECU is designed for simulation, debugging, and pre-calibration of ECU software in a PC-based virtual simulation environment.

In general, virtual ECUs may not be real-time capable. If you control physical devices with a virtual ECU, the system may respond unexpectedly. Take suitable precautions to ensure safe operation.

ETAS GmbH cannot be made liable for damage which is caused by incorrect use and not adhering to the safety information. Please adhere to the ETAS Safety Advice (see `documentation` folder).

1.2 Target Group

This product is directed at trained qualified personnel in development of automotive ECU software (e.g., function developer, application engineer, ECU software integrator, system engineer or calibration engineer) at OEMs, tier-1 or tier-2 suppliers in the automotive 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 Classification of Safety Messages

Safety messages warn of dangers that can lead to personal injury or damage to property:

DANGER

DANGER indicates a hazardous situation that, if not avoided, will result in death or serious injury.

WARNING

WARNING indicates a hazardous situation that, if not avoided, could result in death or serious injury.

CAUTION

CAUTION indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.

NOTICE

NOTICE indicates a situation that, if not avoided, could result in damage to property.

1.4 Privacy Information

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

1.4.1 Data Processing

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

1.4.2 Technical and Organizational Measures

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

Personal data in log files can be deleted by tools in the operating system.

2 About VECU-BUILDER

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

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

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

2.1 Basics

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

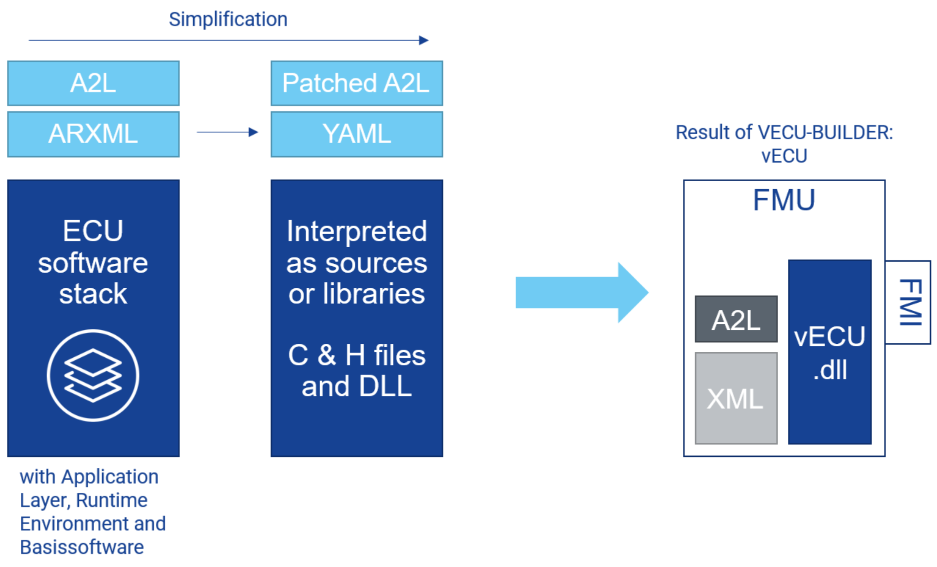

Simplification
A2L
ARXML
Patched A2L
YAML
ECU software stack
with Application Layer, Runtime Environment and Basissoftware
Interpreted as sources or libraries
C & H files and DLL
Result of VECU-BUILDER: vECU
FMU
A2L
XML
vECU .dll
FMI

Figure 1 - Basic concept and result of VECU-BUILDER

2.2 Virtual ECU

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

2.3 vECU Creation Process Workflow

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

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

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

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

3 Installation

This chapter provides information for preparing and performing the installation and for licensing the software.

3.1 System Requirements

The following system prerequisites are required:

PC Hardware	min. 2 GHz 3 GHz Dual-Core or higher recommended
	min. 8 GB RAM 32 GB RAM recommended
Operating System	Windows® 10 (64 bit)
Free Disk Space	5 GB (not including the size for application data) >100 GB recommended
Required third-party tools	CMake (version ≥3.15)
Recommended third-party tools	Notepad++
Optional third-party tools	Microsoft Visual Studio 2015, 2017, 2019

3.2 Preparation

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

NOTE

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

3.3 Installation Content

The installation content can either be downloaded from ETAS license and download portal (http://www.etas.com/support/licensing) or executed from the DVD.

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

3.4 Installing

1. Go to the directory where the installation file is located and execute the `VECU_BUILDER_installer_1.1.0.exe` file.

   ⇨ The Setup Wizard opens.

2. Click **Next**.

   ⇨ The "End User License Agreement" window opens.

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

   ⇨ The "Safety Advice" window opens.

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

   ⇨ The "Installation Path" window opens.

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

   ⇨ The "Ready to Install" window opens.

8. Click **Install**.

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

9. Click **Next**.

   ⇨ The "Third-party Software" window opens.

10. Install CMake (required) and Notepad++ (recommended) see the links below in the installation dialog:
    CMake (version 3.15 or higher)
    Notepad++
11. Click **Next**.

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

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

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

3.5 Installed Files and Folders

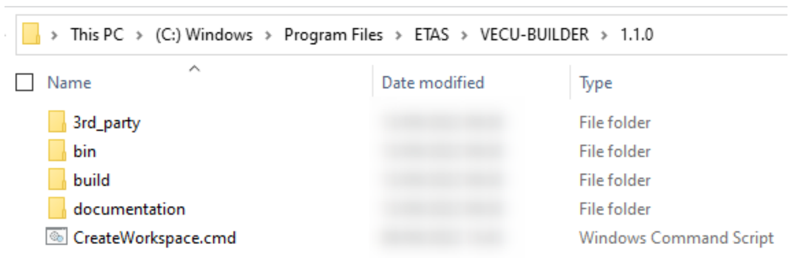
VECU-BUILDER Tool

The default installation location is

`C:\Program Files\ETAS\VECU-BUILDER\1.1.0`

and it is recommended not to alter the installation location.

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



> This PC > (C:) Windows > Program Files > ETAS > VECU-BUILDER > 1.1.0

Name	Date modified	Type
3rd_party	[redacted]	File folder
bin	[redacted]	File folder
build	[redacted]	File folder
documentation	[redacted]	File folder
CreateWorkspace.cmd	[redacted]	Windows Command Script

Figure 2 - Installation content of VECU-BUILDER tool

The content of this folder consists of several sub-folders and one command script:

- **`3rd_party`**: contains the third party tools of FMU Checker and MinGW.
- **`bin`**: contains DLL and EXE 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.cmd`**: creates a new workspace. After executing this CMD you will be guided through the process step by step.

VECU-BUILDER Examples

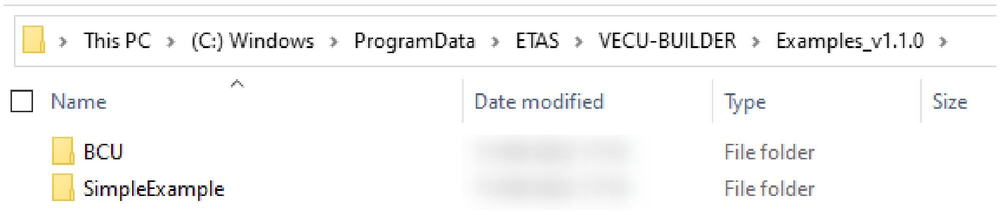
You can find ready-to-use examples in the following location:

`C:\ProgramData\ETAS\VECU-BUILDER\Examples_v1.1.0`

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

The following two examples are delivered along with the tool:

- BCU (Body Control Unit)
- Simple Example



> This PC > (C:) Windows > ProgramData > ETAS > VECU-BUILDER > Examples_v1.1.0 >

Name	Date modified	Type	Size
BCU	[redacted]	File folder	
SimpleExample	[redacted]	File folder	

Figure 3: Examples delivered along with VECU-BUILDER

VECU-BUILDER Workspaces

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

The default folder is created during the installation process:

`C:\Users\Public\Documents\VECU-BUILDER_Workspaces`

Access to Artefacts

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

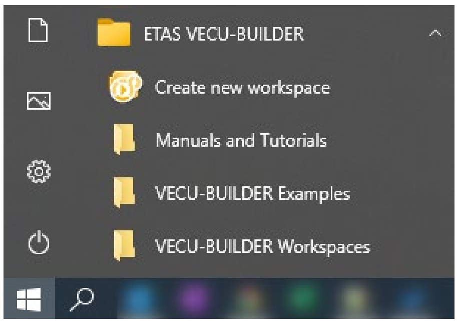


Figure 4 - Start Menu entries

3.6 Licensing

The use of VECU-BUILDER is protected by electronic licensing. Valid licenses are necessary to operate ETAS VECU-BUILDER and its add-ons. The use of unlicensed ETAS software is prohibited. The required licenses are not included in this delivery.

