VEOS

Manual

For VEOS 5.2

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2010 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Manual 13

Introduction and Overview 15

New Features of VEOS ... 16
- New Features of VEOS 5.2 ... 16

Introduction to Offline Simulation with VEOS ... 18
- Basics on VEOS ... 18
- Basics on Simulation Systems ... 20
- Installation and Virtualization of VEOS ... 21
- Simulation Architecture of VEOS ... 23
- Timing in Offline Simulation with VEOS ... 24
- Simulating Multicore V-ECUs ... 27
- VEOS Modules ... 28

Introduction to VEOS Player ... 30
- Basics on the VEOS Player ... 30
- How to Open an OSA File ... 33
- User Interface of the VEOS Player ... 35

Demos for VEOS ... 37
- Demos for VEOS ... 37

Compatibility, Migration and Discontinuations ... 40

Compatibility of VEOS ... 40
- Compatibility of VEOS 5.2 ... 40

Migrating from Prior Versions of VEOS ... 43
- Migrating from VEOS 3.4 to 3.5 ... 45
- Migrating from VEOS 3.5 to 3.6 ... 45
- Migrating from VEOS 3.6 to 3.7 ... 46
- Migrating from VEOS 3.7 to 4.0 ... 48
- Migrating from VEOS 4.0 to 4.1 ... 48
- Migrating from VEOS 4.1 to 4.2 ... 49
- Migrating from VEOS 4.2 to 4.3 ... 51
- Migrating from VEOS 4.3 to 4.4 ... 52
- Migrating from VEOS 4.4 to 4.5 ... 52
- Migrating from VEOS 4.5 to 5.0 ... 53
- Migrating from VEOS 5.0 to 5.1 ... 53

Migrating from VEOS 5.1 to 5.2 ... 54
Migrating ASM Models ... 55

Discontinuations in VEOS ... 55
Discontinuations as of VEOS 3.5 ... 56
Discontinuations as of VEOS 4.0 ... 57
Discontinuations as of VEOS 4.1 ... 57
Discontinuations as of VEOS 4.2 ... 57
Discontinuations as of VEOS 4.3 ... 57
Discontinuations as of VEOS 4.4 ... 58
Discontinuations as of VEOS 4.5 ... 58
Discontinuations in VEOS as of dSPACE Release 2021-B ... 59

Integrating the Simulation System 61

Basics and Instructions ... 62

Introduction to Simulation System Integration ... 62
Basics on Integrating the Simulation System ... 62

Importing V-ECU Implementations ... 65
Basics on Importing V-ECU Implementations ... 65
How to Import V-ECU Implementations ... 67

Importing Simulink Implementation Containers (SICs) ... 70
Basics on Importing Simulink Implementation Containers (SICs) ... 70
How to Import Simulink Implementations ... 71
Elements Generated for Simulink Implementation Containers ... 73

Importing Bus Simulation Containers (BSCs) ... 76
Basics on Importing Bus Simulation Containers (BSCs) ... 77
How to Import Bus Simulation Containers ... 78
Elements Generated for Bus Simulation Containers ... 80

Importing Functional Mock-Up Units (FMUs) ... 83
Definition of the FMI Standard and FMUs ... 84
Basics on Importing Functional Mock-Up Units (FMUs) ... 85
Specifying the FMU Simulation Behavior ... 87
How to Import FMUs ... 90
Variable Description (TRC) File Entries for FMU Variables ... 92

Importing System Model Containers (SMCs) ... 95
How to Import System Model Containers ... 95

Importing VPUs ... 97
How to Import VPUs ... 97

Connecting VPUs ... 100
Basics on Signal Connection Dependencies ... 100
Ways to Connect VPU Signals ... 102
Examples for Connecting VPUs by Dragging Structural Elements ... 104
Specifying Blocking or Non-Blocking Communication ... 109
Connecting Communication Controllers to Communication Clusters ... 111

Exporting System Model Containers ... 114
How to Export System Model Containers ... 114

Reference Information ... 116

Panes ... 116
Model Communication Pane ... 116
Network Topology Pane ... 118
Port Topology Pane ... 120

Elements and Properties ... 122
Bus Type ... 122
Communication Cluster ... 123
Communication Controller ... 124
Communication Package ... 126
Controller-to-Cluster Connection ... 127
Port ... 127
Port Group ... 128
Signal ... 129
Signal Group ... 131
VPU (V-ECU / Environment VPU / Controller VPU / Bus VPU) ... 131

Commands and Dialogs ... 134
Add New Communication Cluster ... 136
Add New Model Communication Package ... 138
Autoconnect ... 138
Clear Recently Exported Files ... 140
Clear Recently Imported Files ... 140
Clear Recently Used Files ... 141
Close ... 142
Collapse All Nodes ... 142
Delete ... 143
Disconnect all Child Connections ... 143
Expand All Nodes ... 144
Expanding and Hiding Elements ... 145
Export ... 145
Export the Variable Description Files (A2L or TRC) when the OSA File is Saved ... 146

Filter Editor ... 147
Filtering Elements ... 148
Hide Connected Communication Clusters ... 149
Hide Connected Communication Controllers ... 150
Hide Connected Incoming Ports ... 150
Hide Connected Outgoing Ports ... 151
Import ... 152
Info ... 158
New ... 159
Open ... 159
Recently Used ... 160
Replace ... 161
Save ... 161
Save As ... 162
Simulation Target Manager ... 163
Update ... 166

Compiler Options ... 166
Compiler Options ... 166

Automation ... 169
Automating the Opening and Saving of an Offline Simulation Application ... 169
Automating the Import and Removal of VPUs ... 170
Automating the Connection of VPUs ... 172

Troubleshooting ... 176
Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target ... 176
Problem When Building a Model Implementation with a 64-Bit Simulation Target ... 177

Limitations ... 178
Limitations for Integrating the Simulation System ... 178

Loading and Running an Offline Simulation Application, and Debugging Source Code 183

Basics and Instructions ... 184

Loading and Running an Offline Simulation Application ... 184
Simulation on a Local or Remote Simulator ... 185
How to Select a Remote Simulator ... 186
How to Load and Run an Offline Simulation Application ... 188
Specific Variables for Simulation ... 189

Bus Simulation on VEOS 196
Bus Communication Features for SIL Testing 196
Restbus Simulation in VEOS 199
Workflow for Simulating Bus Communication 199

Debugging Source Code During Offline Simulation 201
Basics on Debugging Source Code in an Offline Simulation 202
How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation 205
How to Debug Source Code Built with the GCC Compiler in an Offline Simulation 209

Reference Information 212
Break Time 213
Clear Simulator List 213
Generate Bus Log Files 214
Generate Simulation Log Files 215
Generate XCP Log Files 215
Load 216
Pause 218
Real-Time Acceleration Factor 218
Select Simulator 219
Single Step Time 220
Start 221
Step 222
Stop 222
Stop Time 223
Unload 223
Wait for Debugger on Load 224

Automation 226
Automating the Simulation Configuration 226
Automating the Download and Control of an Offline Simulation Application 227
Automating VEOS Player Event Handling with Python 228

Limitations 232
Limitations for Loading and Running an Offline Simulation Application, and Debugging Source Code 232

Working with Adaptive V-ECUs in VEOS 235

Adaptive V-ECUs in VEOS 236
Basics on Simulating Adaptive V-ECUs 236
Hypervisor Requirements for Adaptive V-ECU Simulation 239

Workflow for Creating and Simulating Adaptive V-ECUs 241
Basics on Adaptive V-ECU Implementation Layers 243
Basics on Debugging Adaptive V-ECUs 244
How to Debug an Adaptive V-ECU 245

Working with the VEOS Adaptive Platform Demos 248
Introduction to the VEOS Adaptive Platform Demos 248
Overview of the Demo Scenarios 249
How to Make Preparations for the Demo Simulations 253
How to Create, Configure, and Export the Adaptive V-ECUs 255
How to Build a Simulation System and Run the Demo Simulations 258

Working with the AUTOSAR Adaptive Platform Demonstrator Samples 266
Basics on the AUTOSAR Adaptive Platform Demonstrator 266
How to Build AUTOSAR Adaptive V-ECUs and Integrate them into an Offline Simulation Application 267

Working with the GPU Passthrough Demo 271
Introduction to the GPU Passthrough Demo 271
Overview of the GPU Passthrough Demo 272
How to Prepare the Host PC for Running the Fluid Simulation 276
How to Run the Fluid Simulation on the Host PC 278
How to Enable GPU Passthrough 281
How to Start a Virtual Machine with GPU Passthrough 283
How to Run the Fluid Simulation with VEOS 285

Working with VEOS on Linux 289

Getting Started with VEOS on Linux 290
Introduction to VEOS on Linux 290
How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS 292
Hypervisor Configuration for Adaptive V-ECU Simulation 296
How to Build an Offline Simulation Application for Linux with the VEOS Player 298

Loading and Running an Offline Simulation Application, and Debugging Source Code 301
How to Load an Offline Simulation Application 301
How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux 306

Running VEOS 5.2 in a Linux Docker Container 308
Basics on Running VEOS 5.2 in a Linux Docker Container 308
How to Prepare a Linux Docker Container for Running VEOS 5.2 310
How to Load and Start a VEOS Simulation in a Linux Docker Container 312

Working with the VEOS Command Line Interface 315

Basics on the VEOS Command Line Interface ... 315
VEOS Build Console Command Reference ... 318
VEOS Simulator Console Command Reference ... 320
VEOS Support Console Command Reference ... 322

Automating the VEOS Player 325

Basics and Instructions ... 326
Basics of the VEOS Player Automation Interface ... 326
Demos for Automating the VEOS Player ... 328
Automating Different VEOS Player Versions ... 329
Programming VEOS Player Automation ... 330

Migrating Python Scripts from Python 3.6 to Python 3.9 ... 332
Main Changes in Python 3.9 ... 332
Main Changes in Handling Python 3.9 with dSPACE Software ... 333
General Information on Using Python Installations ... 334
Technical Changes ... 335

API Reference Information ... 337

Automation Interfaces ... 337
BuildStatusEnum ... 339
BusType ... 340
CommunicationPackageProtocolType ... 340
CommunicationPackageType ... 341
IAdaptiveVEcu ... 341
IAdaptiveVEcus ... 342
IAdaptiveVEcuSettings ... 343
IApplication ... 343
IAutomationBase ... 345
IBuildResult ... 345
IBusChannel ... 346
IBusChannels ... 346
IBusCluster ... 347
IBusClusters ... 348
IBusConnector ... 349
IBusConnectors ... 350
ICommunicationPackage ... 351
ICommunicationPackages ... 351
IConnectionList ... 353

IConnectionResult ... 353
IExportSettings ... 354
IImportSettings ... 354
IMessage ... 356
IMessages ... 357
IPlayerAutomationInterface ... 359
IPlayerAutomationRegistrationInterface ... 359
IProject ... 359
IProjects ... 363
ISimulationConfiguration ... 364
ISimulator ... 365
ISimulatorEvents ... 367
IVersionInformation ... 368
IVpu ... 368
IVpuItem ... 370
IVpuPort ... 372
IVpuPortConnection ... 373
IVpuPortGroup ... 374
IVpuPortGroups ... 375
IVpuPorts ... 376
IVpus ... 377
SerialBaudRate ... 378
SimulationProgress ... 378
VpuItemType ... 379
VpuPortDirection ... 379

Limitations ... 380
Limitations for Automating the VEOS Player ... 380

User Interface Handling 381

Basics and Instructions ... 382
User Interface of the VEOS Player ... 382
Basics on Ribbons ... 384
How to Customize the Screen Arrangement ... 386

Reference Information ... 388
About VEOS Player ... 390
Add to Quick Access Toolbar ... 390
Auto Hide ... 391
Clear Filter ... 391
Close All But This ... 392
Copy Demo Files ... 392

Dock ... 393
dSPACE Help ... 394
dSPACE Log Viewer ... 394
Exit ... 396
Filter Panel ... 397
Float ... 398
Help (Backstage View) ... 399
Hide ... 400
Keyboard Help ... 400
Message Dialog ... 401
Messages Pane ... 401
Minimize the Ribbon ... 403
Move to Next Tab Group ... 404
Move to Previous Tab Group ... 404
New Features and Migration ... 405
New Horizontal Tab Group ... 405
New Vertical Tab Group ... 406
Open dSPACE Log ... 407
PDF Files ... 407
Preferences ... 408
Redo ... 408
Remove from Quick Access Toolbar ... 409
Reset View Set ... 410
Show Quick Access Toolbar Above / Below the Ribbon ... 410
Status Bar ... 411
Switch Panes ... 411
Undo ... 412
Using dSPACE Help ... 413

Automation ... 414
Automating the Start of the VEOS Player, and Getting Version Information ... 414

Glossary 417

Appendix: Message Reader API 431

Introduction to the Message Reader API ... 432
Reading dSPACE Log Messages via the Message Reader API ... 432
Supported dSPACE Products and Components ... 434
Example of Reading Messages with Python ... 434
Example of Reading Messages with C# ... 436

dSPACE.Common.MessageHandler.Logging Reference..... 439
ILogMessage Interface..... 439
ILogSession Interface..... 440
MessageReader Class..... 442
MessageReaderSettings Class..... 443
Severity Enumeration..... 445

Index 447

About This Manual

Content

This manual:

- Introduces you to VEOS and the VEOS Player, its user interface.
- Enables you to integrate a simulation system for offline simulation on VEOS.
- Enables you to load and run offline simulations on VEOS.
- Enables you to automate the VEOS Player.

