DRIVING EMBEDDED EXCELLENCE

ETAS VECU-BUILDER V1.0

User Guide

www.etas.com

Copyright

ETAS VECU-BUILDER V1.0 – User Guide R02 EN – 06.2022

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 have to 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 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, the access to calibration and measurement variables as A2L and an executable model as DLL.

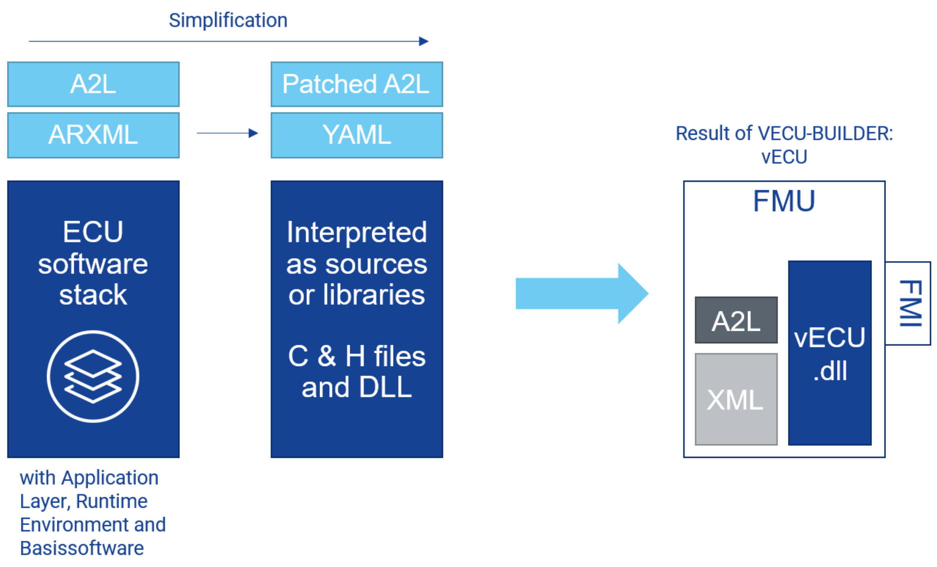

Simplification
A2L
ARXML
Patched A2L
YAML
ECU software stack
with Application Layer, Runtime Environment and Basissoftware
Interpreted as sources or libraries
C & H files and DLL
Result of VECU-BUILDER: vECU
FMU
A2L
XML
vECU .dll
FMI

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 Workflow vECU Creation Process

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 for x86, 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 notation of the variables how they are written in C (like `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 vECU in Co-Simulation environment
- Run and test the vECU in an experiment environment
- Measurement and calibration of vECU
- Debugging with source code editor

2.4 Limitations

VECU-BUILDER supports the generation of Level-1, Level-2, and Level-3 vECUs according to the Prostep Definition of vECUs: https://www.prostep.org/fileadmin/downloads/WhitePaper_V-ECU_2020_05_04-EN.pdf Level-4 vECUs, i.e., hex-files for a specific target, are not supported.

3 Installation

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

3.1 Preparation

Prior to the installation, check that your computer meets the system requirements (see Release Notes "System Prerequisites"). Depending on the operating system used and network connection, you must ensure that you have the required user rights.

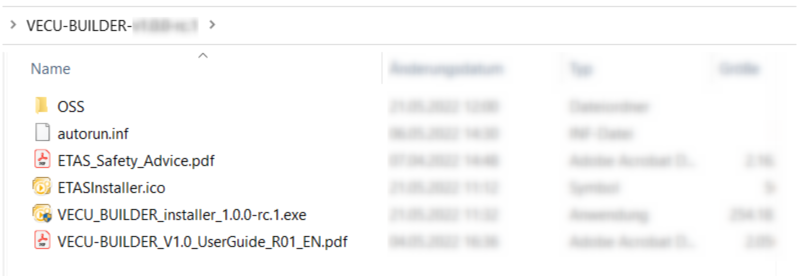
NOTE

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

3.2 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).

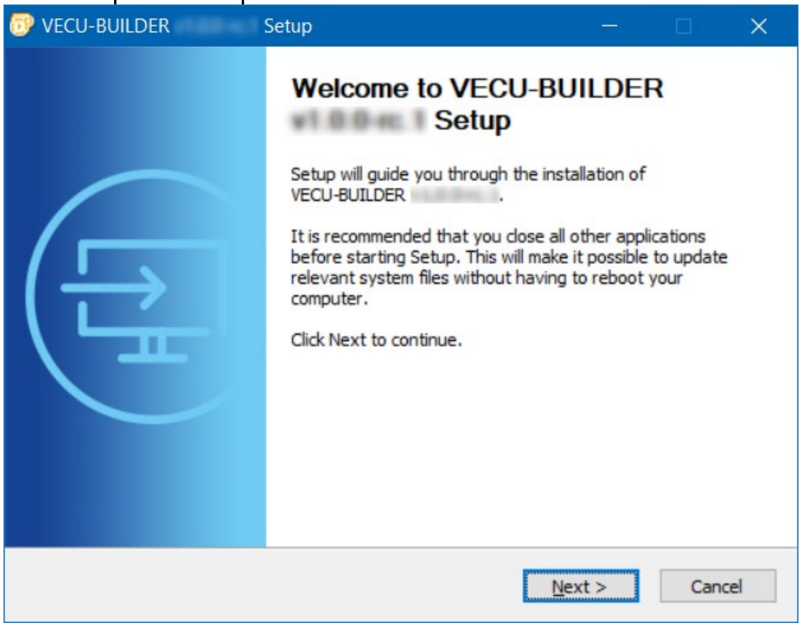


```
> VECU-BUILDER-[illegible] >

Name
OSS
autorun.inf
ETAS_Safety_Advice.pdf
ETASInstaller.ico
VECU_BUILDER_installer_1.0.0-rc.1.exe
VECU-BUILDER_V1.0_UserGuide_R01_EN.pdf
```

3.3 Installing

1. Go to the directory where the installation file is located and double-click on the `VECU_BUILDER_installer_1.0.0.exe` file.

⇨ The Setup Wizard opens.



Welcome to VECU-BUILDER [illegible] Setup

Setup will guide you through the installation of VECU-BUILDER [illegible].

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

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 accept the Safety Advice**.

⇨ The "Installation Path" window opens.

6. Accept the default folder by clicking **Next** or click **Browse** to select another location.

⇨ The "Ready to Install" window opens.

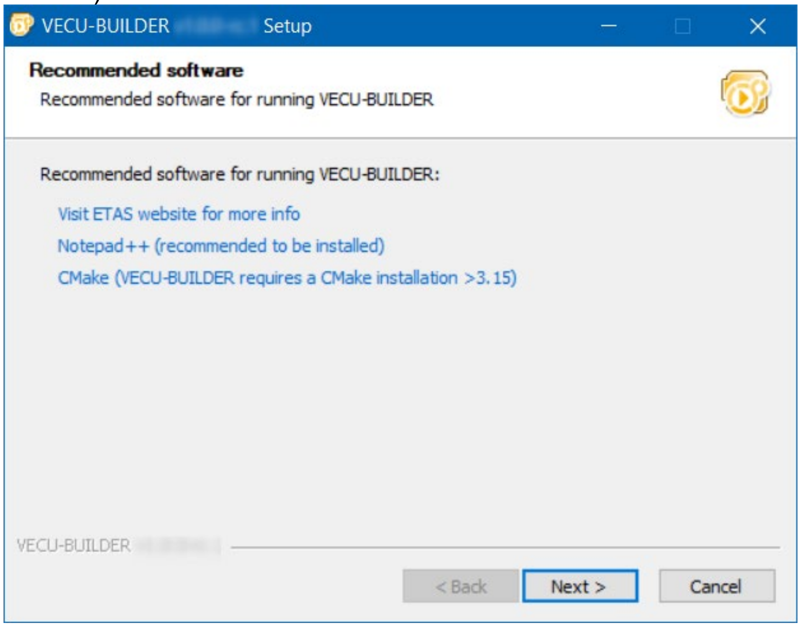
7. If you want to change settings, click **Back**.
   If you want to start the installation, click **Install**.