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

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

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

3.7 Uninstalling

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

4 Working with VECU-BUILDER

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

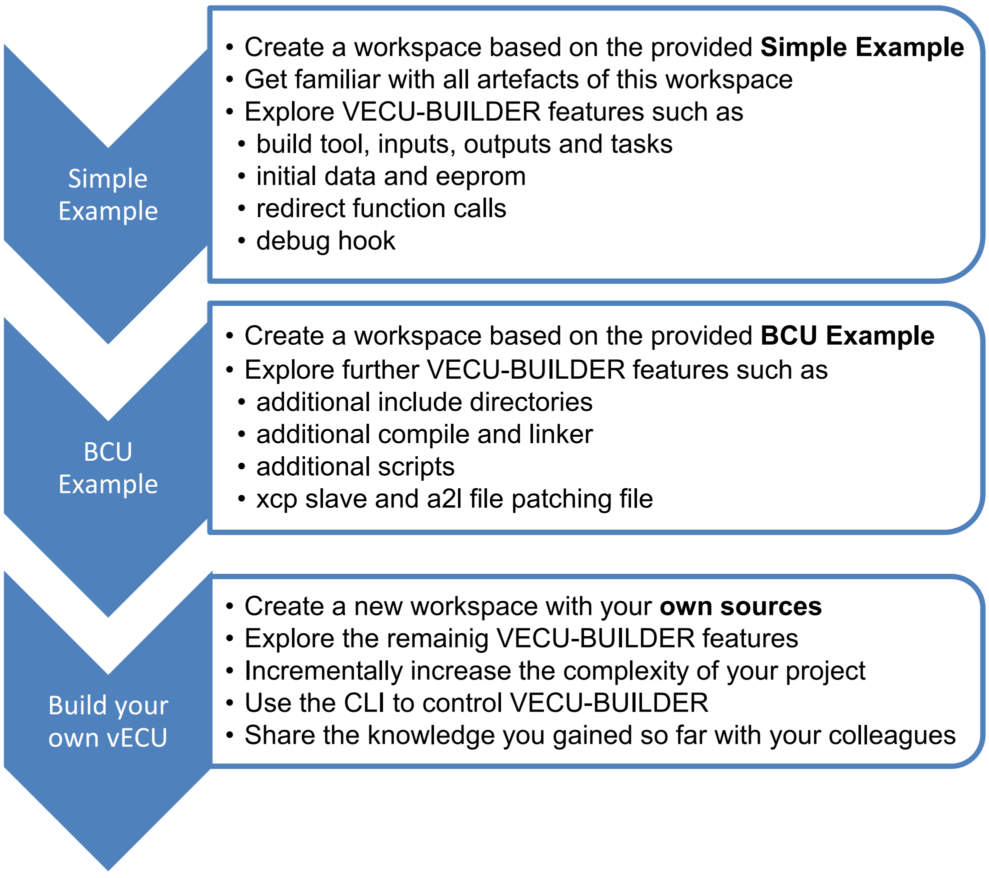

Simple Example
• Create a workspace based on the provided **Simple Example**
• Get familiar with all artefacts of this workspace
• Explore VECU-BUILDER features such as
• build tool, inputs, outputs and tasks
• initial data and eeprom
• redirect function calls
• debug hook
BCU Example
• Create a workspace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

Figure 5 - The learning path

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

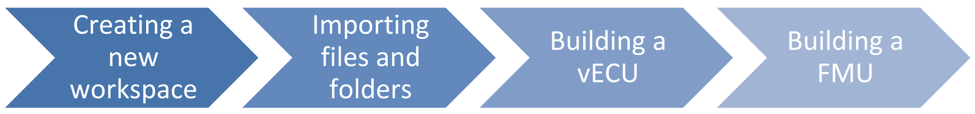


Figure 6 - VECU-BUILDER stages

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

4.1 Creating a New Workspace

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

NOTE

Workspaces are designed for parallel use.

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

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

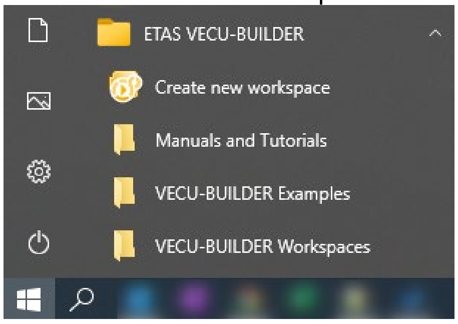
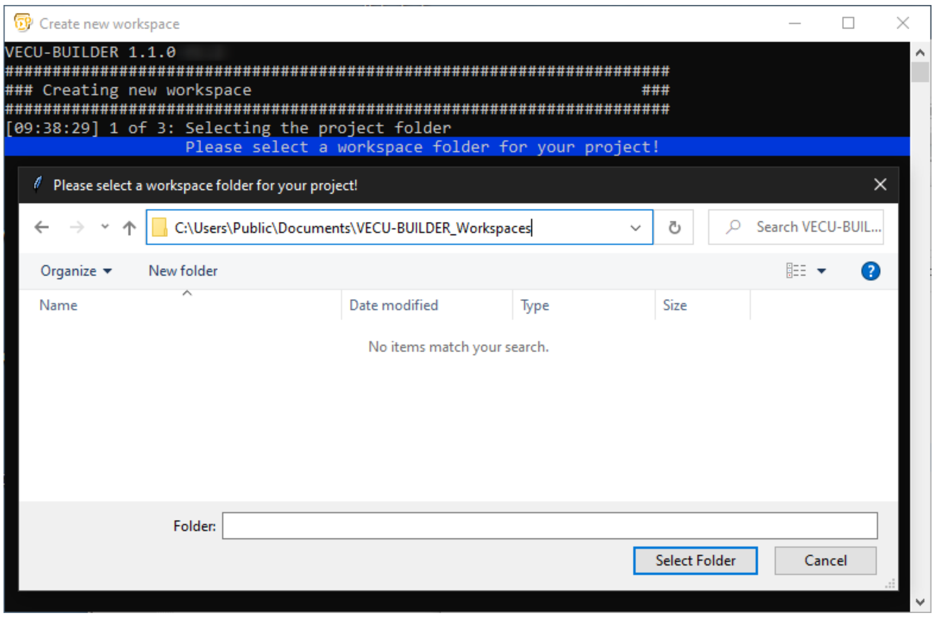


Figure 7 - Start Menu entries

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

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



```
VECU-BUILDER 1.1.0
##############################################################################
### Creating new workspace                                                 ###
##############################################################################
[09:38:29] 1 of 3: Selecting the project folder
                    Please select a workspace folder for your project!
```