Required knowledge

This document is intended for engineers who want to run and access an offline simulation.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the icon in dSPACE Help. The PDF opens on the first page.

Introduction and Overview

Where to go from here

Information in this section

New Features of VEOS.....16
Introduction to Offline Simulation with VEOS.....18
Introduction to VEOS Player.....30
Demos for VEOS.....37
Compatibility, Migration and Discontinuations.....40

New Features of VEOS

New Features of VEOS 5.2

Modeling enhancements

Configurable simulation step size for FMUs For FMUs, VEOS now lets you specify the simulation step size for offline simulation.

For more information, refer to Import on page 152.

Updating and replacing VPUs VEOS now lets you update and replace a VPU while keeping existing signal connections, if possible.

For more information, refer to:

- Replace on page 161
- Update on page 166

Support of variable-size signals and binary signals VEOS now supports the following signal types:

- Variable-size signals in BSC and VECU files
- Binary signals in VECU files

For more information, refer to Basics on Signal Connection Dependencies on page 100.

Assigning controllers to channels VEOS now lets you *assign controllers to channels*. LIN frame collisions, i.e., collisions of frames sent via two different cluster nodes, are an application example for assigning controllers to channels in VEOS.

Assigning controllers to channels in VEOS is equivalent to assigning bus access requests to bus accesses in the Bus Manager of ConfigurationDesk.

For more information, refer to Connecting Communication Controllers to Communication Clusters on page 111.

Improvement for connecting VPUs via automation The automation interface of the VEOS Player now supports a new method for connecting VPUs by using connection lists, which is useful for connecting a large number of signals.

For more information, refer to Automating the Connection of VPUs on page 172.

Running VEOS in a Linux Docker container

You can now run VEOS in a Linux Docker container.

For more information, refer to Basics on Running VEOS 5.2 in a Linux Docker Container on page 308.

Adaptive V-ECU support enhancements

GPU support for adaptive V-ECU simulation If you run VEOS on a Linux host PC equipped with a graphical processing unit (GPU), you can now use the GPU for simulating adaptive V-ECUs. VEOS 5.2 also provides a demo that shows you how to pass the GPU through to the virtual machines in which the adaptive V-ECUs are executed.

For more information, refer to Introduction to the GPU Passthrough Demo on page 271.

Support for QEMU images for adaptive V-ECU creation When you create an adaptive V-ECU for VEOS on Linux, you can now also use QEMU images as the implementation layer.

For more information, refer to Basics on Adaptive V-ECU Implementation Layers on page 243.

Command line interface extension: VEOS Support Console

The VEOS command line interface, which was introduced with VEOS 5.1, has been extended in VEOS 5.2.

The new VEOS Support Console now lets you collect support information.

For more information, refer to VEOS Support Console Command Reference on page 322.

Introduction to Offline Simulation with VEOS

Where to go from here

Information in this section

Basics on VEOS 18
VEOS is dSPACE software for the code-based offline simulation of virtual ECUs and environment models on a PC.

Basics on Simulation Systems 20
VEOS lets you handle simulation systems that comprise executable representations of ECUs as well as a representation of their environment and all their interconnections.

Installation and Virtualization of VEOS 21
Information on the installation and virtualization setups supported by VEOS.

Simulation Architecture of VEOS 23
VEOS has a specific simulation architecture.

Timing in Offline Simulation with VEOS 24
Offline simulation on a PC is performed in a virtual simulation time which is independent of real time.

Simulating Multicore V-ECUs 27
VEOS lets you build and simulate V-ECUs accessing a multicore AUTOSAR operating system (OS). VEOS supports multicore OS APIs in offline simulations only for code-based V-ECUs, i.e., for V-ECUs with code files provided externally.

VEOS Modules 28
VEOS consists of different modules.

Basics on VEOS

Introduction

VEOS is dSPACE software for the code-based offline simulation of virtual ECUs and environment models on a PC.

VEOS components

VEOS basically consists of the following components:

VEOS System Builder The VEOS component for:

- Building a model implementation for offline simulation on VEOS.
- Integrating these model implementations into a simulation system.

VEOS Simulator The VEOS component for:

- Controlling the execution of offline simulations.
- Interconnecting all the VPUs of a simulation system during offline simulation.

VEOS Simulator is executed in a separate task of the PC's operating system.

VEOS Player The graphical user interface for VEOS on Windows.

VEOS Player provides an automation interface.

VEOS command line interface Lets you access the VEOS System Builder and the VEOS Simulator via the command line.

C code debugging during simulation

VEOS supports the debugging of C code during a running offline simulation executed on the PC. Since the virtual simulation time is the same for all the V-ECUs and environment VPUs in a simulation system, simulation results are the same irrespective of whether you step through the code with a debugger.

Refer to Debugging Source Code During Offline Simulation on page 201.

Support of 64-bit applications

Building and simulating 64-bit applications VEOS lets you build 64-bit applications from the following model implementations:

- Functional Mock-up Unit (FMU)
- Simulink implementation container (SIC)
- V-ECU implementation (VECU)

This allows you to integrate model implementations that contain 64-bit libraries.

Stimulating, measuring, and calibrating variables of 64-bit applications You can stimulate, measure, and calibrate variables of 64-bit applications executed on VEOS using dSPACE experiment software such as ControlDesk.

VEOS in the SIL testing tool chain

For information on the integration of VEOS in the SIL testing tool chain, refer to VEOS (SIL Testing Overview).

Related topics

Basics

Basics on Simulation Systems.....20
Basics on the VEOS Command Line Interface.....315
Basics on the VEOS Player.....30
Installation and Virtualization of VEOS.....21
VEOS Modules.....28

Basics on Simulation Systems

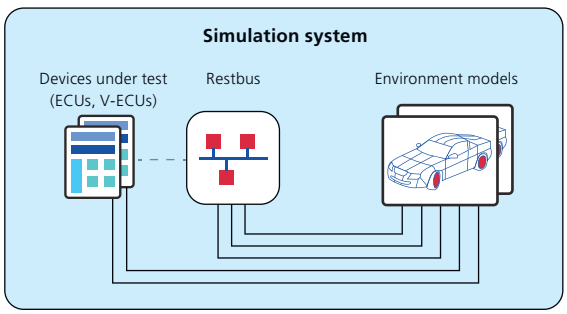
Introduction

VEOS lets you handle simulation systems that comprise executable representations of ECUs as well as a representation of their environment and all their interconnections.

Simulation system

A simulation system consists of the following elements:

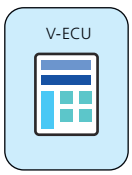
- Devices under test:
  - Virtual ECUs (V-ECUs) to be validated and tested
  - Real ECUs to be validated and tested
- Restbus
- Environment models
- Controller models
- Interconnections between these elements



You can build an offline simulation application for the simulation system to execute it on VEOS.

Virtual ECU

A virtual ECU (V-ECU) is software that represents a real ECU or parts of it in a simulation scenario. The V-ECU comprises components from the application software and optionally from the basic software. It provides functionalities comparable to those of a real ECU.

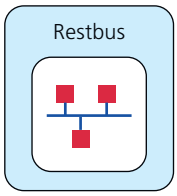


For more information, refer to Basics on Classic V-ECUs (SIL Testing Overview).

Restbus

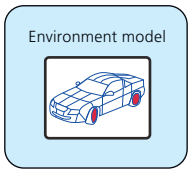
Usually, not all of the ECUs in a simulation system are devices under test (DUTs). In such a system, the ECUs under test can be connected to a *restbus* that simulates the bus communication of other ECUs of the simulation system.

Restbus simulation lets you test one or more ECUs while simulating the other ECUs of the related communication clusters. The restbus can simulate the bus communication of other ECUs of the simulation system. These restbus ECUs are combined and executed in a single application process, which simplifies the simulation system. The bus communication of the restbus can be manipulated, and the effects on the ECU(s) to be tested can be observed.



Environment model

An environment model represents a part of or all of an ECU's environment in a simulation scenario. Variable descriptions of environment models are based on SDF files.



Controller model

A controller model represents a control function of an ECU in a simulation scenario. Variable descriptions of controller models are based on A2L files.

Related topics

Basics

Basics on VEOS.....18
Simulation Architecture of VEOS.....23

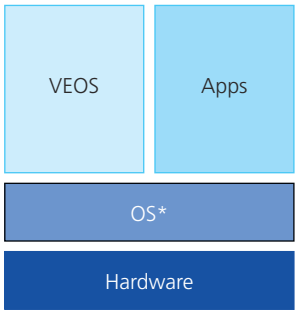
Installation and Virtualization of VEOS

Introduction

VEOS supports the following installation and virtualization setups.

VEOS on a real computer

You can install and run VEOS directly on a real computer.

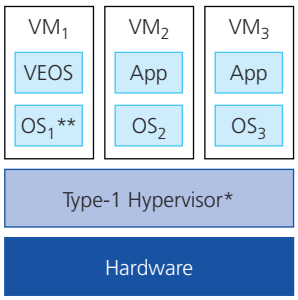


*	Supported operating systems: ▪ Windows / Windows Server For information on installing VEOS, refer to How to Install dSPACE Software (Installing dSPACE Software 🕮). ▪ Ubuntu Linux 18.04 LTS For information on installing VEOS, refer to How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS on page 292.

VEOS on a virtual machine

You can install and run VEOS on a virtual machine (VM).

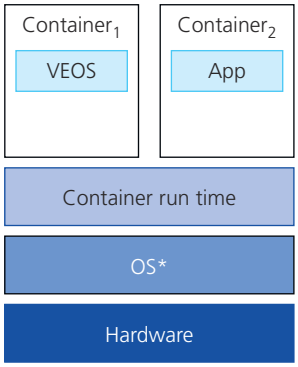
VM in Type-1 hypervisor



*	Supported Type-1 hypervisor: VMware ESXi hypervisor. For general compatibility information, refer to Using dSPACE Software on Virtual Machines (VMs) (Installing dSPACE Software 🕮). For compatibility information specific to adaptive V-ECU simulation, refer to Hypervisor Requirements for Adaptive V-ECU Simulation on page 239.
**	Supported operating systems: ▪ Windows / Windows Server For information on installing VEOS, refer to How to Install dSPACE Software (Installing dSPACE Software 🕮). ▪ Ubuntu Linux 18.04 LTS For information on installing VEOS, refer to How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS on page 292.

VEOS in a container

You can install and run VEOS in a container.



*	Supported operating system: Ubuntu Linux 18.04 LTS For information on container creation, refer to Basics on Running VEOS 5.2 in a Linux Docker Container on page 308.

Hypervisor for adaptive V-ECU simulation

Adaptive V-ECUs are simulated in a virtual machine.

For information on the hypervisor requirements, refer to Basics on Simulating Adaptive V-ECUs on page 236 and Hypervisor Requirements for Adaptive V-ECU Simulation on page 239.