⇨ The installation is performed. A progress indicator shows how the installation is progressing. When the installation is completed, it is shown in the progress bar.

8. Click **Next**.

⇨ The "Installation Complete" window opens.

9. If necessary: Download Notepad++ and CMake (see links in window).



Recommended software
Recommended software for running VECU-BUILDER

Recommended software for running VECU-BUILDER:

Visit ETAS website for more info
Notepad++ (recommended to be installed)
CMake (VECU-BUILDER requires a CMake installation >3.15)

NOTE

The installation of CMake and Notepad++ is a prerequisite for using VECU-BUILDER.

10. Click **Next**.

⇨ The "Completing VECU-BUILDER Setup" window opens. If you wish, you can directly open the User Guide.

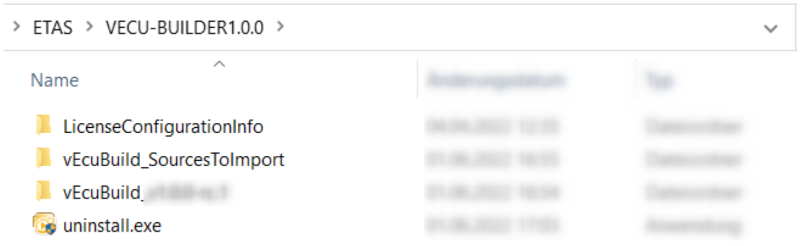
11. Optionally: Activate **Open VECU-BUILDER documentation** checkbox to open the User Guide for example.

12. Click **Finish**.

⇨ The installation is completed. VECU-BUILDER can be used.

3.4 Installed Files and Folders

After installation the following files and folders are available.



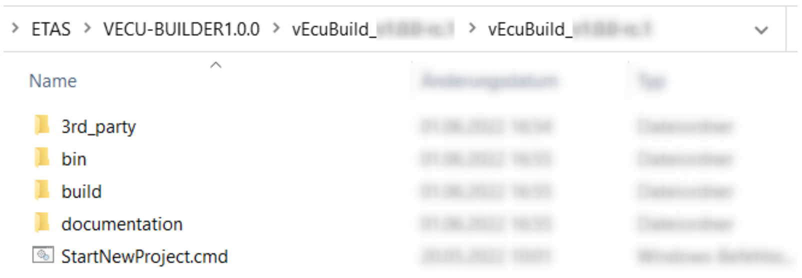
Name
LicenseConfigurationInfo
vEcuBuild_SourcesToImport
vEcuBuild_
uninstall.exe

Figure 2: Installation content of VECU-BUILDER

The core content can be found in the vEcuBuild_vx.x.x folder.

- 3rd_party: contains the third party tools FMI Compliance Checker (FMU Checker) and MinGW.

- bin: contains DLL and EXE files for the build process. These files are important for the build.
- build: contains some templates, resources, and scripts for the build process. These files are important for the build.
- documentation: contains VECU-BUILDER User Guide, OSS Attribution and the ETAS Safety Advice.
- StartNewProject.cmd to create a new project. After executing the CMD you will be guided through the process step by step.



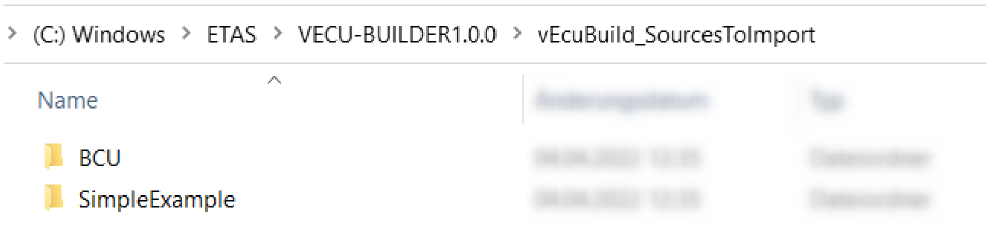
> ETAS > VECU-BUILDER1.0.0 > vEcuBuild_[illegible] > vEcuBuild_[illegible]

Name
3rd_party
bin
build
documentation
StartNewProject.cmd

Figure 3: Core content of VECU-BUILDER

You can find 2 examples in the `vEcuBuild_SourcesToImport` folder.

- BCU (Body Control Unit)
- Simple Example



> (C:) Windows > ETAS > VECU-BUILDER1.0.0 > vEcuBuild_SourcesToImport

Name
BCU
SimpleExample

Figure 4: Examples delivered with VECU-BUILDER installation

See also Get familiar with VECU-BUILDER.

3.5 Licensing

The use of VECU-BUILDER is protected by electronic licensing. Valid licenses are necessary to install 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.

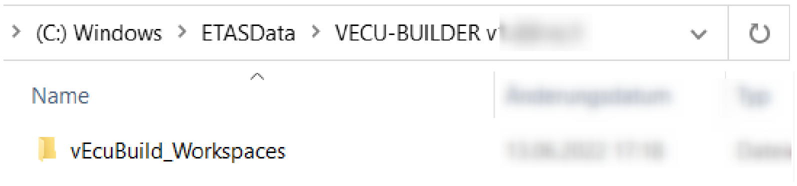
3.6 Uninstalling

1. Open the location where you installed VECU-BUILDER.
   When you used the default location, it can be found in C:\ETAS\VECU-BUILDER.
2. Execute the `uninstall.exe` with double-click.

4 Getting Started

This section helps beginners to learn how to create a first vECU.

As workspace location we recommend using the following path:
C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces



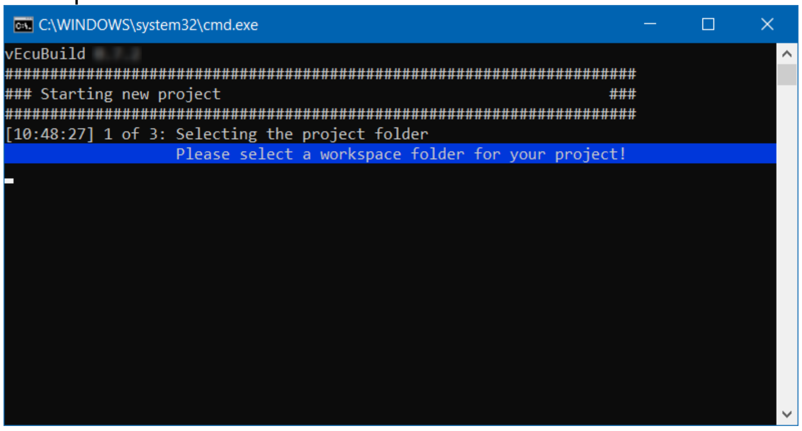
This folder is created during the installation process.

4.1 Create your First virtual ECU

To create your first vECU with VECU-BUILDER we guide you through the process step by step.

1. Go to the location where you installed VECU-BUILDER.
   When you used the default location, it can be found in
   C:\ETAS\VECU-BUILDER
2. Open folder `vEcuBuild_v1.0.0`.
3. Execute `StartNewProject.cmd` with double-click.

⇨ A console window is opened. You will be asked where your workspace should be saved.



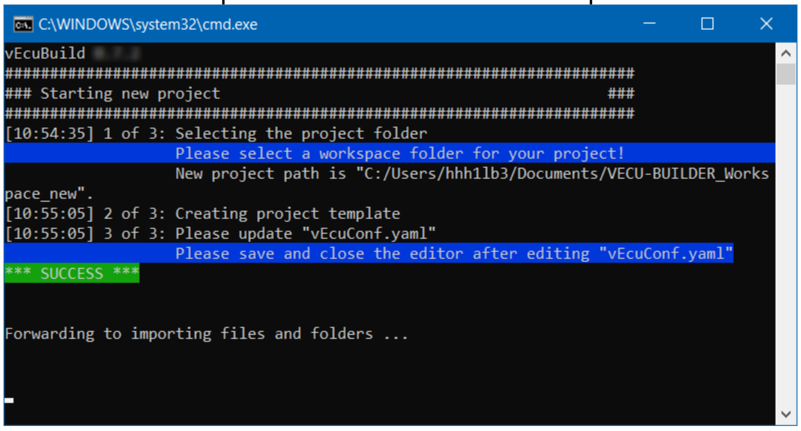
```
vEcuBuild
######################################################################
### Starting new project                                           ###
######################################################################
[10:48:27] 1 of 3: Selecting the project folder
                   Please select a workspace folder for your project!
```

4. Create a new folder or select a folder for your project workspace (C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces).