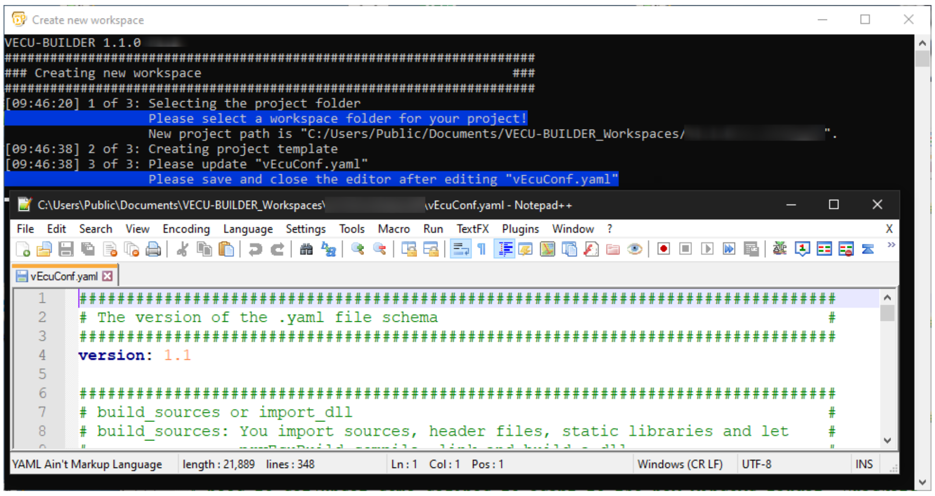
Create new workspace
Please select a workspace folder for your project!
C:\Users\Public\Documents\VECU-BUILDER_Workspaces
Search VECU-BUIL...
Organize
New folder
Name
Date modified
Type
Size
No items match your search.
Folder:
Select Folder
Cancel

Figure 8 - Select workspace location

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

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

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



```
VECU-BUILDER 1.1.0 [redacted]
######################################################################
### Creating new workspace                                         ###
######################################################################
[09:46:20] 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/[redacted]".
[09:46:38] 2 of 3: Creating project template
[09:46:38] 3 of 3: Please update "vEcuConf.yaml"
                   Please save and close the editor after editing "vEcuConf.yaml"
```

C:\Users\Public\Documents\VECU-BUILDER_Workspaces\ [redacted] \vEcuConf.yaml - Notepad++

```
################################################################################
# The version of the .yaml file schema                                         #
################################################################################
version: 1.1

################################################################################
# build_sources or import_dll                                                  #
# build_sources: You import sources, header files, static libraries and let    #
```

YAML Ain't Markup Language   length : 21,889   lines : 348   Ln : 1   Col : 1   Pos : 1   Windows (CR LF)   UTF-8   INS

Create new workspace
File Edit Search View Encoding Language Settings Tools Macro Run TextFX Plugins Window ?
vEcuConf.yaml

Figure 9 - Default configuration

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

   ➯ Your new workspace is now created.

The process will automatically continue with the next stage.

4.2 Importing Files and Folders

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

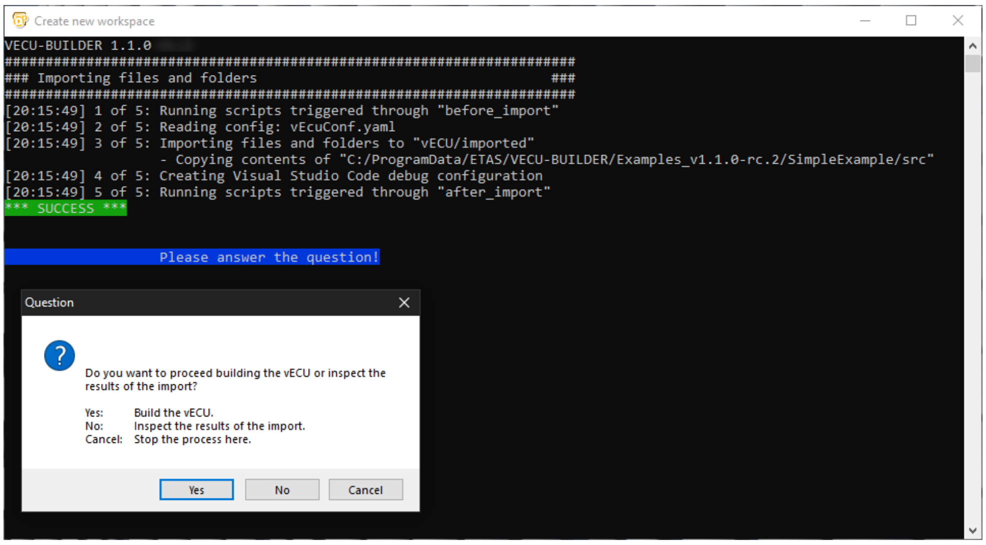
NOTE

During the import stage, files and folders get copied into the workspace. For reasons of portability, we recommend creating 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 the `1_Import.cmd` command script.

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



```
Create new workspace
VECU-BUILDER 1.1.0 [redacted]
########################################################################
### Importing files and folders                                      ###
########################################################################
[20:15:49] 1 of 5: Running scripts triggered through "before_import"
[20:15:49] 2 of 5: Reading config: vEcuConf.yaml
[20:15:49] 3 of 5: Importing files and folders to "vECU/imported"
                   - Copying contents of "C:/ProgramData/ETAS/VECU-BUILDER/Examples_v1.1.0-rc.2/SimpleExample/src"
[20:15:49] 4 of 5: Creating Visual Studio Code debug configuration
[20:15:49] 5 of 5: Running scripts triggered through "after_import"
*** SUCCESS ***


                   Please answer the question!
```

Question

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

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

Figure 10 - Proceed with vECU Build dialog or inspect the results

4. Click **Yes**.

   ⇨ Your new workspace is now created.

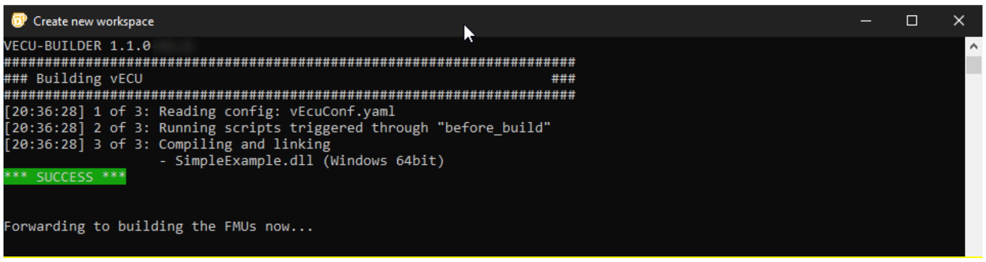
The process will continue with the next stage.

4.3 Building the vECU

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

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

If you work in an already existing workspace, you can trigger this stage by running the `2_Build.cmd` command script.



```
Create new workspace
VECU-BUILDER 1.1.0 [redacted]
########################################################################
### Building vECU                                                    ###
########################################################################
[20:36:28] 1 of 3: Reading config: vEcuConf.yaml
[20:36:28] 2 of 3: Running scripts triggered through "before_build"
[20:36:28] 3 of 3: Compiling and linking
                   - SimpleExample.dll (Windows 64bit)
*** SUCCESS ***


Forwarding to building the FMUs now...
```

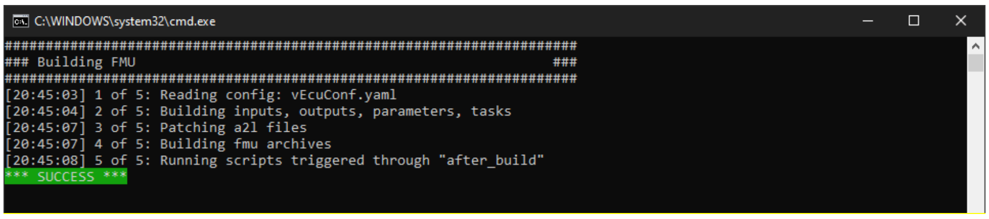
Figure 11 - Building vECU completed

The process will automatically continue with the next stage.

4.4 Building the FMU

During this stage, the DLL 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.