Related topics

Basics

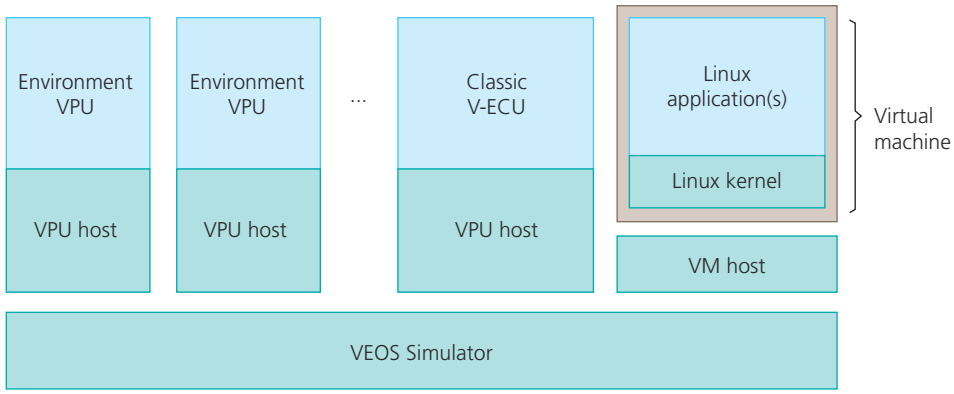
Basics on VEOS 18
Introduction to VEOS on Linux 290
Simulation on a Local or Remote Simulator 185

Simulation Architecture of VEOS

Basics on the VEOS simulation architecture

The VEOS simulation architecture is the basis for simulating the VPUs in a simulation system.

The following illustration shows an example of a simulation system with several VPUs.



VEOS Simulator VEOS Simulator is the central component of the simulation architecture. It controls the execution of offline simulations, and it connects all the VPUs in a simulation system.

VEOS Simulator is executed in a separate task of the PC's operating system.

VPUs Each VPU represents an element of a simulation system. The VEOS Simulator and the individual VPUs exchange data via the *VPU host* and the *VM host* in case of an adaptive V-ECU, respectively.

Each VPU is executed in a separate task of the PC's operating system.

Virtual machine VEOS lets you include Linux-based applications in a simulation using a virtual machine that is hosted by a hypervisor. The hypervisor communicates with the VEOS Simulator via the VM host.

A special case of Linux-based applications are adaptive applications, which can be assembled into an adaptive V-ECU. Refer to Basics on Simulating Adaptive V-ECUs on page 236.

Related topics

Basics

Basics on Simulating Adaptive V-ECUs 236
Basics on Simulation Systems 20

Timing in Offline Simulation with VEOS

Virtual simulation time

Offline simulation on a PC is performed in a *virtual simulation time* which is independent of real time.

This lets you:

- Execute offline simulations faster or slower than real time depending on the PC performance. You can also execute simulation runs in *soft real time*, for

example, to view a 3-D animation with a realistic timing. See Simulation in soft real time on page 26.

- Pause and resume offline simulations. For instructions, refer to How to Load and Run an Offline Simulation Application on page 188.

Zero execution time assumption The real execution time on the target hardware differs from the virtual simulation time: Because VEOS does not know the real execution time on the target hardware, VEOS applies the *zero execution time assumption* for offline simulation, which is common for most offline simulators such as Simulink.

According to the zero execution time assumption, all tasks are executed *instantly* on the virtual simulation time scale, i. e., all tasks are assumed to run with an execution time of $\Delta t_{Virtual} = 0$. As a consequence, reproducible simulation results that are independent of the target hardware can be achieved.

After the offline simulator has executed all the tasks for a discrete simulation event, it jumps to the next simulation event in the simulation queue.

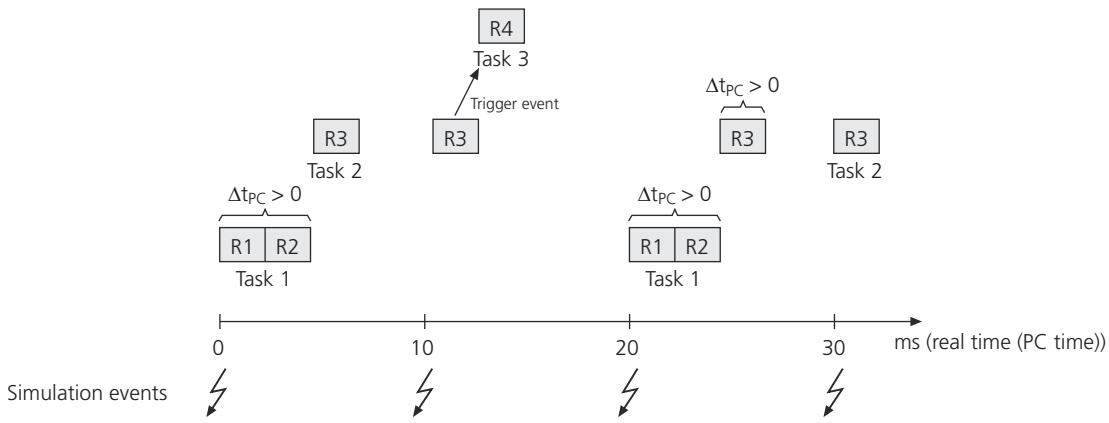
Note

You cannot analyze effects that depend on task execution times due to the zero execution time assumption.

Real time (PC time) The PC requires execution time to execute a task of the simulation, i.e., $\Delta t_{PC} > 0$. Depending on the PC performance, this execution time is normally faster than real time.

When you execute a simulation in *soft real time*, VEOS waits until the real time (PC time) is greater or equal to the virtual time of the next simulation event. Only then does it execute the next simulation event. As a result, the virtual time and the real time are approximated at the points in time of the simulation events.

Scheduling example The illustration below shows a scheduling example with three tasks. The simulation is executed in soft-real time, with the real-time acceleration factor set to 1.



Task 1	▪ Is triggered periodically by the operating system every 20 ms ▪ Calls runnables R1 and R2
Task 2	▪ Is triggered periodically by the operating system every 10 ms. Since task 2 has a lower priority, it is called *after* task 1. ▪ Calls runnable R3.
Task 3	▪ Is not triggered periodically but is activated from time to time by task 2 ▪ Calls runnable R4

When this example scenario is simulated, VEOS creates simulation events every 10 ms to trigger the tasks:

- At $t_{PC} = t_{Virtual} = 0$ ms, VEOS triggers task 1, and the runnables R1 and R2 are called. Then, VEOS triggers task 2, and runnable R3 is called.
- After runnable R3 returns, the virtual simulation time is still 0 ms ($t_{Virtual} = 0$) due to the *zero execution time assumption*, whereas the actual real time (PC time) has advanced by some ms depending on the PC performance.
- To execute the next simulation event, VEOS advances the simulated clock from $t_{Virtual} = 0$ to $t_{Virtual} = 10$ ms. Because the simulation is executed in soft real time, VEOS waits until the real time (PC time) is greater or equal to the virtual time ($t_{PC} \geq t_{Virtual} = 10$ ms). Then, it executes the next simulation event, which triggers Task 2 again.

Simulation in soft real time

VEOS lets you run offline simulations in *soft real time* by specifying a value that is not equal to 0 as the real-time acceleration factor. If you specify 1 as the factor, for example, VEOS tries to keep the virtual simulation time as close to real-time as possible with regard to the constraints of the host PC's operating system.

No loss of measurement and stimulation data To avoid the loss of measurement and stimulation data, VEOS automatically decelerates the simulation run if more measurement data is generated than the experiment software, e.g., ControlDesk, can sample. VEOS also decelerates the simulation if stimulation data is not received in time. This is independent of the specified real-time acceleration factor, i.e., receiving complete measurement data has higher priority than achieving the specified real-time acceleration factor.

Monitoring the deviation between real time and virtual simulation time The variable description (A2L or SDF/TRC) file of each VPU in an offline simulation application provides a variable that lets you monitor the deviation between real time and virtual simulation time.

Refer to Variable for monitoring the deviation between real time and virtual simulation time on page 190.

Warning when soft real time cannot be achieved During simulation in soft real time, VEOS tries to make up the time lost due to disturbances introduced by sporadic Windows events. If this is possible within a limited time, there is no remaining offset between the virtual simulation time and real time unless new disturbances occur.

If the deviation exceeds 1 s, however, VEOS issues a warning and no longer tries to make up for the lost time.

NOTICE

VEOS' soft real time simulation feature is not intended for coupling VEOS with real-time hardware, because the correct timing behavior cannot be guaranteed. Strong deviations from real time can occur, which can lead to unpredictable simulation results and even damage the real-time hardware.

Scheduling effects specific to V-ECUs

In an offline simulation, an *emulated AUTOSAR operating system* is used to run the simulation system's virtual ECUs (V-ECUs) on the PC. During the simulation build process, VEOS uses the AUTOSAR OS configuration of the V-ECUs to generate their operating systems. The operating system's main purpose is to trigger OS tasks, including the V-ECU code, correctly.

As a consequence, the scheduling behavior of a V-ECU application is completely determined by the V-ECU's AUTOSAR operating system, i.e., VEOS simulates the correct order of OS tasks depending on the configured priorities. It complies with the causal scheduling constraints.

Related topics

Basics

Basics on VEOS 18
Specifying the FMU Simulation Behavior 87

Simulating Multicore V-ECUs

Introduction

VEOS lets you build and simulate V-ECUs accessing a multicore AUTOSAR operating system (OS). VEOS supports multicore OS APIs in offline simulations only for code-based V-ECUs, i.e., for V-ECUs with code files provided externally.

Serial task processing on VEOS

On VEOS, the tasks of a multicore V-ECU cannot be calculated in parallel on several cores. Instead, the task processing in an offline simulation is based on the simulated AUTOSAR OS scheduler, which executes all tasks on one physical core of the simulator. The sequence of task execution depends on the scheduling policy. The VEOS implementation of the AUTOSAR OS scheduler respects the configured task priorities in each OS application, but other core-dependent effects cannot be simulated in VEOS.

However, because the deterministic behavior of the offline simulation is ensured, offline simulation of multicore V-ECUs is suitable for detecting errors in the V-ECU code.

Most multicore OS functions behave in the same way as on a real multicore operating system. However, some multicore OS functions are restricted by the serial processing of OS tasks on VEOS. For more information on the limitations, refer to Limitations for Loading and Running an Offline Simulation Application, and Debugging Source Code on page 232.

Related topics

Basics

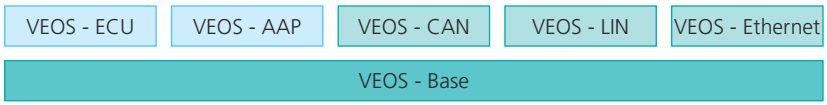
Limitations for Loading and Running an Offline Simulation Application, and Debugging Source Code 232

VEOS Modules

Introduction

VEOS consists of different modules.

The following illustration provides an overview of the available modules:



VEOS - Base

The VEOS base module for:

- Model integration and build
- Offline simulation of:
  - Environment VPUs
  - Controller VPUs
  - Bus VPUs

VEOS - ECU

A VEOS module for offline simulation of classic V-ECUs.

VEOS - AAP

A VEOS module for offline simulation of adaptive V-ECUs, i.e., V-ECUs according to the AUTOSAR Adaptive Platform.

VEOS - CAN

A VEOS module for offline simulation of CAN and CAN FD bus communication.

VEOS - Ethernet	A VEOS module for offline simulation of Ethernet communication.
VEOS - LIN	A VEOS module for offline simulation of LIN bus communication.
Required licenses	Working with the different modules requires that you have the related licenses. **Information on installation and licensing** For information on installing dSPACE software and handling dSPACE licenses, refer to What Do You Want To Do? (Installing dSPACE Software 🕮).
Related topics	Basics Basics on Simulation Systems....20 Basics on the VEOS Player....30 Basics on VEOS....18 Installation and Virtualization of VEOS....21

Introduction to VEOS Player

Where to go from here

Information in this section

Basics on the VEOS Player.....30
VEOS Player is the graphical user interface for VEOS.

How to Open an OSA File.....33
To work with a simulation system, you have to open the related OSA file.

User Interface of the VEOS Player.....35
Provides an overview of the VEOS Player's graphical user interface.

Basics on the VEOS Player

Introduction

VEOS Player is the graphical user interface for VEOS.