⇨ The vECUconf.yaml is opened with a Notepad++ instance.

5. Optionally: Configure your vECU by entering your parameters.
6. Close Notepad++.

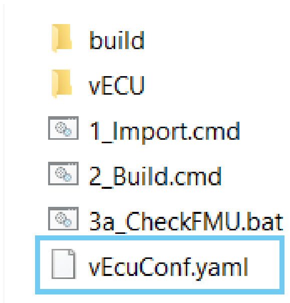
⇨ Files and folders specified in the YAML file are imported.



C:\WINDOWS\system32\cmd.exe

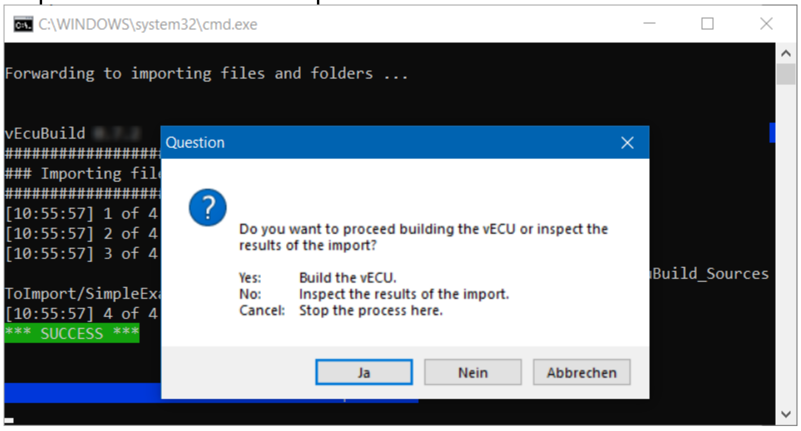
```
vEcuBuild [redacted]
#######################################################################
### Starting new project                                            ###
#######################################################################
[10:54:35] 1 of 3: Selecting the project folder
                   Please select a workspace folder for your project!
                   New project path is "C:/Users/hhh11b3/Documents/VECU-BUILDER_Works
pace_new".
[10:55:05] 2 of 3: Creating project template
[10:55:05] 3 of 3: Please update "vEcuConf.yaml"
                   Please save and close the editor after editing "vEcuConf.yaml"
*** SUCCESS ***


Forwarding to importing files and folders ...
```

The YAML file is saved in your selected workspace. In addition, a `build` folder, a `vECU` folder, 2 CMD files and a BAT file are created.



build
vECU
1_Import.cmd
2_Build.cmd
3a_CheckFMU.bat
vEcuConf.yaml

7. Answer the inquiry „Do you want to proceed building the vECU or inspect the result of the import“ with **Yes**.



C:\WINDOWS\system32\cmd.exe

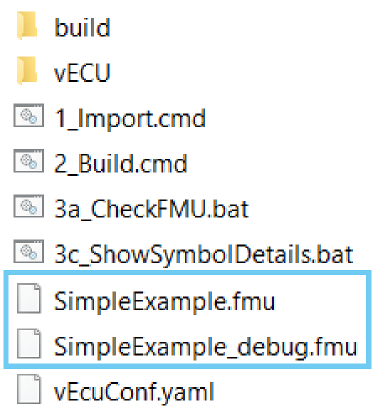
```
Forwarding to importing files and folders ...

vEcuBuild [redacted]
#################
### Importing fil
#################
[10:55:57] 1 of 4
[10:55:57] 2 of 4
[10:55:57] 3 of 4                                                   Build_Sources

ToImport/SimpleEx
[10:55:57] 4 of 4
*** SUCCESS ***
```

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.

⇨ The vECU is built as an FMU file and can be found in your project folder you selected or created in Step 4.


build
vECU
1_Import.cmd
2_Build.cmd
3a_CheckFMU.bat
3c_ShowSymbolDetails.bat
SimpleExample.fmu
SimpleExample_debug.fmu
vEcuConf.yaml

If you did not change the YAML file, the vECU name is `SimpleExample`. VECU-BUILDER created two vECU versions: a release and a debug version.

When you extract the FMUs, the folders of the 2 versions can be compared. The difference is that the debug FMU contains a PDB file in the resources folder and the DLL is debuggable.

The release FMU contains a VARVAL in addition to the debug FMU. The behavior of both is the same.

4.2 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 as well. VECU-BUILDER patches in the values for TCP port and IP address, which were specified in the YAML file. For instance:

Original A2L file	Patched A2L file

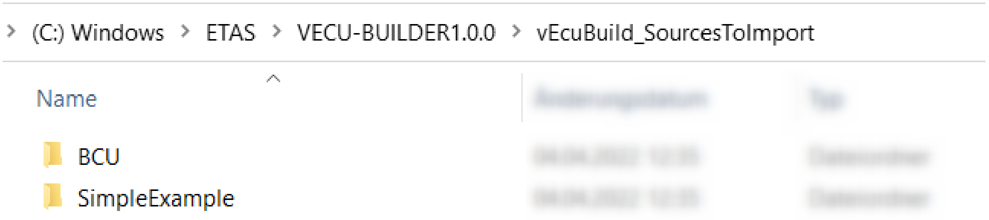
/begin XCP_ON_TCP_IP
    0x0100   /* XCP on IP 1.0 */
    <TCPPORT> /* Port */
    ADDRESS "**<IPADDR>**"
  /end XCP_ON_TCP_IP
	/begin XCP_ON_TCP_IP
	    0x0100 /* XCP on IP 1.0 */
	    12345 /* Port */
	    ADDRESS "**127.0.0.1**"
	/end XCP_ON_TCP_IP

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

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

4.3 Get familiar with VECU-BUILDER

For getting started with VECU-BUILDER and test the iterative approach we provide 2 examples.



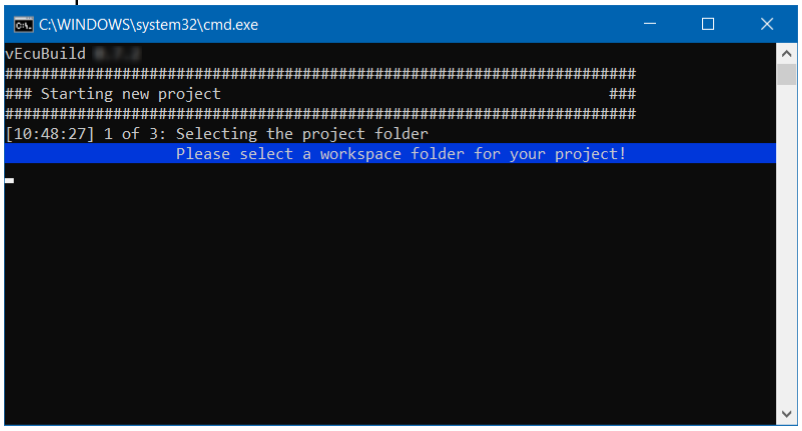
> (C:) Windows > ETAS > VECU-BUILDER1.0.0 > vEcuBuild_SourcesToImport

Name	[redacted]	[redacted]
BCU	[redacted]	[redacted]
SimpleExample	[redacted]	[redacted]

4.3.1 Simple Example

1. Go to your workspace (e.g. C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces).
2. Create a folder “SimpleExample”.
3. Go to the location where you installed VECU-BUILDER. When you used the default location, it can be found in C:\ETAS\VECU-BUILDER
4. Execute `StartNewProject.cmd` with double-click.

⇨ A console window is opened. You will be asked where your workspace should be saved.