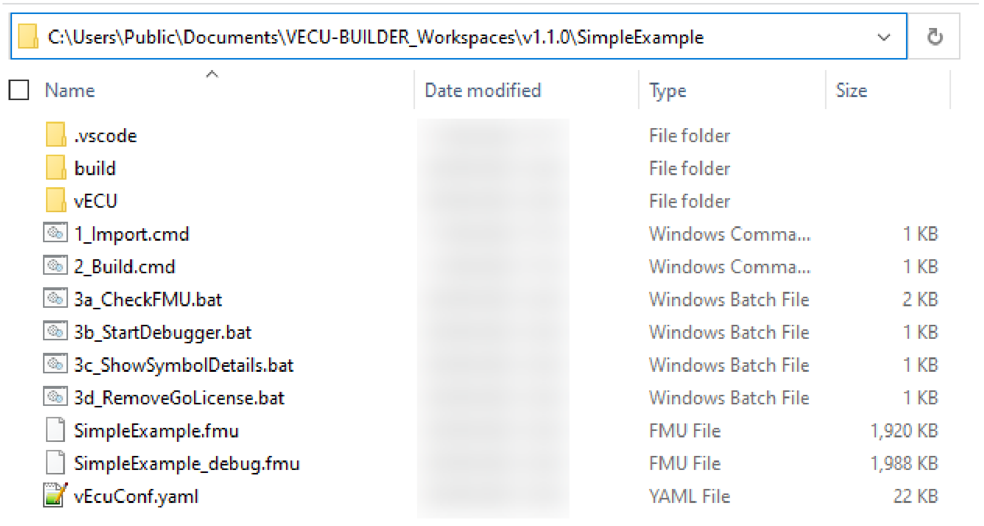


```
C:\WINDOWS\system32\cmd.exe
##########################################################################
### Building FMU                                                       ###
##########################################################################
[20:45:03] 1 of 5: Reading config: vEcuConf.yaml
[20:45:04] 2 of 5: Building inputs, outputs, parameters, tasks
[20:45:07] 3 of 5: Patching a2l files
[20:45:07] 4 of 5: Building fmu archives
[20:45:08] 5 of 5: Running scripts triggered through "after_build"
*** SUCCESS ***
```

Figure 12 - Building FMU completed

4.5 Workspace Content

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



C:\Users\Public\Documents\VECU-BUILDER_Workspaces\v1.1.0\SimpleExample

Name	Date modified	Type	Size
.vscode	[redacted]	File folder	
build	[redacted]	File folder	
vECU	[redacted]	File folder	
1_Import.cmd	[redacted]	Windows Comma...	1 KB
2_Build.cmd	[redacted]	Windows Comma...	1 KB
3a_CheckFMU.bat	[redacted]	Windows Batch File	2 KB
3b_StartDebugger.bat	[redacted]	Windows Batch File	1 KB
3c_ShowSymbolDetails.bat	[redacted]	Windows Batch File	1 KB
3d_RemoveGoLicense.bat	[redacted]	Windows Batch File	1 KB
SimpleExample.fmu	[redacted]	FMU File	1,920 KB
SimpleExample_debug.fmu	[redacted]	FMU File	1,988 KB
vEcuConf.yaml	[redacted]	YAML File	22 KB

Figure 13 - 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: command and shell scripts to perform the individual stages
    - `last_build_footprint` file: details of last performed build stage
    - `SymbolDetails.txt` file: symbols within your sources and their attributes

- `vECU` folder:
    - `buildArtifacts` folder: DLL file and its associated debug information
    - `CMake` folder: CMake project artifacts
    - `imported` folder: all imported artifacts
    - `CMakeLists.txt` file: set of directives and instructions for building your sources

- `1_Import.cmd`
  command script to trigger the Importing Files and Folders stage
- `2_Build.cmd`
  command script to trigger the Building the vECU stage

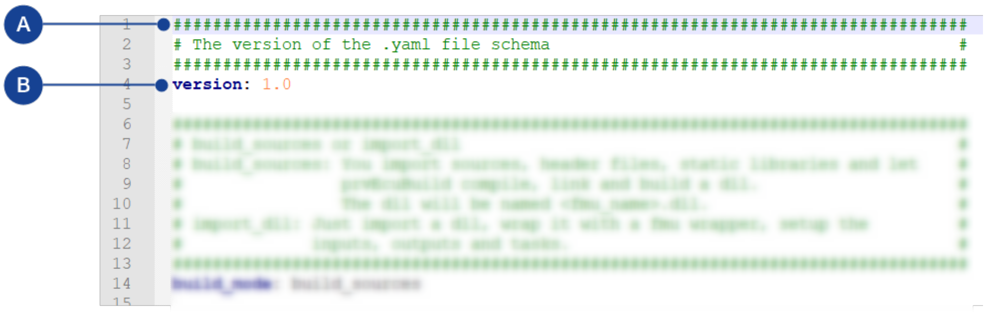
- `3a_CheckFMU.bat`
  batch file to invoke the FMU Checker and inspect the vECU outputs
- `3b_StartDebugger.bat` only present if build_tool is set to one MSVC
  batch file to invoke the configured MSVC as debugger
- `3c_ShowSymbolDetails.bat`
  batch file to invoke Notepad++ and display the Symbol Details
- `3d_RemoveGoLicense.bat` only relevant if vECU was built with GO-license
  batch file to remove the GO license from the vECU

- `SimpleExample.fmu`
  release version of your vECU, for more details see Simple Example
- `SimpleExample_debug.fmu`
  debug version of your vECU, for more details see Simple Example
- `vEcuConf.yaml`
  the YAML configuration file, for more details see Configuration

4.6 Configuration

The YAML file contains the configurations for the import and build process as well as for the vECU itself. It is the only configuration you need to create and maintain.