Features

The VEOS Player provides the following features:

- Creating or importing an offline simulation application
- Adding the following elements to an offline simulation application:
  - Virtual processing units (VPUs)
  - V-ECU implementations
  - FMUs
  - Simulink implementation containers
  - Bus simulation containers
- Connecting VPUs in the offline simulation application
- Specifying blocking or non-blocking VPU communication
- Displaying bus-specific elements of an offline simulation application, such as VPU communication controllers
- Configuring, downloading and running an offline simulation application

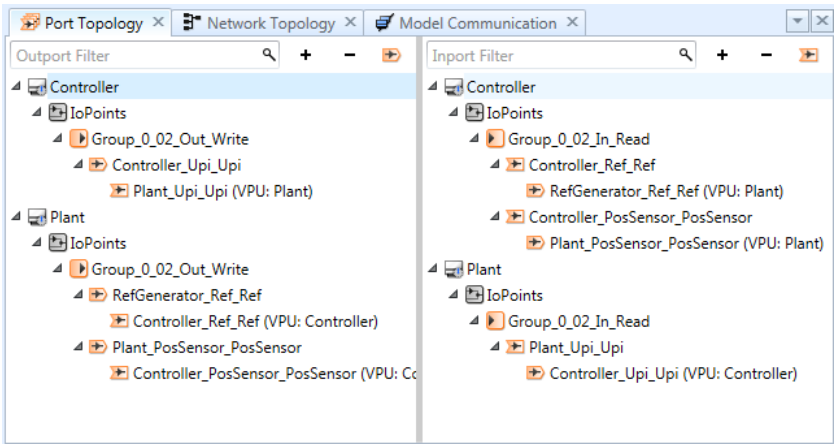
Offline simulation application

Basics on offline simulation applications An *offline simulation application (OSA)* is an application for simulating VPUs — virtual ECUs (V-ECU), environment VPUs, and bus VPUs — on a PC. When you build a simulation system, an OSA file is generated. The file contains the configuration of a simulation and the simulation system's VPUs.

When you open an OSA file in the VEOS Player, information on the simulation system and on the VPUs is displayed.

Displaying the application in the Port Topology pane VPUs and signals of a simulation system are displayed in the Port Topology pane.

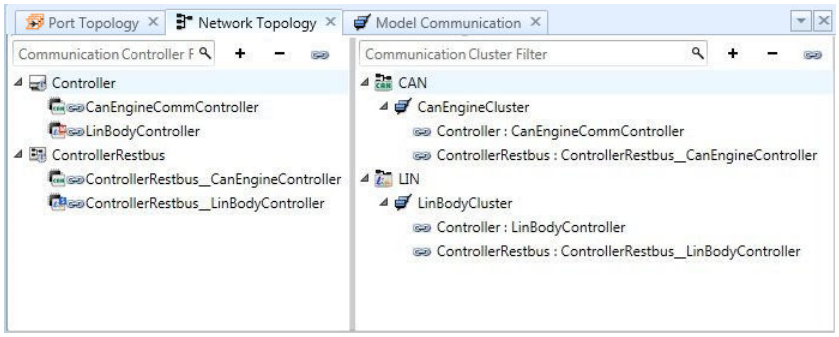
The following illustration shows an example of a simulation system with two VPUs (Plant and Controller).



Displaying the application in the Network Topology pane Communication controllers and clusters of a simulation system are displayed in the Network Topology pane.

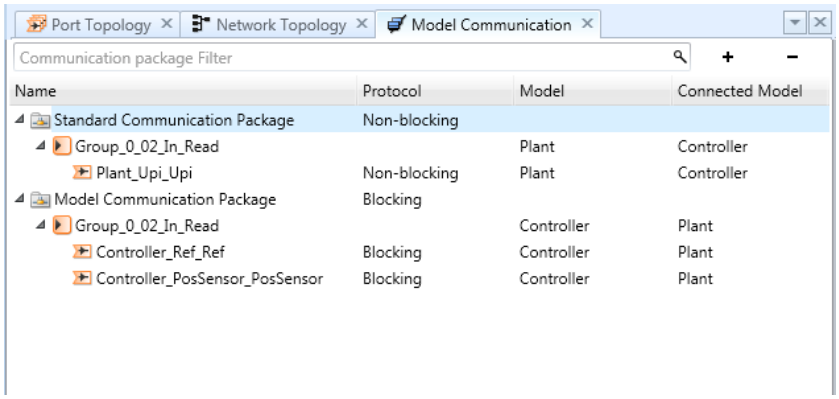
The following illustration shows an example of a simulation system that contains a V-ECU named Controller and a bus VPU named ControllerRestbus:

- The CAN controllers of the V-ECU and bus VPU are connected via the CAN communication cluster named CanEngineCluster.
- The LIN controllers of the V-ECU and bus VPU are connected via the LIN communication cluster named LinBodyCluster.



Displaying the application in the Model Communication pane Communication packages of a simulation system are displayed in the Model Communication pane. Communication packages specify the communication protocol type of the ports they are containing.

The following illustration shows a simulation system with two communication packages (Standard Communication Package and Model Communication Package) as an example.



Connecting VPUs

The VEOS Player lets you connect the VPUs in an offline simulation application via *signals*.

Signals allow direct communication between VPUs without specifying and using a network communication. Signals let you connect ECUs for offline simulations, for example, to simulate controller models on different ECUs and environment models on different environment VPUs.

Refer to Basics on Signal Connection Dependencies on page 100.

Connecting communication controllers and clusters in a simulation system

VEOS Player lets you connect communication controllers and clusters of the VPUs in an offline simulation application.

Connections between communication controllers and clusters allow VPU communication via a network communication.

Refer to Connecting Communication Controllers to Communication Clusters on page 111.

Configuring a simulation and saving the configuration

Configuring a simulation Using the VEOS Player, you can change:

- The simulation configuration in the simulation system. For example, you can specify the simulation duration and a factor to accelerate or decelerate simulation runs in relation to real time.
- The configuration of the VPUs in the simulation system. For example, you can specify the timer drift and offset for each VPU individually.

Saving the configuration You can save the configuration of the simulation system as an offline simulation application (OSA) file.

Downloading and starting a simulation

Using the VEOS Player, you can download an OSA file to VEOS and start an offline simulation on the PC.

For instructions, refer to How to Load and Run an Offline Simulation Application on page 188.

Automating the VEOS Player

You can automate the VEOS Player. For details, refer to Basics of the VEOS Player Automation Interface on page 326.

Related topics

Basics

User Interface of the VEOS Player 35

HowTos

How to Open an OSA File 33

How to Open an OSA File

Objective

To work with a simulation system, you have to open the related OSA file.

Preconditions

The VEOS Player is running.

Method

To open an OSA file

1. From the File ribbon, select Open.
2. In the Open dialog, select an OSA file and click Open to open the contained simulation system.

Result

The selected OSA file is opened in the **VEOS Player**. Elements of the simulation system are displayed in the panes of VEOS Player.

Tip

You can also open an OSA file by

- Dragging it from the file system to the opened **VEOS Player** (if no OSA file is open), or to the **VEOS Player**'s desktop icon.
- Double-clicking it in File Explorer.
- Selecting **Open** from the **Edit** ribbon.

Next steps

- You can import further elements to the system.

  Refer to Basics on Integrating the Simulation System on page 62.
- You can connect unconnected VPUs.

  Refer to Basics on Signal Connection Dependencies on page 100.
- You can specify blocking or non-blocking VPU communication.

  Refer to Specifying Blocking or Non-Blocking Communication on page 109.
- You can connect unconnected communication controllers to communication clusters.

  Refer to Connecting Communication Controllers to Communication Clusters on page 111.

Related topics

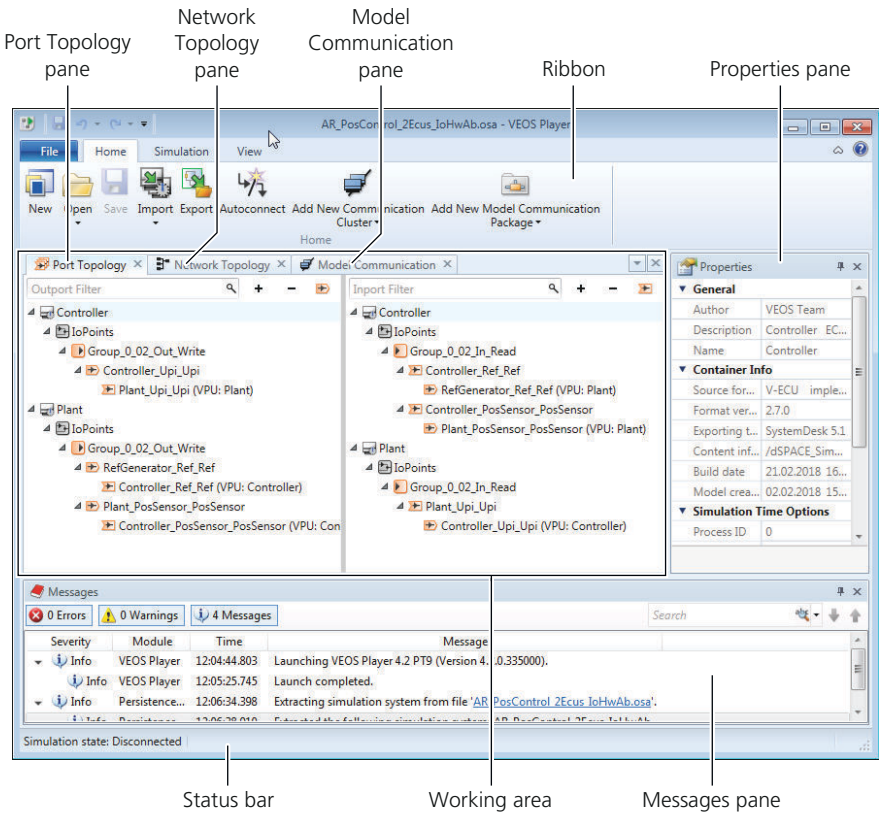
References

Open..... 159

User Interface of the VEOS Player

Overview

The following illustration shows the user interface of the **VEOS Player**.



Panes

Messages pane The **Messages** pane provides a history of all the info, advice, error and warning messages, and all the questions that occur when you work with the product. This helps you check the system state.

For reference information, refer to Messages Pane on page 401.

Model Communication pane The **Model Communication** pane displays the communication packages of the simulation system.

For reference information, refer to Model Communication Pane on page 116.

Network Topology pane The **Network Topology** pane displays the bus simulation configuration of the simulation system.

For reference information, refer to Network Topology Pane on page 118.

Port Topology pane The **Port Topology** pane displays the VPUs and signals of the simulation system.

For reference information, refer to Port Topology Pane on page 120.

Properties pane Via the **Properties** pane, you can view and change the properties of the element currently selected in one of the panes.

For reference information, refer to Elements and Properties on page 122.

Ribbon

Via the ribbon, you can quickly access the VEOS Player commands to edit and simulate the configuration.

For information on using the ribbon, refer to Basics on Ribbons on page 384.

Status bar

Displays the following information from left to right (if available):

- Simulation state (disconnect, running, paused, stopped)
- Simulation time in seconds
- Elapsed real time in seconds

Related topics

References

Commands and Dialogs 134
Elements and Properties 122
Messages Pane 401
Model Communication Pane 116
Network Topology Pane 118
Port Topology Pane 120

Demos for VEOS

Demos for VEOS

Windows demos and sample scripts

The VEOS on Windows installation provides the following demo ZIP files:

Demo ZIP File	Demo Contents
`1Ecu_Standard.zip`[1)]	The demo ZIP file provides the following: ▪ The `AR_PosControl_1Ecu` simulation system with one V-ECU for simulating AUTOSAR software The A2L variable description file of the V-ECU and the original V-ECU implementation are also included in the demo. ▪ The `LoadAndRun.py` Python script[2)] to show how to: ▪ Open an offline simulation application in the VEOS Player ▪ Change some configuration settings of the offline simulation application ▪ Start and control an offline simulation by using VEOS Player ▪ Automate the handling of VEOS Player events. ▪ The `VEOS_SimulationEventsLib.py` Python event library. You can use the library to automate VEOS Player event handling with Python. For details, refer to Automating VEOS Player Event Handling with Python on page 228.
`2Ecu_CanBusSimulation.zip`[1)]	The demo ZIP file provides the following: ▪ The `AR_PosControl_2Ecus_Can` simulation system with two V-ECUs that are interconnected via a CAN bus The A2L variable description files of the V-ECUs and the original V-ECU implementations are also included in the demo. ▪ The `LoadAndImport.py` Python script[2)] to show how to: ▪ Create an offline simulation application ▪ Import all the VPUs from an OSA file via automation ▪ Remove selected VPUs from an offline simulation application ▪ Import selected VPUs via their names from an OSA file via automation
`2Ecu_LinBusSimulation.zip`[1)]	The demo ZIP file provides the following: ▪ The `AR_PosControl_2Ecus_Lin` simulation system with two V-ECUs that are interconnected via a LIN bus The A2L variable description files of the V-ECUs and the original V-ECU implementations are also included in the demo. ▪ The `LoadAndImport.py` Python script[2)] to show how to: ▪ Create an offline simulation application ▪ Import all the VPUs from an OSA file via automation ▪ Remove selected VPUs from an offline simulation application ▪ Import selected VPUs via their names from an OSA file via automation
`2Ecu_VpuPorts.zip`[1)]	The demo ZIP file provides the following: ▪ The `AR_PosControl_2Ecus_VpuPorts` simulation system with two connected V-ECUs The A2L variable description files of the V-ECUs and the original V-ECU implementations are also included in the demo. ▪ The `LoadAndConnect.py` Python script[2)] to show how to connect VPUs of V-ECUs via automation

Demo ZIP File	Demo Contents
`AdaptiveSamples.zip`[1]	The demo ZIP file provides the `AdaptiveSamples` folder, which contains the VEOS adaptive platform demos for illustrating the handling of adaptive V-ECUs in the dSPACE tool chain. Refer to Working with the VEOS Adaptive Platform Demos on page 248.
`AR_Fuelsys_Smc.zip`[1]	The demo ZIP file provides the following: ▪ The `AR_Fuelsys.smc` system model container (SMC) file. This container file provides two V-ECU implementations and one Simulink implementation container. The two V-ECUs are interconnected via a CAN bus. One V-ECU is connected to the Simulink model. When you import the SMC container file, VEOS Player automatically establishes port and bus connections between VPUs.