C:\WINDOWS\system32\cmd.exe

```
vEcuBuild [redacted]
########################################################################
### Starting new project                                             ###
########################################################################
[10:48:27] 1 of 3: Selecting the project folder
                   Please select a workspace folder for your project!
```

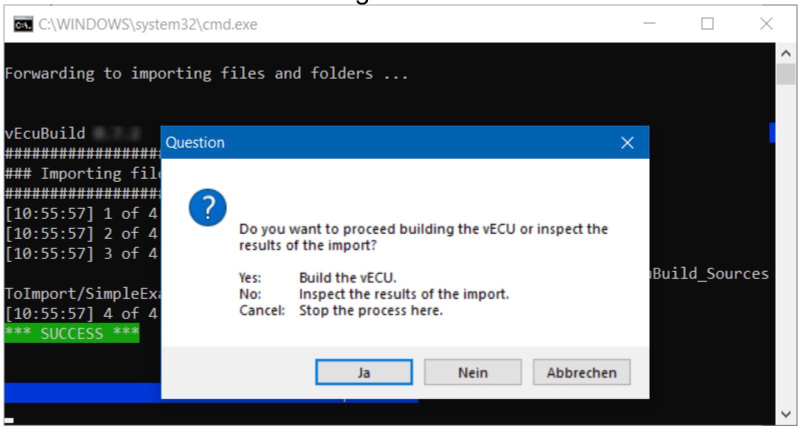
5. Select the folder you created in Step 2.

⇨ The configuration file vEcuConf.yaml is opened in Notepad++.

6. Close the Notepad ++ application.

⇨ The windows console is shown again and displaying the progress. A how to proceed question inquiry is shown.

7. Select **Yes** to continue building the vECU.



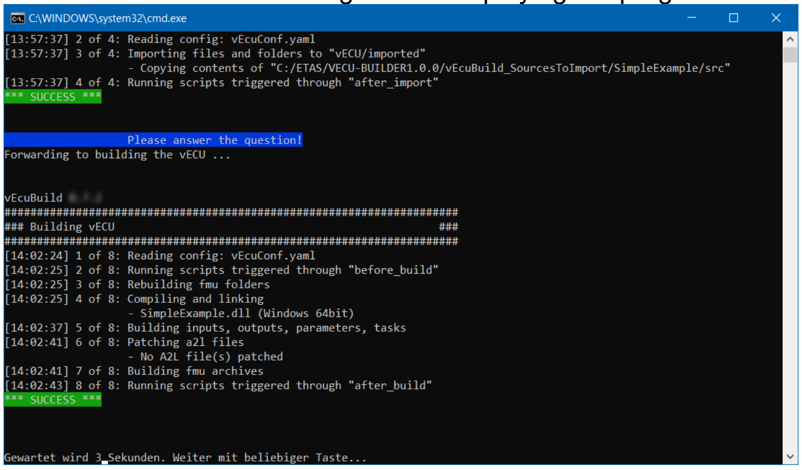
```
C:\WINDOWS\system32\cmd.exe

Forwarding to importing files and folders ...

vEcuBuild [redacted]
#################
### Importing fil
#################
[10:55:57] 1 of 4
[10:55:57] 2 of 4
[10:55:57] 3 of 4                                          Build_Sources

ToImport/SimpleEx
[10:55:57] 4 of 4
*** SUCCESS ***
```

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.

⇨ The vECU has been build in 2 versions (release and debug version) in your workspace with the SimpleExample as name. The windows console is shown again and displaying the progress.

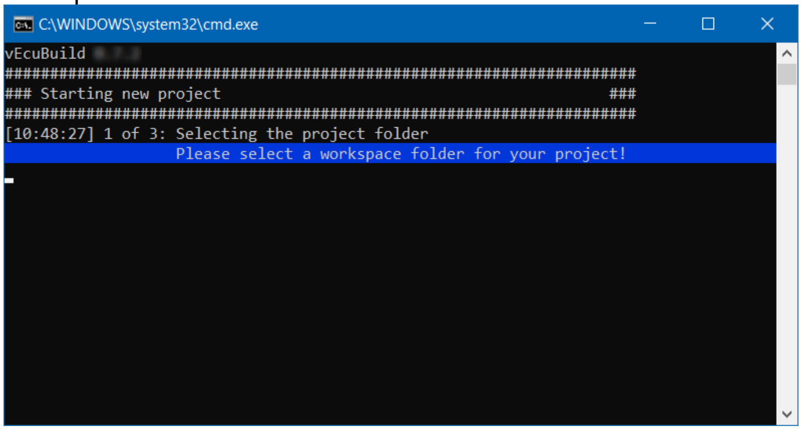


```
C:\WINDOWS\system32\cmd.exe
[13:57:37] 2 of 4: Reading config: vEcuConf.yaml
[13:57:37] 3 of 4: Importing files and folders to "vECU/imported"
                   - Copying contents of "C:/ETAS/VECU-BUILDER1.0.0/vEcuBuild_SourcesToImport/SimpleExample/src"
[13:57:37] 4 of 4: Running scripts triggered through "after_import"
*** SUCCESS ***


                   Please answer the question!
Forwarding to building the vECU ...


vEcuBuild [redacted]
######################################################################
### Building vECU                                                  ###
######################################################################
[14:02:24] 1 of 8: Reading config: vEcuConf.yaml
[14:02:25] 2 of 8: Running scripts triggered through "before_build"
[14:02:25] 3 of 8: Rebuilding fmu folders
[14:02:25] 4 of 8: Compiling and linking
                   - SimpleExample.dll (Windows 64bit)
[14:02:37] 5 of 8: Building inputs, outputs, parameters, tasks
[14:02:41] 6 of 8: Patching a2l files
                   - No A2L file(s) patched
[14:02:41] 7 of 8: Building fmu archives
[14:02:43] 8 of 8: Running scripts triggered through "after_build"
*** SUCCESS ***



Gewartet wird 3 Sekunden. Weiter mit beliebiger Taste...
```

4.3.2 BCU (Body Control Unit)

1. Go to your workspace (e.g. C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces).
2. Create a folder “BCU”.
3. Go to the location where you installed VECU-BUILDER. When you used the default location, it can be found in C:\ETAS\VECU-BUILDER
4. Open folder `vEcuBuild_v1.0.0`.
5. Execute `StartNewProject.cmd` with double-click.

⇨ A console window is opened. You will be asked where your workspace should be saved.



C:\WINDOWS\system32\cmd.exe

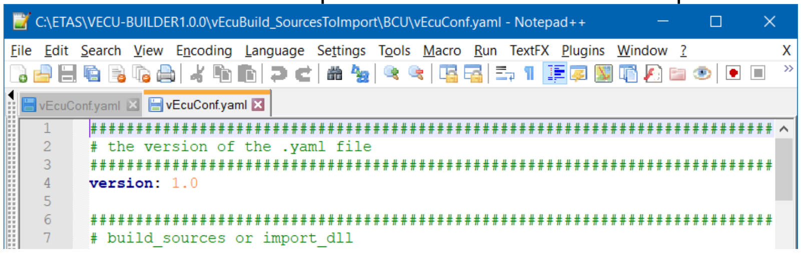
```
vEcuBuild [redacted]
#######################################################################
### Starting new project                                           ###
#######################################################################
[10:48:27] 1 of 3: Selecting the project folder
                   Please select a workspace folder for your project!
```

6. Select BCU folder in C:\ETAS\VECU-BUILDER1.0.0\vEcuBuild_SourcesToImport\BCU.

⇨ The Step “Starting new project” is completed. The configuration file vEcuConf.yaml, which was just created by VECU-BUILDER, is opened in Notepad++.

7. Go to C:\ETAS\VECU-BUILDER1.0.0\vEcuBuild_SourcesToImport\BCU and open preconfigured YAML file (vEcuConf.yaml) of the example folder to copy the content.

⇨ The document should be opened in a second tab of Notepad++.