The YAML file is divided into several sections, each section configuring a particular attribute. You are guided through the YAML file with comments on each section and configuration attributes. Every section is structured in a standardized way:



```
################################################################################
# The version of the .yaml file schema                                         #
################################################################################
version: 1.0

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

A: comment with information on the corresponding section

B: configuration attributes and values

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

- **`version`**

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

- **`build_mode`**

  You can select between 2 modes:

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

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

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

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

- **`fmu_name`**

  Enter the name of you vECU.

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

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

- **`import_into_project`**

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

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

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

  Environment variables can be used like this:

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

- `import_external_vecu_dll.`
  The DLL needs to contain the symbol information.
- `import_external_vecu_dll`
  Only needed if you selected `import_dll` as `build_mode`.

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

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

  `dll_name`: The name of the DLL. A corresponding PDB file with the same file name must exist.

  `get_updates_from`: If VECU-BUILDER can find a DLL and PDB in this folder, the imported DLL will be updated.

  Environment variables can be used like this:

  `'${SystemDrive}\Sandbox'.`
- `architecture`
  Specify the architecture.

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

  In case you are importing an DLL pre-compiled for either 32bit or 64bit architecture, this attribute must be set to the same.
- `xcp_slave`
  Enter the port and IP address of the XCP Slave to be setup in your vECU.

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

NOTE

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

- `operating_system`
  Enter the operating system. Currently only Windows is supported.
- `build_tool`
  Enter your preferred build tool.

  Several MSVC versions and MinGW Makefiles are supported.

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

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

  These artefacts are stored in the "`vECU\CMake`" folder in your workspace.
- `cmake_generator_toolset`
  Define which toolset should be used by CMake during the build process.

  For more details, see CMAKE_GENERATOR_TOOLSET.
- `inputs`, `outputs`, `parameters`, `locals`
  Enter the variables you wish to expose as ports of your FMU.

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

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

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

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

NOTE

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

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

  The initial data is virtually flashed into memory during initialization. It simulates a part of the NVRAM (non-volatile RAM).

  `source`: Where to get the file.

  `destination`: Where to store the file relative to the resources folder of the FMU (optional).

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

- **`eeprom`**

  Specify the eeprom simulation attributes.

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

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

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

  `sync`: This must be a UNC pathname. When the vECU is initializing, this file is copied to the 'destination', if it exists. When the vECU terminates, the updated file in 'destination' is copied to the 'sync' location (optional).

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

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

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

- **`tasks`**

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

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

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

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

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

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

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

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

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

- **`redirect_function_calls`**

  Enter the names functions to be replaced and their substitutes.

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

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

  `substitute_function`: The function name of the function that substitutes the replaced function.

- **`build_include_filters`** & **`build_exclude_filters`**

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

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

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

- **`assembly_list_files`**

  Specify your assembly list files for the build process.

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

  If no assembly list files are configured, all sources are compiled.

- **`additional_include_directories`** & **`additional_defines`**

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

  These values are passed to the preprocessor. This is useful if you need to set/unset some defines to adapt them to the new PC target.

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

> **NOTE**
>
> The following limitations apply to filename paths, command-line and response-file lengths in the Windows environment.
>
> - Filename paths cannot be longer than MAX_PATH (260) characters.
> - Command-line lengths cannot be longer than 32,768 characters.
> - Response-file lengths cannot be longer than 131,072 characters.

- **`additional_compile_flags`**

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

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

  The flags are written into the CMakeLists.

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

- **`additional_static_libraries`**

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

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

- **`environment_variables`**

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

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

  These variables can be configured and modified in one location and can be accessed from scripts and configuration files.

  Process-level environment variable of `VECUBUILDER_WORKSPACE` is created automatically during the build process with its value pointing to the current workspace.

- **`additional_scripts`**

  Define your additional scripts for execution.

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

  You can utilize these to copy or modify files, add files to the FMU archive, parse files, etc. You may use Python scripts or .bat and .cmd scripts.

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

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

  `command_line`: Full absolute path to the interpreter.

  `trigger`: Select from four available options:

  `before_import`

  `after_import`

  `before_build`

  `after_build`

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

- **`patch_a2l_file`**

  Enter the name of your A2L file to be patched.

  An A2L File is required to connect an MCD tool such as INCA to the running vECU. The A2L file needs to be located in the folder: “`vEcu/imported`".

  VECU-BUILDER will update the memory addresses of all measurements and characteristics in the provided A2L file. The original A2L file is renamed by appending `.bak` to its name.

  For more details, see A2L File Patching.

- **`debug_hook`**

  Specify whether to enable or disable a debug hook.

  When enabled, the FMU execution is interrupted when the FMU is instantiated until a debugger is attached.

  For more details, see Debugging vECU.

- **`additional_link_flags`**

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

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

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

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

5 Exploring the Examples

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

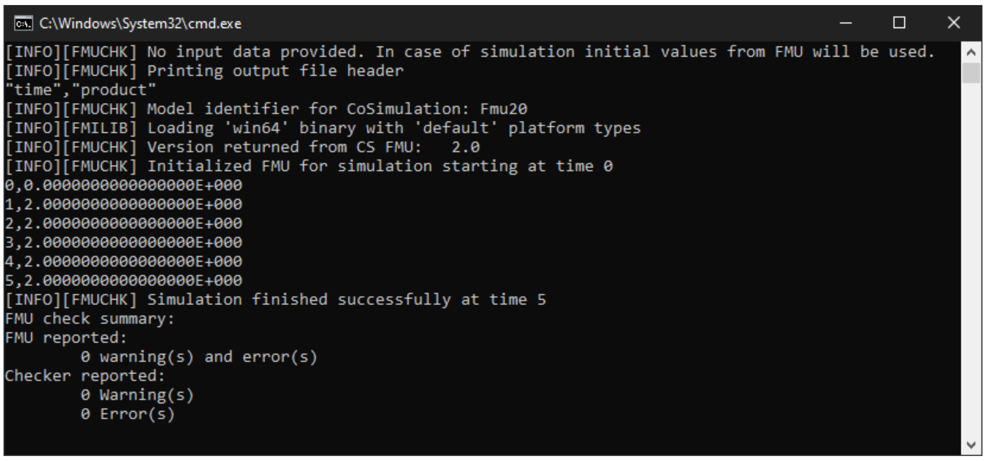
5.1 Simple Example

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

5.1.1 FMU Checker

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

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



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

Figure 14 - FMU Checker output

5.1.2 Difference Between Debug and Release vECUs

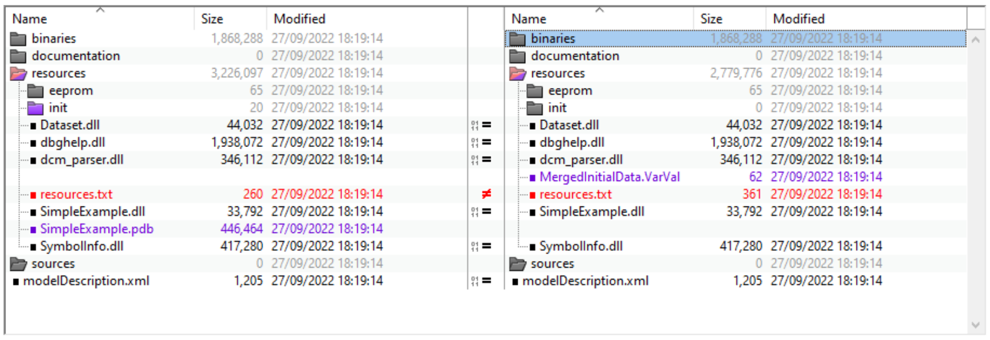
You find two FMUs in this workspace, one named `SimpleExample.fmu` (which will be referred to as ‘release vECU’) and the other one named `SimpleExam-ple_debug.fmu` (which will be referred to as ‘debug vECU’).

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

The functional behavior of both vECUs is identical.

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

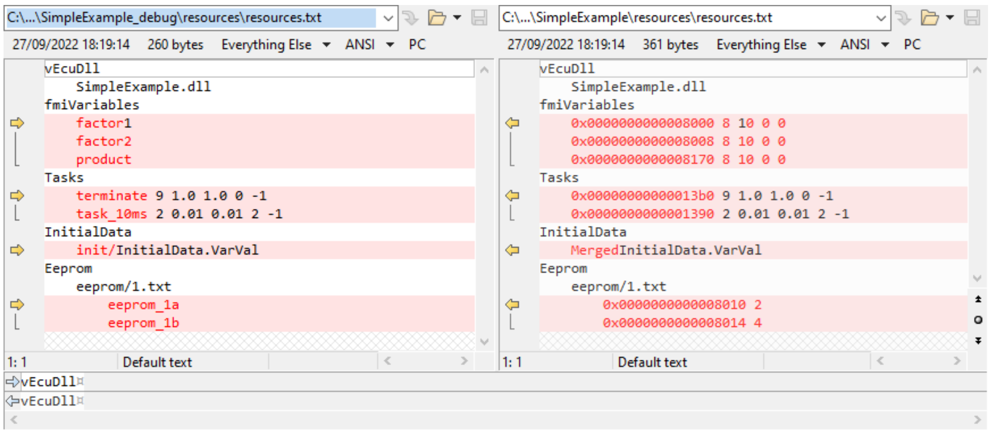
When you compare the two extracted folders, you will notice that the main difference is in the resources folder.



Name	Size	Modified		Name	Size	Modified
binaries	1,868,288	27/09/2022 18:19:14		binaries	1,868,288	27/09/2022 18:19:14
documentation	0	27/09/2022 18:19:14		documentation	0	27/09/2022 18:19:14
resources	3,226,097	27/09/2022 18:19:14		resources	2,779,776	27/09/2022 18:19:14
eeprom	65	27/09/2022 18:19:14		eeprom	65	27/09/2022 18:19:14
init	20	27/09/2022 18:19:14		init	0	27/09/2022 18:19:14
Dataset.dll	44,032	27/09/2022 18:19:14	=	Dataset.dll	44,032	27/09/2022 18:19:14
dbghelp.dll	1,938,072	27/09/2022 18:19:14	=	dbghelp.dll	1,938,072	27/09/2022 18:19:14
dcm_parser.dll	346,112	27/09/2022 18:19:14	=	dcm_parser.dll	346,112	27/09/2022 18:19:14
				MergedInitialData.VarVal	62	27/09/2022 18:19:14
resources.txt	260	27/09/2022 18:19:14	≠	resources.txt	361	27/09/2022 18:19:14
SimpleExample.dll	33,792	27/09/2022 18:19:14	=	SimpleExample.dll	33,792	27/09/2022 18:19:14
SimpleExample.pdb	446,464	27/09/2022 18:19:14				
SymbolInfo.dll	417,280	27/09/2022 18:19:14	=	SymbolInfo.dll	417,280	27/09/2022 18:19:14
sources	0	27/09/2022 18:19:14		sources	0	27/09/2022 18:19:14
modelDescription.xml	1,205	27/09/2022 18:19:14	=	modelDescription.xml	1,205	27/09/2022 18:19:14

Figure 15 - Comparison of debug and release vECU

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



C:\...\SimpleExample_debug\resources\resources.txt
27/09/2022 18:19:14 260 bytes Everything Else ANSI PC