`ThrottleControl.zip`[1]	The demo ZIP file provides the following:
	▪ The `ThrottleControl.vecu` V-ECU implementation (VECU) file for simulating non-AUTOSAR software
	▪ The `LoadAndImport.py` Python script[2] to show how to:
	  ▪ Create an offline simulation application
	  ▪ Import and build a V-ECU implementation via automation
	  ▪ Start and control an offline simulation using VEOS Player
	  ▪ Get version information on VEOS Player

[1] To copy the demo files to a user-specific folder, use the Copy Demo Files command.
[2] You can run the script in an external interpreter, such as PythonWin.

Features covered by the Windows demos

The following table lists important features, and shows in which demo each of them is covered (✓) or not (—).

Feature	1Ecu_Standard	2Ecu_CanBusSimulation	2Ecu_LinBusSimulation	2Ecu_VpuPorts	AdaptiveSamples	AR_Fuelsys_Smc	MDAutomotive	ThrottleControl
CAN bus communication	—	✓	—	—	—	✓	—	—
LIN bus communication	—	—	✓	—	—	—	—	—
Ethernet bus communication	—	—	—	—	✓	—	—	—
VPU connection via signals	—	—	—	✓	—	✓	—	—
Integration of the MotionDesk service	—	—	—	—	—	—	✓	—
Simulation of adaptive V-ECUs	—	—	—	—	✓	—	—	—

Demos for automating the VEOS Player The following table lists features of the VEOS Player's automation interface, and shows in which demo each of them is covered (✓) or not (—).

Feature	1Ecu_Standard	2Ecu_CanBusSimulation	2Ecu_LinBusSimulation	2Ecu_VpuPorts	AdaptiveSamples	AR_Fuelsys_Smc	MDAutomotive	ThrottleControl
Loading an offline simulation application	✓	✓	✓	✓	✓	—	—	✓
Importing V-ECU implementations	—	—	—	—	✓	—	—	✓
Importing Simulink implementation containers	—	—	—	—	—	—	✓	—
Importing VPUs	—	✓	✓	—	—	—	—	—
Connecting VPUs	—	—	—	✓	✓	—	—	—
Starting and controlling an offline simulation	✓	—	—	—	✓	—	—	✓
Getting version information on VEOS Player	—	—	—	—	—	—	—	✓
Handling VEOS Player events	✓	—	—	—	—	—	—	—

Decrypting Windows demos and copying them to a user-specific folder

You can work with the Windows demos only if they have been decrypted in dSPACE Installation Manager. Decryption has to be performed only once. For information on decryption, refer to How to Decrypt Encrypted Archives of dSPACE Software Installations (Managing dSPACE Software Installations 🕮).

The Copy Demo Files command lets you copy the demo files to a user-specific folder.

Linux demos

The VEOS on Linux installation includes the GPU passthrough demo, which demonstrates how a graphics processing unit (GPU) can be used by an adaptive V-ECU in an offline simulation application.

Refer to Working with the GPU Passthrough Demo on page 271.

Related topics

References

Copy Demo Files..... 392

Compatibility, Migration and Discontinuations

Where to go from here

Information in this section

Compatibility of VEOS.....40
Migrating from Prior Versions of VEOS.....43
Discontinuations in VEOS.....55

Compatibility of VEOS

Compatibility of VEOS 5.2

General compatibility

dSPACE recommends using only software products from the same dSPACE Release. This ensures maximum run-time compatibility.

Supported compiler versions

For information on supported compiler versions, refer to Basics on Integrating the Simulation System on page 62.

Supported operating systems

Windows For information on the Windows operating systems supported by VEOS, refer to Operating System (Installing dSPACE Software 🕮).

Linux For information on the Linux distribution recommended for VEOS, refer to Introduction to VEOS on Linux on page 290.

BSC file compatibility

VEOS 5.2 is compatible with bus simulation container (BSC) files created with the Bus Manager of dSPACE Release 2021-A (BSC version 1.10).

- If a BSC file was generated *without an SIC file*, or *with an SIC file that was generated for the dsrt.tlc system target file*, you can integrate the BSC file in a VEOS simulation on Windows.
- If a BSC file was generated *with an SIC file that was generated for the dsrt64.tlc system target file*, you can integrate the BSC file in a VEOS simulation on Linux.

CTLGZ/VECU file compatibility

The following table shows the compatibility between VEOS 5.2 and V-ECU implementation container (CTLGZ/VECU) files:

V-ECU Implementations Created With...	V-ECU Implementation Version
dSPACE Release 2021-A and 2020-B: ▪ SystemDesk 5.5 ▪ TargetLink 5.1	3.0[1)]
dSPACE Release 2020-A and 2019-B: ▪ SystemDesk 5.4 ▪ TargetLink 5.0	2.10[1)]

1) You can integrate files of this version in a VEOS simulation on Windows and in a VEOS simulation on Linux.

FMU file compatibility

- VEOS supports Functional Mock-up Units (FMUs) that comply with the FMI 2.0 standard.
- VEOS supports only the *FMI for Co-Simulation interface*, but not the FMI for Model Exchange interface.

You can integrate an FMU file in a VEOS simulation on Windows and in a VEOS simulation on Linux.

For detailed and up-to-date compatibility information on dSPACE FMI support, refer to:

http://www.dspace.com/go/FMI-Compatibility.

OSA file compatibility

The following table shows the compatibility between VEOS 5.2 and offline simulation application (OSA) files:

OSA Files Created with Products Of ...	OSA Version
dSPACE Release 2021-A	5.2
dSPACE Release 2020-B	5.1[1)]
dSPACE Release 2020-A	5.0[1)]
dSPACE Release 2019-B	4.5[1)]

1) You cannot modify the properties of VPUs contained in an OSA file if you open the OSA file in a later VEOS version than the version with which the OSA file was originally created. However, you can edit port and network connections. Therefore, it is recommended to rebuild the binary OSA files from existing model implementation container files (CTLGZ/VECU, SIC, BSC, FMU) when you migrate from one VEOS version to another.

SIC file compatibility

The following table shows the compatibility between VEOS 5.2 and Simulink implementation container (SIC) files:

SIC Files Created With ...	SIC Version[1)]
dSPACE Release 2021-A: ▪ Model Interface Package for Simulink 4.5 ▪ TargetLink 5.1	1.10
dSPACE Release 2020-B: ▪ Model Interface Package for Simulink 4.4 ▪ TargetLink 5.1	1.9[2)]
dSPACE Release 2020-A: ▪ Model Interface Package for Simulink 4.3 ▪ TargetLink 5.0	1.8[2)]
dSPACE Release 2019-B: ▪ Model Interface Package for Simulink 4.2 ▪ TargetLink 5.0	1.7[2)]

1) If an SIC file was generated for the `dsrt.tlc` system target file, you can integrate it in a VEOS simulation on Windows. If an SIC file was generated for the `dsrt64.tlc` system target file, you can integrate it in a VEOS simulation on Linux. Refer to How to Generate a Simulink Implementation Container (Model Interface Package for Simulink - Modeling Guide).

2) If the SIC file is created with a previous dSPACE Release and if the SIC file contains an ASM model, you cannot simulate the model in VEOS 5.2 (dSPACE Release 2021-A). For more information, refer to Migrating ASM Models on page 55.

SMC file compatibility

The following table shows the compatibility between VEOS 5.2 and system model container (SMC) files:

SMC Files Created With ...	SMC Version
dSPACE Release 2021-A: ▪ SYNECT 2.11 ▪ VEOS 5.2	1.2
dSPACE Release 2020-B: ▪ SYNECT 2.10 ▪ VEOS 5.1	1.2
dSPACE Release 2020-A: ▪ SYNECT 2.9 ▪ VEOS 5.0	1.2
dSPACE Release 2019-B: ▪ SYNECT 2.8 ▪ VEOS 4.5	1.1

You also have to consider the following compatibility restrictions of the individual container files contained in the SMC file to be imported: If the SMC file contains a container of an unsupported version, VEOS 5.2 imports neither the unsupported container nor the connections to the application process based on the unsupported container.

Real-Time Testing compatibility

To use RTT in connection with VEOS and ControlDesk, the Real-Time Testing (RTT) version used by the VEOS Simulator that runs the simulation system and the RTT version that is active on the PC must be identical.

The following table shows the VEOS Simulator version and the corresponding RTT version:

VEOS Simulator	RTT Version
... of VEOS 5.2	Real-Time Testing Version 5.0
... of VEOS 5.1	Real-Time Testing Version 4.4
... of VEOS 5.0	Real-Time Testing Version 4.3
... of VEOS 4.5	Real-Time Testing Version 4.2
... of VEOS 4.4	Real-Time Testing Version 4.1
... of VEOS 4.3	Real-Time Testing Version 4.0
... of VEOS 4.2	Real-Time Testing Version 3.4
... of VEOS 4.1	Real-Time Testing Version 3.3
... of VEOS 4.0	Real-Time Testing Version 3.2

ControlDesk 7.4 automatically uses the VEOS Simulator of VEOS 5.2. You can therefore use RTT in connection with VEOS and ControlDesk if RTT 5.0 is active on the PC.

AUTOSAR Adaptive Platform compatibility

For the simulation of AUTOSAR adaptive V-ECUs, VEOS 5.2 is compatible with the following release of the AUTOSAR Adaptive Platform:

- R19-11

Related topics

Basics

Hypervisor Requirements for Adaptive V-ECU Simulation..... 239

Migrating from Prior Versions of VEOS

Where to go from here

Information in this section

Migrating from VEOS 3.4 to 3.5..... 45
To migrate from VEOS 3.4 to VEOS 3.5, you might have to carry out additional migration steps.

Migrating from VEOS 3.5 to 3.6 45
To migrate from VEOS 3.5 to VEOS 3.6, you might have to carry out additional migration steps.

Migrating from VEOS 3.6 to 3.7 46
To migrate from VEOS 3.6 to VEOS 3.7, you might have to carry out additional migration steps.

Migrating from VEOS 3.7 to 4.0 48
To migrate from VEOS 3.7 to VEOS 4.0, you might have to carry out additional migration steps.

Migrating from VEOS 4.0 to 4.1 48
To migrate from VEOS 4.0 to VEOS 4.1, you might have to carry out additional migration steps.

Migrating from VEOS 4.1 to 4.2 49
To migrate from VEOS 4.1 to VEOS 4.2, you might have to carry out additional migration steps.

Migrating from VEOS 4.2 to 4.3 51
To migrate from VEOS 4.2 to VEOS 4.3, you might have to carry out additional migration steps.

Migrating from VEOS 4.3 to 4.4 52
To migrate from VEOS 4.3 to VEOS 4.4, you might have to carry out additional migration steps.

Migrating from VEOS 4.4 to 4.5 52
To migrate from VEOS 4.4 to VEOS 4.5, you might have to carry out additional migration steps.

Migrating from VEOS 4.5 to 5.0 53
To migrate from VEOS 4.5 to VEOS 5.0, you might have to carry out additional migration steps.