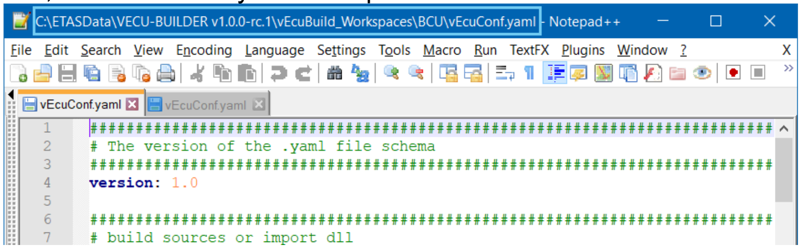


C:\ETAS\VECU-BUILDER1.0.0\vEcuBuild_SourcesToImport\BCU\vEcuConf.yaml - Notepad++

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

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

##############################################################################
# build_sources or import_dll
```

8. Select the whole content with STRG/CTRL+A and copy it with STRG/CTRL+C.
9. Switch to the current YAML file. This should be the first tab of Notepad ++. If you are unsure, check the storage location in the title, it should show your workspace location.



C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces\BCU\vEcuConf.yaml - Notepad++

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

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

##############################################################################
# build sources or import dll
```

10. Select the whole content with STRG/CTRL+A and paste the content into the YAML file with STRG/CTRL+V of your workspace.
11. Navigate to line 82: `build_tool`.
12. Change `MinGW Makefiles` into `Visual Studio 16 2019`.

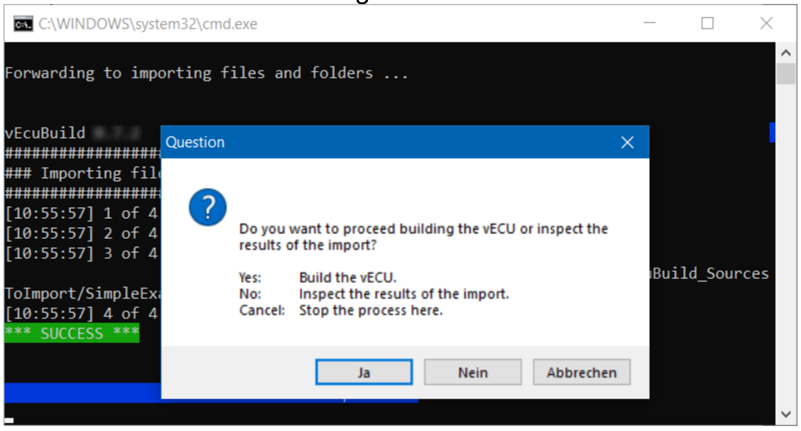
⇨ For the moment everything is configured and the Step “Importing files and folders” is completed.

In the background some files have been added to your workspace, e.g. imported files in the *vECU folder* > *imported* like A2L files or stubs as C files and other folders. Everything was defined in the YAML file. See `import_into_project` (line 28) in the YAML file.

13. Close the Notepad ++ application.

⇨ The windows console is shown again and displaying the progress. A how to proceed question inquiry is shown.

14. Select **Yes** to continue building the vECU.

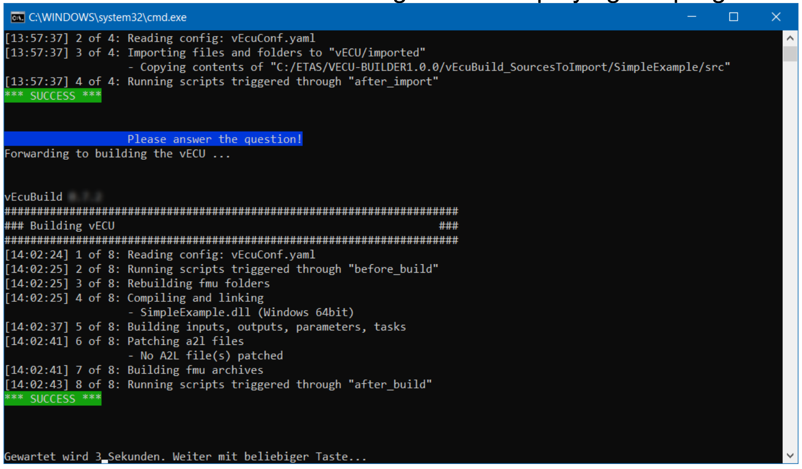


```
Forwarding to importing files and folders ...

vEcuBuild [redacted]
#################
### Importing fil
#################
[10:55:57] 1 of 4
[10:55:57] 2 of 4
[10:55:57] 3 of 4                                          Build_Sources

ToImport/SimpleEx
[10:55:57] 4 of 4
*** SUCCESS ***
```

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

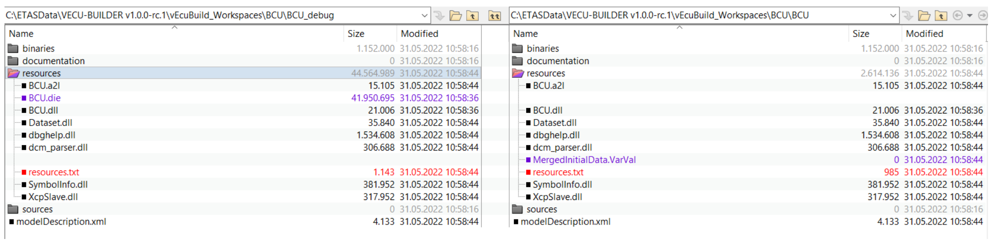
Yes: Build the vECU.
No: Inspect the results of the import.
Cancel: Stop the process here.

⇨ The Step “Building vECU” completed. In the background the vECU has been added in 2 versions (release and debug version) to your workspace with the name you specified in the YAML file. As we did not touch this, the name should be `BCU.fmu`. See `import_into_project` (line 28) in the YAML file.
Also additional BAT files were added: StartDebugger.bat and ShowSymbolDetails.bat
The windows console is shown again and displaying the progress.



```
[13:57:37] 2 of 4: Reading config: vEcuConf.yaml
[13:57:37] 3 of 4: Importing files and folders to "vECU/imported"
                   - Copying contents of "C:/ETAS/VECU-BUILDER1.0.0/vEcuBuild_SourcesToImport/SimpleExample/src"
[13:57:37] 4 of 4: Running scripts triggered through "after_import"
*** SUCCESS ***


                   Please answer the question!
Forwarding to building the vECU ...


vEcuBuild [redacted]
######################################################################
### Building vECU                                                  ###
######################################################################
[14:02:24] 1 of 8: Reading config: vEcuConf.yaml
[14:02:25] 2 of 8: Running scripts triggered through "before_build"
[14:02:25] 3 of 8: Rebuilding fmu folders
[14:02:25] 4 of 8: Compiling and linking
                   - SimpleExample.dll (Windows 64bit)
[14:02:37] 5 of 8: Building inputs, outputs, parameters, tasks
[14:02:41] 6 of 8: Patching a2l files
                   - No A2L file(s) patched
[14:02:41] 7 of 8: Building fmu archives
[14:02:43] 8 of 8: Running scripts triggered through "after_build"
*** SUCCESS ***



Gewartet wird 3 Sekunden. Weiter mit beliebiger Taste...
```

To inspect the differences of the two FMU versions (release and debug) extract the FMU and compare them with an appropriate tool. The mainly difference can be found in the resources folder:



C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces\BCU\BCU_debug

Name	Size	Modified
binaries	1.152.000	31.05.2022 10:58:16
documentation	0	31.05.2022 10:58:16
resources	44.564.989	31.05.2022 10:58:44
BCU.a2l	15.105	31.05.2022 10:58:44
BCU.die	41.950.695	31.05.2022 10:58:36
BCU.dll	21.006	31.05.2022 10:58:36
Dataset.dll	35.840	31.05.2022 10:58:44
dbghelp.dll	1.534.608	31.05.2022 10:58:44
dcm_parser.dll	306.688	31.05.2022 10:58:44
resources.txt	1.143	31.05.2022 10:58:44
SymbolInfo.dll	381.952	31.05.2022 10:58:44
XcpSlave.dll	317.952	31.05.2022 10:58:44
sources	0	31.05.2022 10:58:16
modelDescription.xml	4.133	31.05.2022 10:58:44