```
vEcuDll
    SimpleExample.dll
fmiVariables
    factor1
    factor2
    product
Tasks
    terminate 9 1.0 1.0 0 -1
    task_10ms 2 0.01 0.01 2 -1
InitialData
    init/InitialData.VarVal
Eeprom
    eeprom/1.txt
        eeprom_1a
        eeprom_1b
```

1: 1 Default text

C:\...\SimpleExample\resources\resources.txt
27/09/2022 18:19:14 361 bytes Everything Else ANSI PC

```
vEcuDll
    SimpleExample.dll
fmiVariables
    0x0000000000008000 8 10 0 0
    0x0000000000008008 8 10 0 0
    0x0000000000008170 8 10 0 0
Tasks
    0x00000000000013b0 9 1.0 1.0 0 -1
    0x0000000000001390 2 0.01 0.01 2 -1
InitialData
    MergedInitialData.VarVal
Eeprom
    eeprom/1.txt
        0x0000000000008010 2
        0x0000000000008014 4
```

1: 1 Default text

```
vEcuDll¤
vEcuDll¤
```

Figure 16 - Comparison of Resources.txt

5.1.3 Features to Explore in the Simple Example Workspace

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

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

5.2 BCU Example

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

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

   `C:\ProgramData\ETAS\VECU-BUILDER\Examples_v1.1.0\BCU`

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

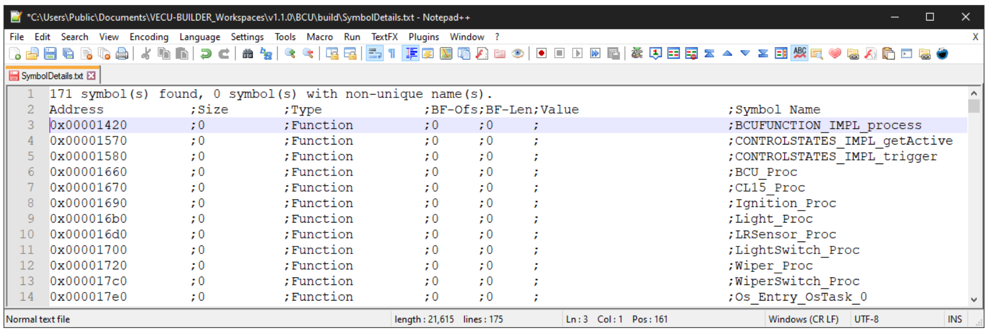
5.2.1 Show Symbol Information

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

2. Run the command:

   `3c_ShowSymbolDetails.bat`

⇨ A text editor window opens, and symbol details are shown.


*C:\Users\Public\Documents\VECU-BUILDER_Workspaces\v1.1.0\BCU\build\SymbolDetails.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run TextFX Plugins Window ?
SymbolDetails.txt

```
171 symbol(s) found, 0 symbol(s) with non-unique name(s).
Address           ;Size       ;Type             ;BF-Ofs;BF-Len;Value                       ;Symbol Name
0x00001420        ;0          ;Function         ;0     ;0     ;                            ;BCUFUNCTION_IMPL_process
0x00001570        ;0          ;Function         ;0     ;0     ;                            ;CONTROLSTATES_IMPL_getActive
0x00001580        ;0          ;Function         ;0     ;0     ;                            ;CONTROLSTATES_IMPL_trigger
0x00001660        ;0          ;Function         ;0     ;0     ;                            ;BCU_Proc
0x00001670        ;0          ;Function         ;0     ;0     ;                            ;CL15_Proc
0x00001690        ;0          ;Function         ;0     ;0     ;                            ;Ignition_Proc
0x000016b0        ;0          ;Function         ;0     ;0     ;                            ;Light_Proc
0x000016d0        ;0          ;Function         ;0     ;0     ;                            ;LRSensor_Proc
0x00001700        ;0          ;Function         ;0     ;0     ;                            ;LightSwitch_Proc
0x00001720        ;0          ;Function         ;0     ;0     ;                            ;Wiper_Proc
0x000017c0        ;0          ;Function         ;0     ;0     ;                            ;WiperSwitch_Proc
0x000017e0        ;0          ;Function         ;0     ;0     ;                            ;Os_Entry_OsTask_0
```

Normal text file   length : 21,615   lines : 175   Ln : 3   Col : 1   Pos : 161   Windows (CR LF)   UTF-8   INS

Figure 17 - Symbol Details of BCU example

5.2.2 A2L File Patching

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

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

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

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

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

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

5.2.3 Features to Explore in the BCU Workspace

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

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

6 Controlling VECU-BUILDER

6.1 Manual Interaction

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

This control method is recommended for all new users. Only this method has been used in this document.

6.2 Command-Line Interface

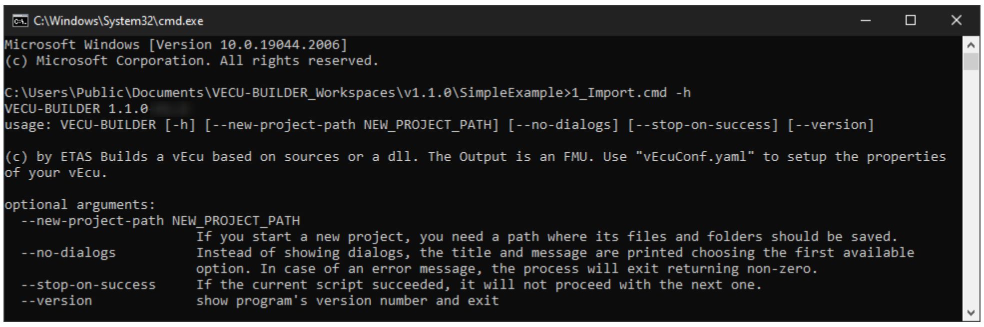
Besides the manual Interaction method, you can also operate VECU-BUILDER via a command-line interface (CLI). VECU-BUILDER is a CLI native application, and the command and batch scripts allow manual interaction.

The following arguments exist:

- `--new-project-path`: Path where the workspace is to be created.
- `--no-dialogs`: Suppress all dialogs and always select the default option.
- `--stop-on-success`: Prevent automatic forwarding to the next stage (create workspace, import, build).
- `--version`: Print the version information.
- `-h`: Print list of all optional arguments.

To see all CLI optional arguments and their description

1. Open a terminal (type cmd in the Windows Explorer) in your workspace.
2. Run the command:
   `1_Import.cmd -h`



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Public\Documents\VECU-BUILDER_Workspaces\v1.1.0\SimpleExample>1_Import.cmd -h
VECU-BUILDER 1.1.0[redacted]
usage: VECU-BUILDER [-h] [--new-project-path NEW_PROJECT_PATH] [--no-dialogs] [--stop-on-success] [--version]

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

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

Figure 18 - CLI optional arguments

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

The following table gives an overview of which script uses which arguments:

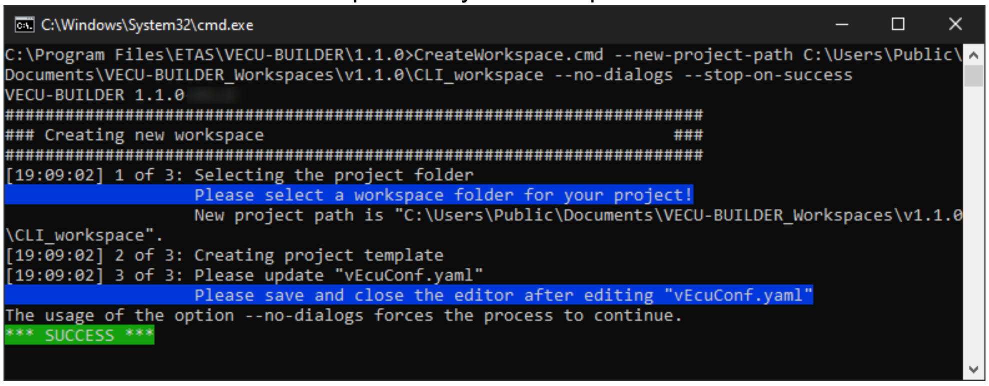
argument	`CreateWorkspace.cmd`	`1_Import.cmd`	`2_Build.cmd`
`--new-project-path`	Used (required)	Ignored	Ignored
`--no-dialogs`	Used (optional)	Used (optional)	Used (optional)
`--stop-on-success`	Used (optional)	Used (optional)	Ignored
`--version`	Used (optional)	Used (optional)	Used (optional)
`-h`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

Table 1 – Mapping of CLI arguments to scripts

To build the SimpleExample via two command lines

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

1. Open a terminal
2. Go to the installation location
   `cd %VECUBUILDER_HOME%`
3. Run the command:
   `CreateWorkspace.cmd`
   with the arguments
   `--new-project-path <destination>`
   `--no-dialogs`
   `--stop-on-success`
   where `<destination>` points to your workspace folder.