Migrating from VEOS 5.0 to 5.1 53
To migrate from VEOS 5.0 to VEOS 5.1, you might have to carry out additional migration steps.

Migrating from VEOS 5.1 to 5.2 54
To migrate from VEOS 5.1 to VEOS 5.2, you might have to perform additional migration steps.

Migrating ASM Models 55
You cannot simulate an ASM model on VEOS if the model is contained in an OSA or SIC file that was created with a dSPACE Release earlier than the one to which your VEOS installation belongs.

Migrating from VEOS 3.4 to 3.5

Introduction

To migrate from VEOS 3.4 to VEOS 3.5, you might have to carry out additional migration steps.

Changed automation behavior when importing a model implementation to the VEOS Player

In VEOS 3.5, there is an incompatible change of the `Import` method of the `IProject` interface of the VEOS Player API. The method lets you import model implementations such as SIC and CTLGZ files to a VEOS Player project:

- Up to and including VEOS 3.4, the `Import` method provides a list of all compiler messages as a return parameter of the `<System.String[]>` data type.
- As of VEOS 3.5, the `Import` method provides build information via the `IBuildResult`.

Refer to `IBuildResult` on page 345.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 3.5 to 3.6

Introduction

To migrate from VEOS 3.5 to VEOS 3.6, you might have to carry out additional migration steps.

Changed behavior when importing an FMU to VEOS Player

As of VEOS 3.6, the import behavior for Functional Mock-up Units (FMUs) is as follows:

If the FMU to be imported contains C source code files and a DLL compiled for Windows, VEOS Player uses the latter for the build process.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 3.6 to 3.7

Introduction

To migrate from VEOS 3.6 to VEOS 3.7, you might have to carry out additional migration steps.

Display and connectability of array VPU ports

The display and connectability of array VPU ports has changed from VEOS 3.6 to VEOS 3.7.

Up to and Including VEOS 3.6	As of VEOS 3.7
Display	
Array VPU ports and their elements were displayed in the VEOS Player with the / icon.	Array VPU ports (and scalar VPU ports) are displayed in the VEOS Player with the / icon. Individual elements of an array VPU port are no longer displayed. Instead, the number of elements is displayed as the array VPU port's **Data width** property. To specify initial values for the individual elements of an array VPU port, enter a comma-separated list in the **Initial value** property edit field of the port.
Connectability	
▪ Connecting an entire array VPU port was possible. ▪ Connecting the elements of an array VPU port individually was possible.	▪ Connecting an entire array VPU port is possible. ▪ Connecting the elements of an array VPU port individually is not possible.[1]

[1] In VEOS 3.7, when you open an offline simulation application last saved with VEOS 3.6 or earlier, the connections between individual elements of array VPU ports are kept.

VEOS Player automation changes

You have to consider the following changes to the automation interface of the VEOS Player in VEOS 3.7.

Up to and Including VEOS 3.6	As of VEOS 3.7
Accessing VPU ports was possible via the `VpuPorts` property of the `IVpu` interface.	The `VpuPorts` property in the `IVpu` interface lets you access only the VPU ports that are at the root level of a VPU. Accessing VPU ports at deeper levels is not possible using this approach.

Up to and Including VEOS 3.6	As of VEOS 3.7
	Instead, you can use one of the following approaches: ▪ The `IVpuPorts` and the `IVpuPortGroups` interfaces provide an `Item` method that lets you return the underlying `IVpuPort` and `IVpuPortGroup` interfaces. ▪ As an alternative, the `IVpu` and the `IVpuPortGroup` interfaces provide an `Item` method that lets you return underlying generic `IVpuItem` interfaces. Via the `VpuItemType` property in the `IVpuItem` interface, you can get the item type, either a VPU port or a VPU port group. This change is due to the hierarchical display of VPU ports supported as of VEOS 3.7. Refer to `IVpu` on page 368 and `IVpuPortGroup` on page 374.
The type of the `SendsTo` property of the `IVpuPort` interface was `System.String[]`.	The type of the `SendsTo` property of the `IVpuPort` interface is `IVpuPorts`. This change is due to the hierarchical display of VPU ports supported as of VEOS 3.7.
The type of the `ReceivesFrom` property of the `IVpuPort` interface was `String`.	The type of the `ReceivesFrom` property of the `IVpuPort` interface is `IVpuPort`. This change is due to the hierarchical display of VPU ports supported as of VEOS 3.7.
Accessing individual elements of an array VPU port was possible via the `DataElements` property of the `IVpuPort` interface.	Accessing individual elements of an array VPU port is no longer possible. As a consequence, the `DataElements` property of the `IVpuPort` interface is no longer available. Instead, the number of elements of an array VPU port is available via the `DataWidth` property of the `IVpuPort` interface.
To get version information, the `Version` property of the `IApplication` interface could be used.	To get version information, the `IVersionInformation` interface can be used. Refer to `IVersionInformation` on page 368. **Note** The `Version` property of the `IApplication` interface is still supported for compatibility reasons. However, it will be discontinued in the future.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

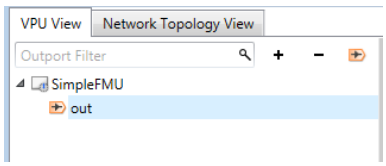
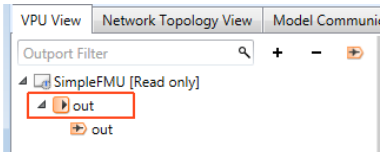
Migrating from VEOS 3.7 to 4.0

Introduction

To migrate from VEOS 3.7 to VEOS 4.0, you might have to carry out additional migration steps.

Changed hierarchy for VPU ports

The display of VPU ports has changed from VEOS 3.7 to VEOS 4.0 in some cases.

Up to and Including VEOS 3.7	As of VEOS 4.0
Up to and including VEOS 3.7, it was possible for VPU ports to be located *directly below a VPU*. The following illustration shows an example:  	As of VEOS 4.0, VEOS Player automatically adds one VPU port group to each VPU port if the VPU port is originally located directly below a VPU. The VPU port group gets the name of the VPU port. This change was necessary because with VEOS 4.0 it is possible to block communication between VPU ports. Specifying blocking/non-blocking communication between VPU ports, however, requires superordinate VPU port groups. The following illustration shows an example: 

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 4.0 to 4.1

Introduction

To migrate from VEOS 4.0 to VEOS 4.1, you might have to carry out additional migration steps.

Changed behavior when importing elements to VEOS Player via automation

Changed behavior when importing elements to VEOS Player via its user interface

- Up to and including Version 4.0, VEOS Player automatically saved the project after you imported, for example, a V-ECU implementation using the Import command.
- As of Version 4.1, VEOS Player does not automatically save the project after you use the Import command.

  Use the Save command after the import.

Changed behavior when importing elements to VEOS Player via automation The behavior of the following methods of the `IProject` interface was changed in VEOS 4.1:

- `Import`
- `ImportNamedVpu`
- `ImportVpus`

Using these methods, you can import container files and OSA files to the project that is open in VEOS Player.

- Up to and including Version 4.0, VEOS Player automatically saved the project after you used one of the `Import` methods listed above.
- As of Version 4.1, VEOS Player does not automatically save the project after you use one of the `Import` methods listed above.

  Use the `Save` method of the `IProject` interface after the import.

Refer to `IProject` on page 359.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 4.1 to 4.2

Introduction

To migrate from VEOS 4.1 to VEOS 4.2, you might have to carry out additional migration steps.

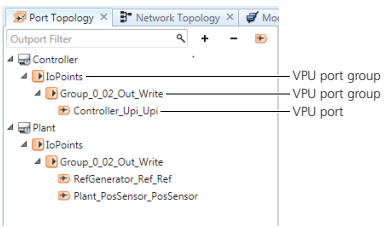
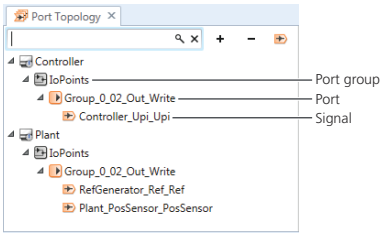
Changed hierarchy of VPU connections

VEOS Player's Port Topology pane lets you connect the VPUs of an offline simulation application to establish inter-VPU communication.

The hierarchy of VPU subelements such as VPU ports used to establish inter-VPU communication has been changed from VEOS 4.1 to VEOS 4.2 as described in the following table:

Up to and Including VEOS 4.1	**As of VEOS 4.2**
VPU port group (,)	One of the following: ▪ Port group ( ) (if the original VPU port group had a subordinate VPU port group) ▪ Port (,) (if the original VPU port group had no subordinate VPU port group)
VPU port structure (,)	Signal group (,)
VPU port (,)	One of the following: ▪ Port (,) and signal (,) (both have the same name) (if the original VPU port was located directly below the VPU or contained in a VPU port group) ▪ Signal (,) (if the original VPU port was not contained in a VPU port group)

Example As an example, the following illustrations show the Port Topology pane with the child elements of two VPUs in VEOS 4.1 and in VEOS 4.2:

Port Topology Pane in VEOS 4.1	**Port Topology Pane in VEOS 4.2**
	

Automatic migration In VEOS Player (VEOS 4.2), when you open an offline simulation application last saved with VEOS 4.1 or earlier, the VPU subelements are automatically migrated as described in the table above. All the connections

between VPUs are kept during migration so that you do not have to carry out any migration step.

Connecting VPUs via automation You can also connect VPUs via the VEOS Player automation interface. Unlike the graphical user interface that was changed as described above, the automation interface was not changed from VEOS 4.1 to VEOS 4.2. As a consequence, you do not have to migrate existing automation scripts.

For details, refer to Automating the Connection of VPUs on page 172.

Migration issue when importing container files with static libraries compiled with VEOS 4.1 or earlier

You have to consider the following migration issue when you import a CTLGZ or SIC container file containing static libraries to VEOS 4.2 or later.

Problem When you import a container file containing static libraries compiled with VEOS 4.1 or earlier to VEOS 4.2 or later, VEOS 4.2 or later can link the libraries and build an application successfully, without build warnings. However, the download of the offline simulation application might fail, or the results of the simulation might be incorrect. The simulation might even crash.

Description The VEOS default compiler settings used for struct alignment were changed in VEOS 4.2 according to the compiler default settings.

Solution To import a container file that contains static libraries with the compiler settings of VEOS 4.2 or later, you have to also recompile these libraries also with VEOS 4.2 or later.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 4.2 to 4.3

Introduction

To migrate from VEOS 4.2 to VEOS 4.3, you might have to carry out additional migration steps.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 4.3 to 4.4

Introduction

To migrate from VEOS 4.3 to VEOS 4.4, you might have to carry out additional migration steps.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 4.4 to 4.5

Introduction

To migrate from VEOS 4.4 to VEOS 4.5, you might have to carry out additional migration steps.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 4.5 to 5.0

Introduction

To migrate from VEOS 4.5 to VEOS 5.0, you might have to carry out additional migration steps.

Changed address range for 64-bit applications

Up to and including VEOS 4.5, the simulation of 64-bit applications was limited to the lower 32-bit address range. As of VEOS 5.0, simulations of 64-bit applications use the entire 64-bit address range.

As a result, if the model implementation to build contains code that uses typecasting of memory addresses to 32 bits, the simulation might lead to unexpected results or crash if you build the model implementation with a 64-bit compiler.

To solve the problem, you have two options:

- Build the model implementation with a 32-bit compiler. This is not supported for Linux operating systems.
- Adapt the code contained in the model implementation to a 64-bit simulation target, then build the model implementation with a 64-bit compiler.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 5.0 to 5.1

Introduction

To migrate from VEOS 5.0 to VEOS 5.1, you might have to carry out additional migration steps.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating from VEOS 5.1 to 5.2

Introduction

To migrate from VEOS 5.1 to VEOS 5.2, you might have to perform additional migration steps.

Note

To migrate to VEOS 5.2 from versions earlier than 5.1, you might also have to perform the migration steps of the intervening VEOS versions.

Changed naming conventions for V-ECUs

The naming conventions for V-ECUs based on the VECU container file format (V-ECU implementation version 3.0 introduced with dSPACE Release 2020-B) have been changed:

- VEOS 5.1 derived the V-ECU name *from the container file name of the related container*.
- As of Version 5.2, VEOS derives the V-ECU name *from the model identifier in the related container*.

Migrating to Python 3.9

As of dSPACE Release 2021-A, dSPACE products no longer support Python 3.6. Instead, Python 3.9 is supported.

There are no VEOS-specific migration aspects.