C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces\BCU\BCU

Name	Size	Modified
binaries	1.152.000	31.05.2022 10:58:16
documentation	0	31.05.2022 10:58:16
resources	2.614.136	31.05.2022 10:58:44
BCU.a2l	15.105	31.05.2022 10:58:44
BCU.dll	21.006	31.05.2022 10:58:36
Dataset.dll	35.840	31.05.2022 10:58:44
dbghelp.dll	1.534.608	31.05.2022 10:58:44
dcm_parser.dll	306.688	31.05.2022 10:58:44
MergedInitialData.VarVal	0	31.05.2022 10:58:44
resources.txt	985	31.05.2022 10:58:44
SymbolInfo.dll	381.952	31.05.2022 10:58:44
XcpSlave.dll	317.952	31.05.2022 10:58:44
sources	0	31.05.2022 10:58:16
modelDescription.xml	4.133	31.05.2022 10:58:44

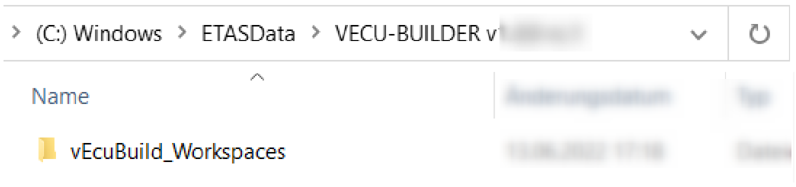
The debug version is debuggable and uses variable names or task names. In the release version this is replaced with addresses. The behavior of both FMU is the same. The release FMU will be derived of the debug FMU without using a compiler or build tool.

In summary, the idea behind the release and debug FMU versions is to have a FMU which can be used as a vECU and does not contain all information on the one hand (release version) and to have a FMU containing all information which is debuggable (debug version) on the other hand.

5 Working with VECU-BUILDER

The *Working with VECU-BUILDER* section provides information with focus on step-by-step instructions as how-tos.

As workspace location we recommend using the following path:
C:\ETASData\VECU-BUILDER v1.0.0-rc.1\vEcuBuild_Workspaces



> (C:) Windows > ETASData > VECU-BUILDER v

Name

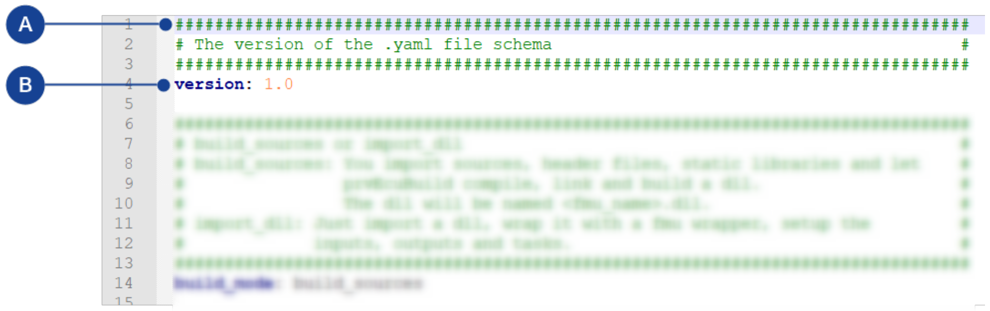
vEcuBuild_Workspaces

This folder is created during the installation process.

5.1 YAML File Reference

The YAML file contains the configuration for the vECU and is divided in different sections. You are guided through the YAML file with providing comments for each section and configuration attributes.

Every section is structured in a standardized way:



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

A: comment with information to the appropriate section

B: configuration attribute and value

The following attributes are provided within the YAML file. You can also enter own comments in the YAML file.

NOTE

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

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

- **`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, header files, static libraries (either as AUTOSAR Classic compliant or legacy C-code) and let prvEcuBuild compile, link and build a DLL file. The DLL file will be named <fmu_name>.dll

`import_dll`: You import an existing, already compiled and linked software in the form of a DLL, wrap it with a FMU wrapper, setup the inputs, outputs and tasks. That DLL already contains the code of your vECU. See also line 43.

- **`fmu_name`**

  Enter a name for the vECU depicted as a standalone FMU. Technically it is a ZIP archive. The code of your vECU will be located inside the FMU in the folder "resources/<fmu_name>.dll". This DLL file will be loaded and run by the FMU.

  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.

- **`import_into_project`**

  Enter the path to files which should be imported. Here you specify paths to the individual files, such as .c, .h, .cpp, .hpp or .zip archives that will be extracted upon importing or folders with your code. The import target is the folder "vEcu/imported". You can import files, folders and ZIP archives. Environment variables can be used like this: ${SomeEnvironmentVariable} → e.g. –
  `'${VECUBUILD_HOME}\..\vEcuBuild_SourcesToImport\SimpleExample\src'`

  Values must be set in apostrophes

- **`import_external_vecu_dll`**

  Only needed if you use an existing DLL and you selected `import_dll` for `build_mode` (line 14).

  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. There must exist a corresponding PDB file with the same file name.

  `get_updates_from`: If vEcuBuild can find a DLL and PDB in this folder, it will update the imported DLL. Environment variables can be used like this: ${SomeEnvironmentVariable} → e.g. `'${SystemDrive}\Sandbox'`

  Values must be set in apostrophes

- **`architecture`**

  Specify the architecture. When importing sources, the setting of this attribute is 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 is to be set to the same. In the default YAML file the 32bit is commented out.

- **`xcp_slave`**

  Enter the port and IP address for XCP slave. These values are transferred to the A2L file. The used protocol is TCP. See also NEW A2L File Preparation.

  `port`: default is 1802

`ip_address`: default is 127.0.0.1

- **`operating_system`**

  Enter the operating system. Currently Windows is supported.
- **`build_tool`**

  Enter the build tool as the preferred development environment. Currently you can choose between several Visual Studio versions and MinGW Makefiles. In one case Visual Studio project files are generated, in the other case a make file for use with mingw32-make. In the default YAML file the Visual Studio 16 2019 is commented out.

  NOTE

  If Visual Studio is used as build tool, the vECU.dll can be investigated with a debugger. If MinGW is used as build tool, the vECU.dll cannot be investigated with a debugger.
- **`inputs`**, **`outputs`**, **`parameters`**, **`locals`**

  Enter the variables. Inputs, outputs, parameters, and locals refer to the causality of the FMI. Each wildcard must match a global variable. Wildcards * and ? are allowed. Arrays can be added using myArray*, same for structures. If your wildcard expression breaks the YAML compatibility, set it in apostrophs.

  Example: '*a' finds all symbols ending with an 'a'. In the default YAML file some outputs are commented out.
- **`initial_data`**

  Enter the path for source and target 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.

  Values must be set in apostrophes
- **`eeprom`**

  Specify the eeprom (Electrically Erasable Programmable Read-Only Memory). 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 behaviour where EEPROM stored data are preserved and not lost once the simulation of vECU terminates. A typical application of such feature is the storage of total mileage information in ESP controller.

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

  `sync`: This can 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)

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

  c_variables: The C variable names that store the eeprom data. (optional). Skipping c_variables means, the vECU handles loading and saving the eeprom. This tool will just provide the files then.

  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 with the following specifications. To simulate the microcontroller environment with its periodically executed tasks in your software, these are to be defined as tasks of your vECU.

  `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

  See also hints as comments in line 164, 165 and 166.

- **`redirect_function_calls`**

  Enter the names for replacing functions. Consider the signatures of the replaced and the substitute function must be the same. This allows you to test the behaviour of your software using alternative implantation without changing the code and to replace unfinished or hardware-dependent functions with mock functions.

  `replaced_function`: Enter the function name of the function that should 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` for `build_mode` (line 14), contrary to line 43. Here you can select files and/or folders that should be included or excluded in/from the vECU build process. Files are included into the build if and if only they are matched by at least one build_include_filter and not matched by any build_exclude_filter.