```
C:\Program Files\ETAS\VECU-BUILDER\1.1.0>CreateWorkspace.cmd --new-project-path C:\Users\Public\
Documents\VECU-BUILDER_Workspaces\v1.1.0\CLI_workspace --no-dialogs --stop-on-success
VECU-BUILDER 1.1.0 [redacted]
######################################################################
### Creating new workspace                                         ###
######################################################################
[19:09:02] 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\v1.1.0
\CLI_workspace".
[19:09:02] 2 of 3: Creating project template
[19:09:02] 3 of 3: Please update "vEcuConf.yaml"
                   Please save and close the editor after editing "vEcuConf.yaml"
The usage of the option --no-dialogs forces the process to continue.
*** SUCCESS ***
```

Figure 19 - Workspace creation via CLI

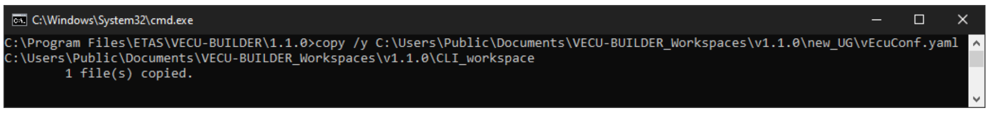
NOTE

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:

4. Run the command:
   `copy /y <source> <destination>`



C:\Windows\System32\cmd.exe

```
C:\Program Files\ETAS\VECU-BUILDER\1.1.0>copy /y C:\Users\Public\Documents\VECU-BUILDER_Workspaces\v1.1.0\new_UG\vEcuConf.yaml
C:\Users\Public\Documents\VECU-BUILDER_Workspaces\v1.1.0\CLI_workspace
        1 file(s) copied.
```

Figure 20 - Copy your project specific YAML file

NOTE

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

To continue building your workspace:

5. Navigate to this new workspace by running the command:
   `cd <destination>`
6. Run the command:
   `1_Import.cmd --no-dialogs`

7 Debugging vECU

VECU-BUILDER provides features to debug the vECU. One such feature is the `debug_hook`, which can be enabled in the YAML file. vECUs built with this attribute activated enter their instantiation and wait for a debugger to be attached before entering their execution.

The following table summarizes the possible combinations of build tool and debugger in terms of debugging:

		Debugger			
		VS Code	VS 2015	VS 2017	VS 2019
Build tool	MinGW	**recommended**	unavailable	experimental	recommended
	VS 2015	experimental	recommended	possible	possible
	VS 2017	experimental	unavailable	recommended	possible
	VS 2019	experimental	unavailable	unavailable	**recommended**

Table 2 - Debugging possibilities

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

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

7.1 Debugging with Visual Studio 2019

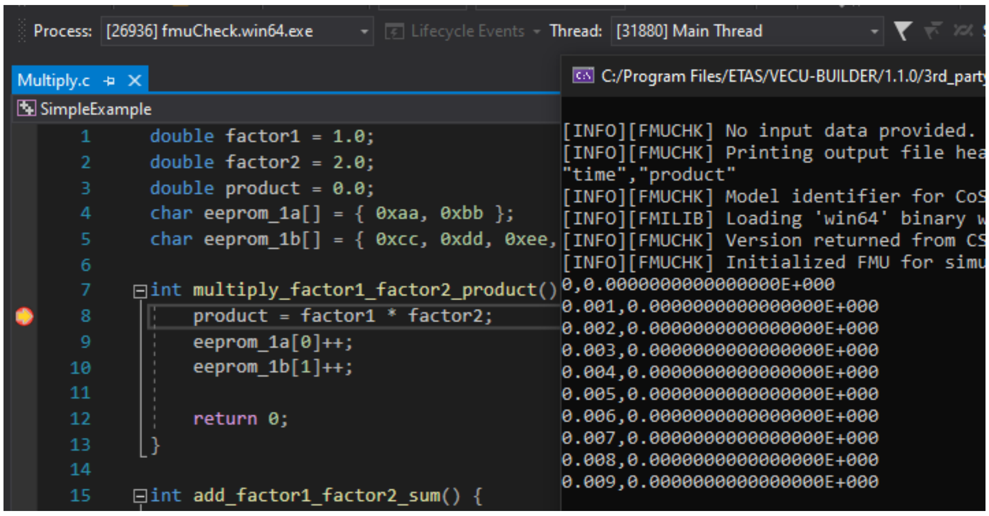
This chapter describes how to debug vECU built with Visual Studio 2019 using Visual Studio 2019 as the debugger.

1. Navigate to your workspace.
2. Execute the `3b_StartDebugger.bat` file.

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

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

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



Process: [26936] fmuCheck.win64.exe Lifecycle Events Thread: [31880] Main Thread

Multiply.c — SimpleExample

```
double factor1 = 1.0;
double factor2 = 2.0;
double product = 0.0;
char eeprom_1a[] = { 0xaa, 0xbb };
char eeprom_1b[] = { 0xcc, 0xdd, 0xee,

int multiply_factor1_factor2_product()
    product = factor1 * factor2;
    eeprom_1a[0]++;
    eeprom_1b[1]++;

    return 0;
}

int add_factor1_factor2_sum() {
```

C:/Program Files/ETAS/VECU-BUILDER/1.1.0/3rd_party

```
[INFO][FMUCHK] No input data provided.
[INFO][FMUCHK] Printing output file hea
"time","product"
[INFO][FMUCHK] Model identifier for CoS
[INFO][FMILIB] Loading 'win64' binary w
[INFO][FMUCHK] Version returned from CS
[INFO][FMUCHK] Initialized FMU for simu
0,0.0000000000000000E+000
0.001,0.0000000000000000E+000
0.002,0.0000000000000000E+000
0.003,0.0000000000000000E+000
0.004,0.0000000000000000E+000
0.005,0.0000000000000000E+000
0.006,0.0000000000000000E+000
0.007,0.0000000000000000E+000
0.008,0.0000000000000000E+000
0.009,0.0000000000000000E+000
```

Figure 21 - VS 2019 Debugger attached

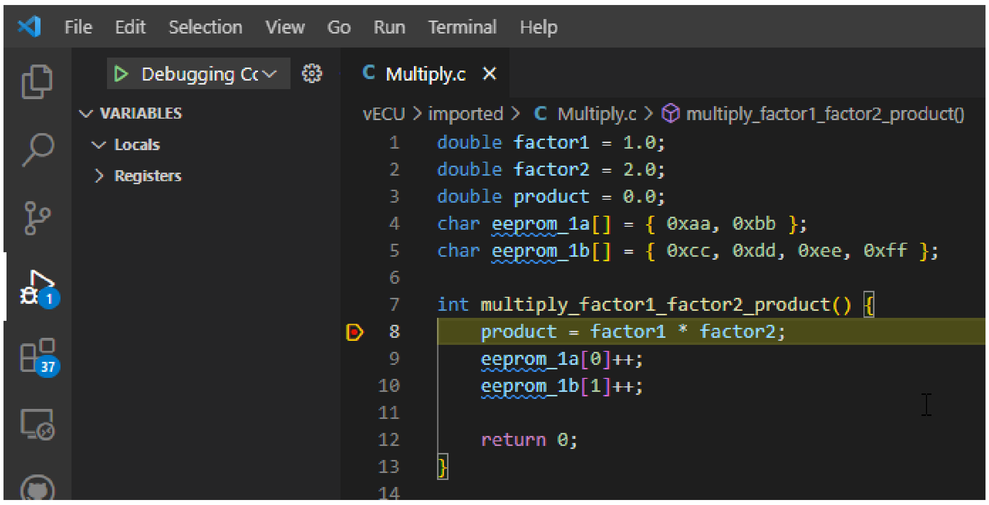
7.2 Debugging with Visual Studio Code

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

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

   ⇨ Visual Studio Code opens.
3. Navigate to where you want to start the debugging and place a breakpoint there.
4. In the menu panel on the left click **Run and Debug**.
5. Click **Start Debugging** (**F5**).

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