For general migration aspects, refer to Migrating Python Scripts from Python 3.6 to Python 3.9 on page 332.

Migrating ASM models

There is a migration issue when you simulate ASM models on VEOS.

For more information, refer to Migrating ASM Models on page 55.

Related topics

Basics

Compatibility of VEOS 5.2 40
Migrating ASM Models 55

Migrating ASM Models

Migrating ASM models

You cannot simulate an ASM model on VEOS if the model is contained in an OSA or SIC file that was created with a dSPACE Release earlier than the one to which your VEOS installation belongs.

To simulate an ASM model that was last saved with a dSPACE Release earlier than the dSPACE Release to which your VEOS version belongs, perform the following steps:

1. Migrate the ASM model to the dSPACE Release to which your VEOS version belongs.

   For information on migrating ASM models, refer to Migrating ASM Models (ASM User Guide 🕮).
2. Generate a Simulink implementation container (SIC) file on the basis of the ASM model by using the *Model Interface Package for Simulink*.

   For instructions, refer to Generating Simulink Implementation Containers (Model Interface Package for Simulink - Modeling Guide 🕮).
3. Import the SIC file to the VEOS Player of your VEOS version.

   For instructions, refer to How to Import Simulink Implementations on page 71.

Discontinuations in VEOS

Where to go from here

Information in this section

Discontinuations as of VEOS 3.5 56
Provides information on the features discontinued as of VEOS 3.5.

Discontinuations as of VEOS 4.0 57
Provides information on the features discontinued as of VEOS 4.0.

Discontinuations as of VEOS 4.1 57
Provides information on the features discontinued as of VEOS 4.1.

Discontinuations as of VEOS 4.2 57
Provides information on the features discontinued as of VEOS 4.2.

Discontinuations as of VEOS 4.3 57
Provides information on the features discontinued as of VEOS 4.3.

Discontinuations as of VEOS 4.4 58
Provides information on the features discontinued as of VEOS 4.4.

Discontinuations as of VEOS 4.5 58
Provides information on the features discontinued as of VEOS 4.5.

Discontinuations in VEOS as of dSPACE Release 2021-B..... 59
Provides information on the features discontinued in VEOS as of dSPACE Release 2021-B.

Discontinuations as of VEOS 3.5

Discontinuation of the dSPACE Target for Offline Simulation and migration

Model Interface Package for Simulink The *dSPACE Target for Offline Simulation* was delivered for the last time with dSPACE Release 2015-A. To prepare a Simulink® model for offline simulation with VEOS, use the *Model Interface Package for Simulink*, which was introduced with dSPACE Release 2015-A.

The Model Interface Package for Simulink provides the `dsrt.tlc` system target file, which lets you generate a Simulink implementation container (SIC) file from a Simulink® model. The SIC file is independent of the simulation platform, so that you can integrate the same SIC file in an offline simulation application for VEOS and in a real-time application for SCALEXIO.

New workflow The workflow that the *Model Interface Package for Simulink* provides for preparing a Simulink® model for offline simulation is different from the workflow provided by the *dSPACE Target for Offline Simulation*.

To prepare the offline simulation of a Simulink® model, perform the following steps:

1. In the Simulink® model, specify the `dsrt.tlc` system target file provided by the Model Interface Package for Simulink.
2. Generate code for the Simulink® model.

   Unlike the *dSPACE Target for Offline Simulation*, the *Model Interface Package for Simulink* does not generate an application for a specific simulation platform. Instead, it generates a Simulink implementation container (SIC) file containing Simulink® model code.

   For instructions, refer to Generating Simulink Implementation Containers (Model Interface Package for Simulink - Modeling Guide 🕮).
3. Import the SIC file to the VEOS Player to integrate the model in an offline simulation application for VEOS.

   The VEOS Player builds the SIC file for the VEOS simulation platform.

   For instructions, refer to How to Import Simulink Implementations on page 71.

Discontinuations as of VEOS 4.0

Discontinuation of processor-in-the-loop (PIL) simulation with VEOS and SystemDesk

As of dSPACE Release 2017-A, VEOS and SystemDesk no longer support processor-in-the-loop (PIL) simulation. This includes the generation of V-ECUs for PIL simulation and the simulation of these V-ECUs on evaluation boards.

Discontinuation of the import of CTLGZ 1.0 files

As of VEOS 4.0 (dSPACE Release 2017-A), VEOS no longer supports the import of version 1.0 V-ECU implementation (CTLGZ) files created with products from dSPACE Release 2013-B and earlier.

Discontinuations as of VEOS 4.1

Discontinuation of the import of CTLGZ 2.0 files

As of VEOS 4.1 (dSPACE Release 2017-B), VEOS no longer supports the import of version 2.0 V-ECU implementation (CTLGZ) files.

Discontinuations as of VEOS 4.2

Discontinuation of the XCP buffer size setting

As of VEOS 4.2 (dSPACE Release 2018-A), VEOS no longer supports the XCP buffer size setting.

Up to and including VEOS 4.1, the setting was used in connection with the import of CTLGZ files, and available on the XCP Service page of the VEOS Import dialog.

The relevant `XcpBufferSize` property of the `IImportSettings` automation interface is deprecated.

Discontinuations as of VEOS 4.3

Discontinuation of XCP stimulation

As of VEOS 4.3 (dSPACE Release 2018-B), VEOS no longer supports XCP stimulation.

Up to and including VEOS 4.2, enabling XCP stimulation was possible in connection with the import of CTLGZ files. The relevant setting was available on the XCP Service page of the VEOS Import dialog.

The relevant `XcpStimulationEnabled` property of the `IImportSettings` automation interface is deprecated.

Discontinuations as of VEOS 4.4

Simultaneous access to a V-ECU by two tools

As of VEOS 4.4 (dSPACE Release 2019-A), you can access a V-ECU by only one external tool, such as ControlDesk, at a time. The tool accesses the V-ECU via an XCP port of the XCP service implemented in the V-ECU code.

Up to and including VEOS 4.3, simultaneous access to a V-ECU via two external tools was possible.

The relevant setting is available on the XCP Service page of the VEOS Import dialog. The relevant `XcpPort1` and `XcpPort2` properties of the `IImportSettings` automation interface are deprecated. Use the `XcpPort` property instead.

Related topics

Basics

Discontinuations in VEOS.....55

Discontinuations as of VEOS 4.5

Microsoft Visual C/C++ Compiler versions 11 and 14

As of version 4.5, VEOS no longer supports the following versions of the Microsoft Visual C/C++ Compiler:

- 11 (provided by Microsoft Visual Studio 2012 or Visual Studio 2012 Express)
- 14 (provided by Microsoft Visual Studio 2015 or Visual Studio 2015 Express)

As of version 4.5, VEOS supports version 15 of the Microsoft Visual C/C++ Compiler (provided by Microsoft Visual Studio 2017 or Visual Studio 2017 Express).

Refer to Simulation Target Manager on page 163.

Related topics

Basics

Discontinuations in VEOS.....55

Discontinuations in VEOS as of dSPACE Release 2021-B

DsIdBusIf support

As of dSPACE Release 2021-B, VEOS will no longer support V-ECUs or V-ECU implementations with a *dSPACE Idealized Bus Interface* (DsIdBusIf) module. The DsIdBusIf module is a dSPACE-specific module to specify unsupported bus protocols using an idealized bus without specific arbitration.

Use the *Signal-to-Frame Converter* (Sfc) module instead. The Sfc module is a dSPACE-specific module that lets you connect V-ECUs with modeled bus communication to environment models or other V-ECUs that use signal-based communication, such as SIC and FMU, or buses that are not supported. The bus signals can be transferred.

For more information, refer to Signal-to-Frame Converter (Sfc) (SystemDesk Manual 🕮).

Related topics

Basics

Discontinuations in VEOS..... 55

Integrating the Simulation System

Where to go from here

Information in this section

Basics and Instructions.....62
Reference Information.....116
Automation.....169
Troubleshooting.....176
Limitations.....178

Basics and Instructions

Where to go from here

Information in this section

Introduction to Simulation System Integration.....62
Importing V-ECU Implementations.....65
Importing Simulink Implementation Containers (SICs).....70
Importing Bus Simulation Containers (BSCs).....76
Importing Functional Mock-Up Units (FMUs).....83
Importing System Model Containers (SMCs).....95
Importing VPUs.....97
Connecting VPUs.....100
Exporting System Model Containers.....114

Introduction to Simulation System Integration

Basics on Integrating the Simulation System

Introduction

Integrating the simulation system means importing and building container files, e.g., V-ECU implementations and Simulink implementation containers. The VPUs resulting from these build processes can then be interconnected in a second step.

Importing container files

The VEOS Player lets you import the following container file types to integrate them in a simulation system:

Container File Type	Refer to
V-ECU implementation (VECU) files	Importing V-ECU Implementations on page 65.
Simulink implementation container (SIC) files	Importing Simulink Implementation Containers (SICs) on page 70.
Bus simulation container (BSC) files	Importing Bus Simulation Containers (BSCs) on page 76.

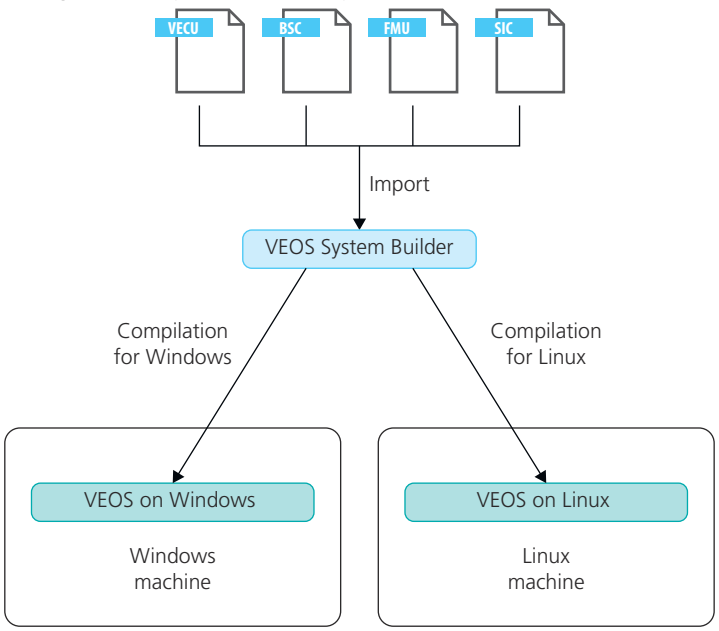
Container File Type	Refer to
Functional Mock-Up Unit (FMU) files	Importing Functional Mock-Up Units (FMUs) on page 83.
System model container (SMC) files	Importing System Model Containers (SMCs) on page 95.

Build process and supported simulation targets

During the container file import, VEOS automatically performs a build process. The build process results in a VPU.

64-bit applications VEOS lets you build 64-bit applications from model implementations, which allows you to integrate model implementations that contain 64-bit libraries in an offline simulation.

Compilation for Windows or Linux To create an offline simulation application for Windows or Linux, you have to select the corresponding compiler during model implementation import.



Supported simulation targets VEOS supports the following simulation targets for the build process:

Simulation Target		Abbreviation	Version(s)
Target	**Compiler**		
HostPC32	GNU GCC C Compiler	HostPC32/GCC	5.2[1)]
HostPC32	Microsoft Visual C/C++ Compiler	HostPC32/MSVC[2)]	14.1x[3)]
HostPC64	GNU GCC C Compiler	HostPC64/GCC[4)]	5.2[1)]

Simulation Target		Abbreviation	Version(s)
Target	**Compiler**		
HostPC64	Microsoft Visual C/C++ Compiler	HostPC64/MSVC[2]	14.1x[3]
HostPCLinux64[5]	GNU GCC C Compiler (cross compiler)	HostPCLinux64/GCC	5.5[1]

[1] Provided by VEOS 5.2.
[2] For limitations in connection with the MSVC compiler, refer to Limitations for Integrating the Simulation System on page 178.
[3] Provided by Microsoft Visual Studio 2017. A version of the Windows 10 SDK is required.
[4] When you import and build a model implementation with the HostPC64/GCC simulation target, the build process might fail. For troubleshooting information, refer to Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target on page 176.
[5] To build an offline simulation application for VEOS on Linux. For more information, refer to How to Build an Offline Simulation Application for Linux with the VEOS Player on page 298.

To specify the simulation target, you have to use the **Simulation Target Manager**.