  Values must be set in apostrophes

- **`assembly_list_files`**

  Refers to line 188&193. From the given sources defined by "build_include_filters" and "build_exclude_filters" only pass the ones to the compiler that are listed in a file. If no assembly list files are configured, all sources will be compiled. See more details in the comment in line 204.

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

  Only usable if you selected `build_sources` for `build_mode` (line 14), contrary to line 43. `additional_defines` are passed to the preprocessor. This is useful if you need to set/unset some defines to adapt to the new windows target. Brackets '(', ')' must be escaped as '\(', '\)'. In the default YAML file the examples are commented out.

- **`additional_static_libraries`**

  Only usable if you selected `build_sources` for `build_mode` (line 14), contrary to line 43. The libraries need to be located in the folder "./projects/vEcu/imported". Environment variables can be used like this: ${SomeEnvironmentVariable}

  In the default YAML file the examples are commented out.

→ Configuring the attributes `build_include_filters`, `build_exclude_filters`, `assembly_list_files`, `additional_include_directories`, `additional_include_directories` enables you to remove the hardware dependencies of your SW so that your vECU can be executed in a virtual environment based on x86 hardware.

- `environment_variables`

  You can define environment variables that will be set by the build process or the FMI wrapper of the vECU. Example: PATH=c:\Temp;${PATH}

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

  In the default YAML file the examples are commented out.

- `calls` & `additional_scripts`

  You can define calls or additional scripts. Project-specific scripts can be configured to be executed at various phases of the import and build process. Like copy or modify files, add files to the FMU archive, parse files. Enter `filename`, `arguments` (only for calls), `command_line`, `trigger` and `priority`. Environment variables can be used.

  In the default YAML file the examples are commented out.

  Additional scripts refer to the folder C:\ETAS\VECU-BUILDER1.0.0\vEcuBuild_v1.0.0-rc.1\vEcuBuild_v1.0.0-rc.1\build\project_template\build\additional_scripts

- `patch_a2l_file`

  Enter the name of the A2L file. An A2L File is required to connect an MCD tool like e.g. INCA to the running vECU. The A2L file needs to be located in the folder: "./projects/vEcu/imported". The vEcuBuild will update the memory addresses of all measurements and characteristics (not only the inputs and outputs) in the A2L file. Environment variables can be used like this: ${SomeEnvironmentVariable}

  Only one A2L file can be defined. In the default YAML file the example is commented out.

5.2 Command Line Interface

VECU-BUILDER is controlled via a command-line interface. Using the CMD files provided with the installation you will be guided within a command line interface and through dialog windows.

The sequence of the building process is always the same when you use “StartNewProject.cmd”. It contains 3 main steps. The current status is always shown in the console window.

STARTING NEW PROJECT

1. Selecting the project folder
2. Creating project template
3. Update vEcuConf.yaml

IMPORTING FILES and FOLDERS

1. Running scripts triggered through “before_import”
2. Reading config: vEcuConf.yaml
3. Importing files and foldes to “vECU/imported”

4. Running scripts triggered through “after_import”

BUILDING VECU

1. Reading config: vEcuConf.yaml
2. Running scripts triggered through “before_import”
3. Rebuilding FMU folders
4. Compiling and Linking (BCU.dll)
5. Building inputs, outputs, parameters, taks
6. Patching A2L files
7. Building FMU archives
8. Running scripts triggered through “after_import”

5.3 Create a vECU

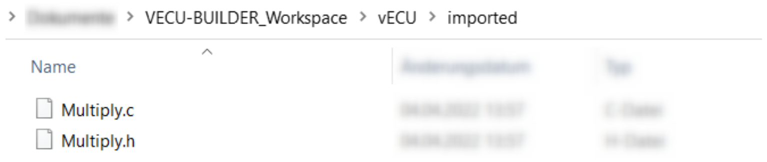
There are two concepts to create a vECU:

- Create vECU from scratch
- Edit existing vECU

The first concept is to Create vECU from scratch using the `StartNewProject.cmd`. Starting this batch file you will be guided through the whole process including import and build step. Needed files will be imported and the vECU will be build.

The second concept is to edit an existing YAML file and changing the configuration. You can also import files and folder manually using the `1_Import.cmd`. When you are finished you can use `2_Build.cmd` to create the vECU based in the import and the configuration.

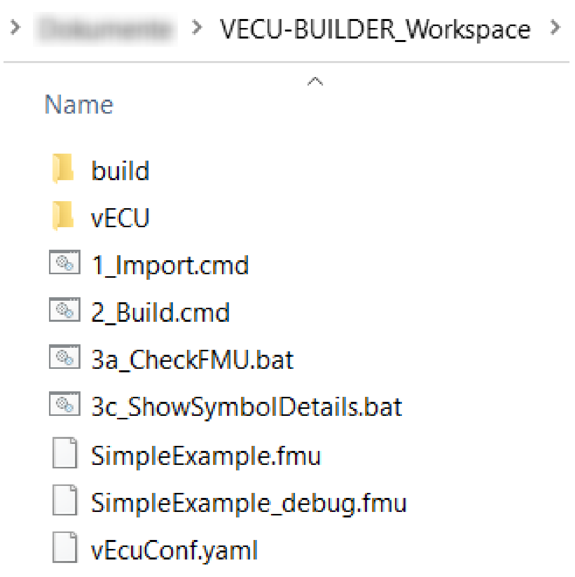
In both cases the resulting vECU is self-contained. That means dependencies are not present and the vECU can be used independently. The needed files for that are copied into the workspace and can be found in `imported` folder (see vECU\imported).



Multiply.c
Multiply.h

Figure 5: Imported folder with copied dependencies of the vECU

Every vECU gets a separate workspace containing the same structure.



> [redacted] > VECU-BUILDER_Workspace >

Name

- build
- vECU
- 1_Import.cmd
- 2_Build.cmd
- 3a_CheckFMU.bat
- 3c_ShowSymbolDetails.bat
- SimpleExample.fmu
- SimpleExample_debug.fmu
- vEcuConf.yaml

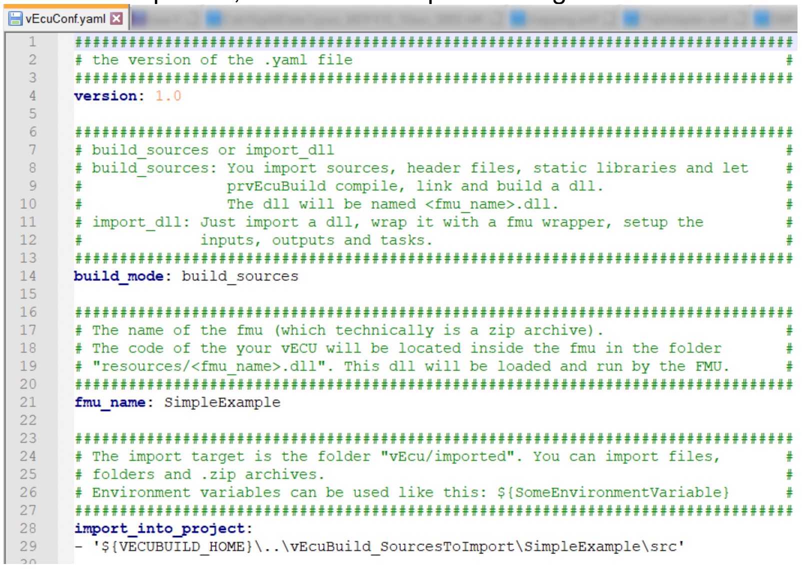
Figure 6: Standardized structure of vECU workspace

The created vECUs with VECU-BUILDER are self-contained. Possible dependencies are copied into the workspace, see `imported` folder in vECU\imported.