```c
double factor1 = 1.0;
double factor2 = 2.0;
double product = 0.0;
char eeprom_1a[] = { 0xaa, 0xbb };
char eeprom_1b[] = { 0xcc, 0xdd, 0xee, 0xff };

int multiply_factor1_factor2_product() {
    product = factor1 * factor2;
    eeprom_1a[0]++;
    eeprom_1b[1]++;

    return 0;
}
```

Figure 22 - VS Code Debugger attached

8 Migrating Workspaces of VECU-BUILDER V1.0

If you have created projects with VECU-BUILDER V1.0 some files of your workspace need to be changed and migrated for using VECU-BUILDER V1.1.

This task is divided into the following main steps:

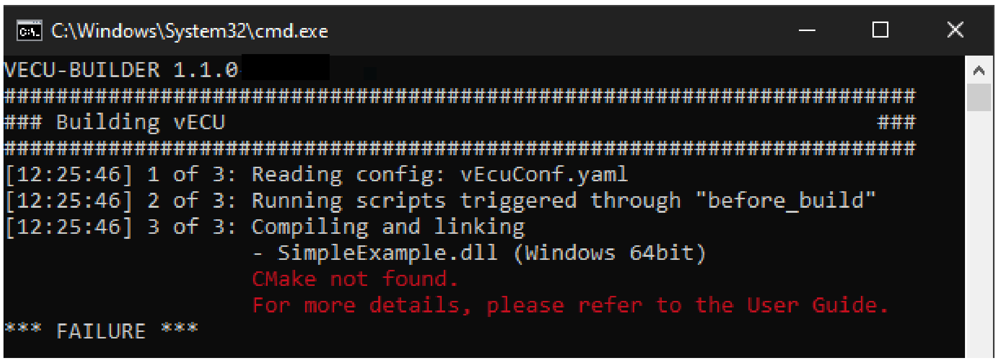
1. Create new workspace with VECU-BUILDER V1.1.
   - See Creating a New Workspace.
2. Transfer the project-specific config content of the old YAML file into the new one.
   - The `vEcuConf.yaml` of the newly created workspace opens in Notepad++ as part of the workspace creation process.
3. Copy the project-specific content into the new YAML file.
   - TIP: To identify the deviations easily use a comparison tool.
4. Save the changes and close the new YAML file, the creation of the workspace will continue.
5. When VECU-BUILDER asks to continue with the Build stage, select **No**.
6. Copy the contents of the '`imported`' folder in the old workspace to the '`imported`' folder in the new workspace.
   - If you used the default folder when creating your workspace with VECU-BUILDER V1.0, it is located here:

     `C:\ETASData\VECU-BUILDER v1.0\vEcuBuild_Workspaces.`
7. The vECU can now be built and used as before with VECU-BUILDER V1.0. For the build use `2_Build.cmd` of your new created workspace.

9 Troubleshooting

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

9.1 CMake not found



```
VECU-BUILDER 1.1.0 [redacted]
#######################################################################
### Building vECU                                                   ###
#######################################################################
[12:25:46] 1 of 3: Reading config: vEcuConf.yaml
[12:25:46] 2 of 3: Running scripts triggered through "before_build"
[12:25:46] 3 of 3: Compiling and linking
                   - SimpleExample.dll (Windows 64bit)
                   CMake not found.
                   For more details, please refer to the User Guide.
*** FAILURE ***
```

Figure 24 – CMake not found error

Possible Reason

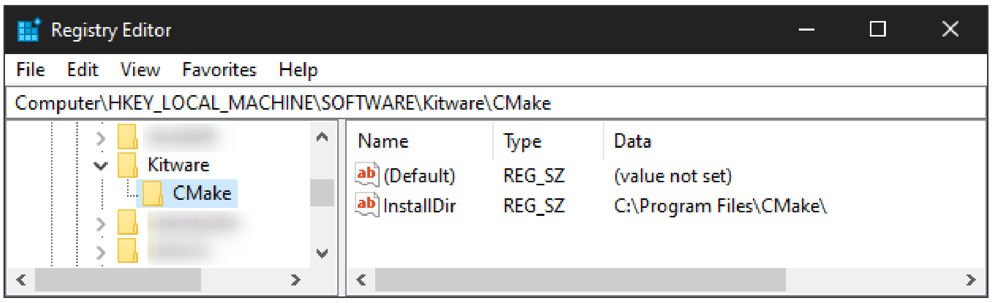
A CMake installation is required (see System Requirements) and must be registered properly. This registry entry is used to locate the CMake installation, if it does not exist, the build fails.

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

Possible Solution

Ensure the following:

- CMake is installed (version 3.15 or higher).
- `Kitware` and `CMake` keys exist in the Windows Registry.
- The CMake registry key
  `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Kitware\CMake`
  contains the string value `InstallDir` pointing to the CMake installation path:



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

Figure 25 – Windows Registry with Kitware\CMake registry key

9.2 Notepad++ Does Not Open During Workspace Creation Process

Possible Reason

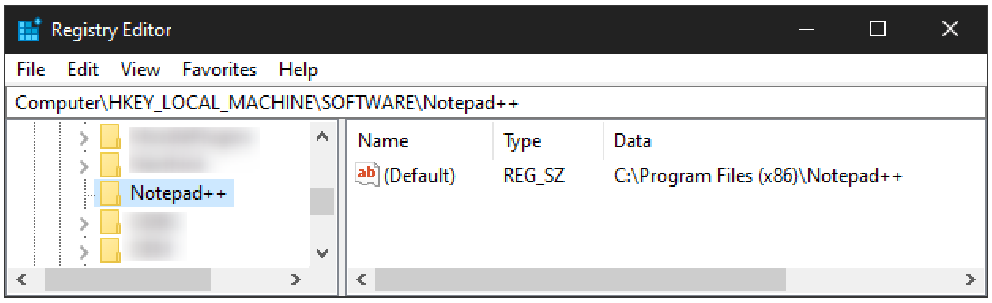
Notepad++ is the recommended text editor to be used along with VECU-BUILDER (see System Requirements). For it to work as intended, it must be installed and registered properly.

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

Possible Solution

Ensure the following:

- Notepad++ is installed.
- Notepad++ key exists in the Windows Registry.
- The Notepad++ registry key
  `Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Notepad++`
  contains the string value `(Default)` pointing to the Notepad++ installation path:



Registry Editor

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Notepad++

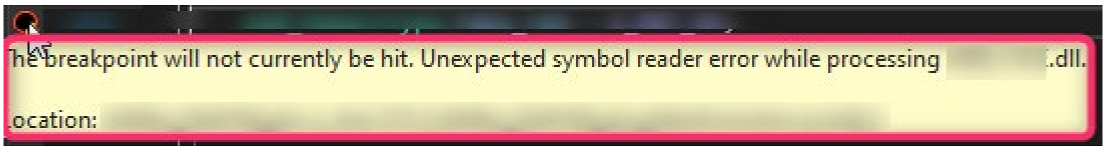
Name	Type	Data
(Default)	REG_SZ	C:\Program Files (x86)\Notepad++

Figure 26 - Windows Registry with Notepad++ registry key

9.3 Some Breakpoints Not Being Hit

Possible Reason

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



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

ocation: [redacted]

Figure 27 – Breakpoint not being hit

Possible Solution

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

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

9.4 (SymbolInfo.dll) The *.die File Is too Large to Load

Possible Reason

The amount of symbol information (stored in the `*.die` file) does not fit into the memory provided by the operating system.

32bit processes are limited to less than 4 GB of RAM, this might be insufficient to load a large `*.die` file.

Possible Solution

Use 64bit architecture when building the vECU (configured in the YAMl file) and use a PC with sufficient memory.

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/en/hotlines.php

ETAS Headquarters

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