5.3.1 Configuring the YAML file

1. Open the vEcuConf.yaml of your workspace.
2. Adapt the configuration to your needs.

⇨ Every section contains a comment with a description which content is expected, and selection options are given.



vEcuConf.yaml

```
 1  ################################################################################
 2  # the version of the .yaml file                                                #
 3  ################################################################################
 4  version: 1.0
 5
 6  ################################################################################
 7  # build_sources or import_dll                                                  #
 8  # build_sources: You import sources, header files, static libraries and let    #
 9  #                prvEcuBuild compile, link and build a dll.                    #
10  #                The dll will be named <fmu_name>.dll.                         #
11  # import_dll: Just import a dll, wrap it with a fmu wrapper, setup the         #
12  #             inputs, outputs and tasks.                                       #
13  ################################################################################
14  build_mode: build_sources
15
16  ################################################################################
17  # The name of the fmu (which technically is a zip archive).                    #
18  # The code of the your vECU will be located inside the fmu in the folder       #
19  # "resources/<fmu_name>.dll". This dll will be loaded and run by the FMU.      #
20  ################################################################################
21  fmu_name: SimpleExample
22
23  ################################################################################
24  # The import target is the folder "vEcu/imported". You can import files,       #
25  # folders and .zip archives.                                                   #
26  # Environment variables can be used like this: ${SomeEnvironmentVariable}      #
27  ################################################################################
28  import_into_project:
29  - '${VECUBUILD_HOME}\..\vEcuBuild_SourcesToImport\SimpleExample\src'
```

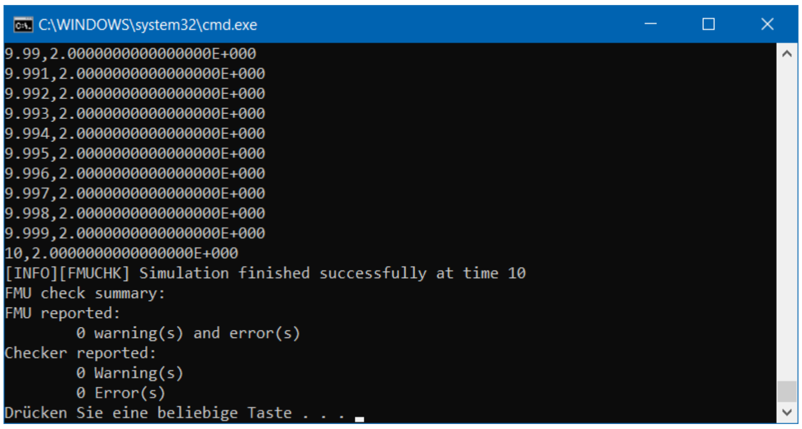
In the comment above line 14 you can see that there are 2 options to choose from for the `build_mode`: `build_sources` or `import_dll`.

5.4 Check created vECU (release version)

You can check if the created FMU can be executed.

1. Go to your workspace folder.
   When you used the default location, it can be found in
   C:\Users\Username\Documents\
2. Execute `3a_CheckFMU.bat` with double-click.

⇨ A console window is opened. The FMU is checked, and the result is shown in the console window.



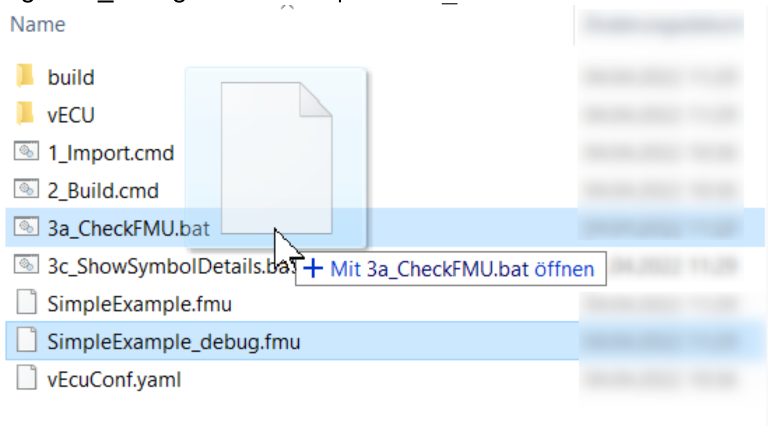
```
9.99,2.0000000000000000E+000
9.991,2.0000000000000000E+000
9.992,2.0000000000000000E+000
9.993,2.0000000000000000E+000
9.994,2.0000000000000000E+000
9.995,2.0000000000000000E+000
9.996,2.0000000000000000E+000
9.997,2.0000000000000000E+000
9.998,2.0000000000000000E+000
9.999,2.0000000000000000E+000
10,2.0000000000000000E+000
[INFO][FMUCHK] Simulation finished successfully at time 10
FMU check summary:
FMU reported:
        0 warning(s) and error(s)
Checker reported:
        0 Warning(s)
        0 Error(s)
Drücken Sie eine beliebige Taste . . .
```

The first number represents the simulation time, the second number is the output.

5.5 Check created vECU (debug version)

You can check if the created FMU can be executed.

1. Go to your workspace folder.
2. Drag the *_debug.fmu and drop it to `3a_CheckFMU.bat`.



⇨ A console window is opened. The FMU is checked, and the result is shown in the console window.

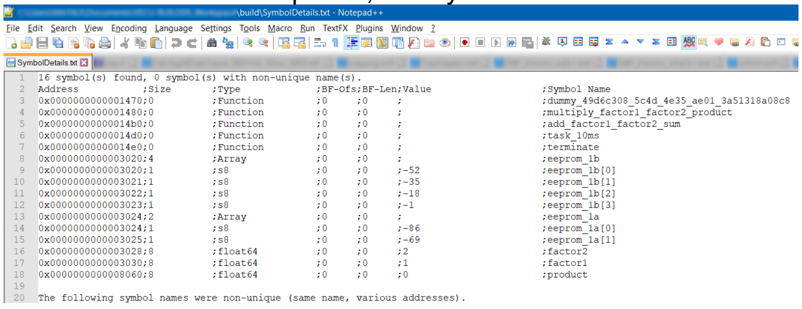
5.6 Show Symbol information

To check the compile, you can have a look into the symbol details.

1. Go to your workspace folder.
   When you used the default location, it can be found in
   C:\Users\Username\Documents\

2. Execute `3c_ShowSymbolDetails.bat` with double-click.

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


[redacted]\build\SymbolDetails.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run TextFX Plugins Window ?
SymbolDetails.txt

```
16 symbol(s) found, 0 symbol(s) with non-unique name(s).
Address            ;Size       ;Type            ;BF-Ofs;BF-Len;Value                     ;Symbol Name
0x0000000000001470;0          ;Function        ;0     ;0     ;                          ;dummy_49d6c308_5c4d_4e35_ae01_3a51318a08c8
0x0000000000001480;0          ;Function        ;0     ;0     ;                          ;multiply_factor1_factor2_product
0x00000000000014b0;0          ;Function        ;0     ;0     ;                          ;add_factor1_factor2_sum
0x00000000000014d0;0          ;Function        ;0     ;0     ;                          ;task_10ms
0x00000000000014e0;0          ;Function        ;0     ;0     ;                          ;terminate
0x0000000000003020;4          ;Array           ;0     ;0     ;                          ;eeprom_1b
0x0000000000003020;1          ;s8              ;0     ;0     ;-52                       ;eeprom_1b[0]
0x0000000000003021;1          ;s8              ;0     ;0     ;-35                       ;eeprom_1b[1]
0x0000000000003022;1          ;s8              ;0     ;0     ;-18                       ;eeprom_1b[2]
0x0000000000003023;1          ;s8              ;0     ;0     ;-1                        ;eeprom_1b[3]
0x0000000000003024;2          ;Array           ;0     ;0     ;                          ;eeprom_1a
0x0000000000003024;1          ;s8              ;0     ;0     ;-86                       ;eeprom_1a[0]
0x0000000000003025;1          ;s8              ;0     ;0     ;-69                       ;eeprom_1a[1]
0x0000000000003028;8          ;float64         ;0     ;0     ;2                         ;factor2
0x0000000000003030;8          ;float64         ;0     ;0     ;1                         ;factor1
0x0000000000008060;8          ;float64         ;0     ;0     ;0                         ;product

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

6 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