Build result The VPUs resulting from the build processes are part of the offline simulation application (OSA). The OSA file contains the simulation system's VPUs and the configuration of the simulation.

IP protection aspects

- When you import a model implementation, disable source code debugging to preserve the intellectual property (IP) in the model implementation software. Disabling source code debugging prevents debug symbol files from being added to the generated VPU.
- Unlike a model implementation, such as a V-ECU implementation (VECU), an OSA file does not contain source code files. It is therefore suitable for exchanging offline simulation applications while protecting intellectual property (IP).

  However, the OSA file also contains a variable description file with symbol names for each VPU, which may impair IP protection.

Connecting VPUs and preparing bus communication

After the import and build, you have to connect VPUs. To prepare bus communication for offline simulation, you have to connect communication controllers to communication clusters.

Refer to Connecting VPUs on page 100.

Related topics

Basics

Importing Bus Simulation Containers (BSCs) 76

Importing Functional Mock-Up Units (FMUs) ... 83
Importing Simulink Implementation Containers (SICs) ... 70
Importing System Model Containers (SMCs) ... 95
Importing V-ECU Implementations ... 65
Limitations for Integrating the Simulation System ... 178
Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target ... 176

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player ... 298

References

Simulation Target Manager ... 163

Importing V-ECU Implementations

Where to go from here

Information in this section

Basics on Importing V-ECU Implementations ... 65
VEOS lets you import V-ECU implementation (VECU) files to integrate them in a simulation system.

How to Import V-ECU Implementations ... 67
The VEOS Player lets you import V-ECU implementations into a simulation system.

Basics on Importing V-ECU Implementations

Introduction

VEOS supports the integration and simulation of V-ECU implementations for classic V-ECUs and adaptive V-ECUs.

VECU container file format

As of version 5.1, VEOS supports the *VECU container file format* for V-ECU implementation containers.

The VECU container file format is the successor to the CTLGZ format.

- The VECU file format *supports IP protection between ECU software suppliers and OEMs* because it lets suppliers integrate precompiled object and library

files. Adding precompiled ECU code to a VECU container file also has the advantage that build times in the subsequent workflow are reduced.

- The VECU file format *provides multi-platform support*, i.e., it lets the user integrate simulation code for various simulation platforms in one VECU file, including descriptions of the relevant platform dependencies.

A VECU file is structured in *packages*, each of which can be exchanged individually. This makes incremental VECU file updates possible.

Importing V-ECU implementations (classic V-ECUs)

VECU files for classic V-ECUs A VECU file for classic V-ECUs contains the following kinds of files:

- Code files (H, C)
- Object and library files precompiled for one or more simulation platforms, including descriptions of the relevant platform dependencies
- Configuration files (ARXML)
- Variable description (A2L) files
- The `DsFxp` library, which is a library of auxiliary functions used in TargetLink-generated code, can be added to the VECU file if required. For model-based classic V-ECUs, this is managed by SystemDesk automatically.

Import steps Importing a V-ECU implementation involves two steps:

1. The VEOS System Builder builds the V-ECU implementation to be imported.

   When a V-ECU implementation is built, the following steps are performed:

   - The XCP service is integrated into the VPU. The service lets you measure, calibrate, and stimulate VPU variables using a measurement and calibration tool, such as ControlDesk.
   - The Real-Time Testing service is integrated. Real-Time Testing lets you perform tasks such as stimulating variables of the V-ECU during offline simulation.
   - If a V-ECU implementation contains a Rapid Prototyping Access (RptAccess) module, the *dSPACE Internal Bypassing Service* is integrated into the V-ECU.

     This lets you use the *RTI Bypass Blockset* to dynamically download new models into a V-ECU, for example, to implement an internal bypass, which accesses V-ECU-internal variables and functions.
2. The VPU resulting from the build process is imported to the simulation system.

Support of memory segments and page switching The calibration memory of classic V-ECUs generated with VEOS is divided into two calibration pages. You can use ControlDesk to switch the active V-ECU calibration page during run time.

During the build process, VEOS also generates an ECU Image file that contains the initial parameter values of the V-ECU application. Adding the ECU Image file to the XCP on Ethernet device that represents the V-ECU in a ControlDesk experiment lets you perform offline calibration on the V-ECU with ControlDesk, for example.

For more information on handling calibration pages with ControlDesk, refer to Basics on Memory Pages (ControlDesk Calibration and Data Set Management 🕮) and How to Activate the Working or Reference Page (ControlDesk Calibration and Data Set Management 🕮).

Importing V-ECU implementations (adaptive V-ECUs)

VEOS supports the integration and simulation of V-ECU implementations for adaptive V-ECUs.

The architecture for simulating adaptive V-ECUs requires you to install hypervisor software on your PC. Refer to Basics on Simulating Adaptive V-ECUs on page 236.

Related topics

Basics

Compatibility of VEOS 5.2 40
Simulation Architecture of VEOS 23

HowTos

How to Import V-ECU Implementations 67

References

VEOS Build Console Command Reference 318

How to Import V-ECU Implementations

Objective

The VEOS Player lets you import V-ECU implementations (VECU files) into an offline simulation application.

A V-ECU implementation (VECU) contains parts of the software of an ECU.

Preconditions

- An OSA file must be opened in the VEOS Player.
  Refer to How to Open an OSA File on page 33.
- To integrate and simulate adaptive V-ECUs on VEOS, some preconditions apply.
  Refer to Importing V-ECU implementations (adaptive V-ECUs) on page 67.

Restrictions

Compatibility aspects and possible naming conflicts:

- Keep in mind the compatibility aspects. Refer to Compatibility of VEOS 5.2 on page 40.

- If a model implementation contains custom C code with symbol prefixes and names that cause naming conflicts, you cannot integrate it in an offline simulation. For more information, refer to Avoiding Naming Conflicts when Integrating Custom C Code (SIL Testing Overview 🕮).

Method

To import a V-ECU implementation

1 On the **Home** ribbon, select **Import**.

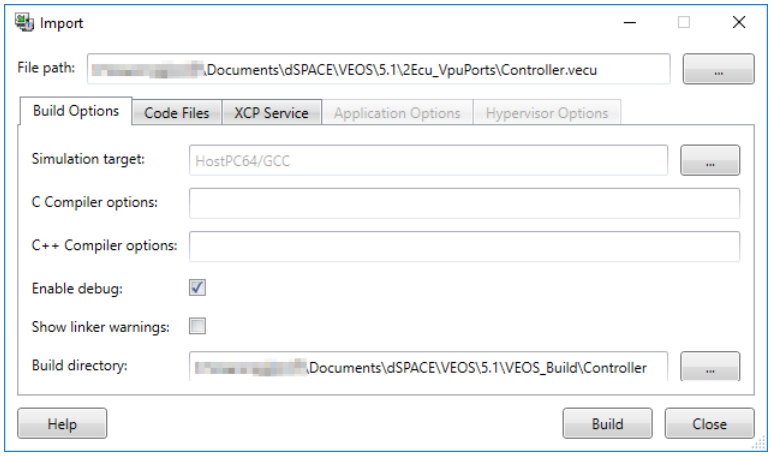
VEOS Player opens the **Select File for Import** dialog.

2 In the **Select File for Import** dialog, select a VECU file and click **Open**.

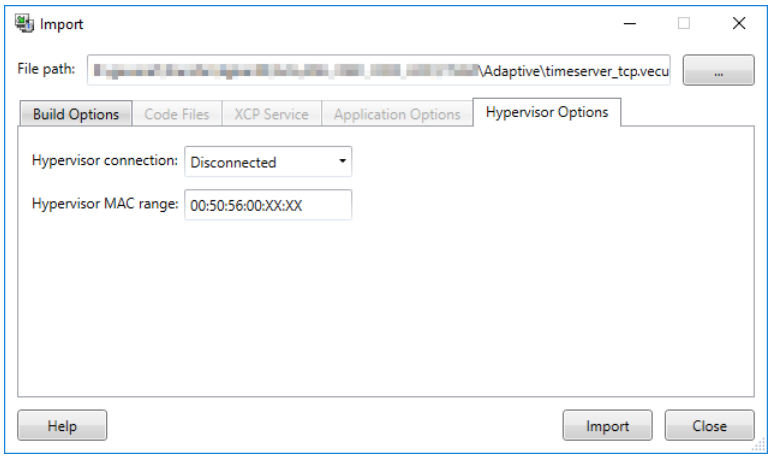
The VEOS Player opens the **Import** dialog.

Depending on whether you select a VECU file of a classic or adaptive V-ECU, the dialog provides different dialog pages:

- When you specify the VECU file for a classic V-ECU, the **Import** dialog opens as shown in the following illustration:



- When you specify the VECU file for an adaptive V-ECU, the **Import** dialog opens as shown in the following illustration:



3 In the **Import** dialog, specify the settings according to your requirements.

4 In the **Import** dialog:
   - Click **Build** to build the V-ECU implementation of the classic V-ECU and import the resulting VPU to the simulation system.

     The **Build log** dialog opens. It displays the processing steps and messages of the build output.
   - Click **Import** to import the V-ECU implementation of the adaptive V-ECU to the simulation system.

     The **Build log** dialog opens. It displays messages of the import.

5 Click **Close** to close the **Build log** dialog.

6 Click to save the modified simulation system.

Result

An imported V-ECU implementation appears in the panes of VEOS Player with one of the following VPU icons:

- (classic V-ECU)
- (adaptive V-ECU)

Next steps

You can now do the following:

- Interconnect the VPUs. Refer to Basics on Signal Connection Dependencies on page 100.
- Connect the communication controllers to communication clusters if required. Refer to Connecting Communication Controllers to Communication Clusters on page 111.

Related topics

Basics

Basics on Importing V-ECU Implementations.....65
Basics on the VEOS Command Line Interface.....315
Compatibility of VEOS 5.2.....40

References

Compiler Options.....166
Import.....152
Simulation Target Manager.....163

Importing Simulink Implementation Containers (SICs)

Where to go from here

Information in this section

Basics on Importing Simulink Implementation Containers (SICs)..... 70
VEOS lets you import Simulink implementation container (SIC) files to integrate them in a simulation system.

How to Import Simulink Implementations..... 71
The VEOS Player lets you import Simulink implementations into a simulation system.

Elements Generated for Simulink Implementation Containers..... 73
Information on the mapping of elements in a Simulink implementation container file to elements in VEOS Player.

Basics on Importing Simulink Implementation Containers (SICs)

Introduction

VEOS lets you import Simulink implementation container (SIC) files to integrate them in a simulation system.

Simulink implementation container (SIC) An implementation of a Simulink® model. It contains the source code files, a variable description and descriptions of the interface to the simulator platform.

A Simulink model implementation container can be generated with the *Model Interface Package for Simulink*. The generation output is an SIC file.

Support of Windows DLLs and Linux SOs VEOS supports SICs that include Windows dynamic link libraries (DLLs) and Linux shared objects (SOs).

Importing and building Simulink implementation containers

Importing a Simulink implementation involves two steps:

1. The VEOS System Builder builds the Simulink implementation to be imported.

   When building a Simulink implementation that supports A2L file generation:

   - The XCP service is integrated into the VPU. The service lets you measure, calibrate, and stimulate VPU variables using a measurement and calibration tool, such as ControlDesk.
   - The Real-Time Testing service is integrated. Real-Time Testing lets you stimulate variables of the environment VPU during offline simulation, for example.

When building a Simulink implementation that supports TRC file generation:

- The DsDAQ measurement service is integrated in the VPU. The service lets you measure and calibrate VPU variables using dSPACE experiment software such as ControlDesk.
- The Real-Time Testing service is integrated. Real-Time Testing lets you, for example, stimulate variables of the environment VPU during offline simulation.

2. The VPU resulting from the build process is imported to the simulation system.

Related topics

Basics

Compatibility of VEOS 5.2 40

HowTos

How to Import Simulink Implementations 71

References

VEOS Build Console Command Reference 318

How to Import Simulink Implementations

Objective

The VEOS Player lets you import Simulink implementation containers (SIC files) into an offline simulation application.

A Simulink implementation container contains code of a Simulink behavior model.

Preconditions

- An OSA file must be opened in the **VEOS Player**.

  Refer to How to Open an OSA File on page 33.
- The SIC file must have been generated for the operating system for which you want to build the offline simuation application.

  Refer to How to Generate a Simulink Implementation Container (Model Interface Package for Simulink - Modeling Guide 🕮).

Restrictions

- There are some limitations for integrating SIC files. Refer to Limitations for Integrating the Simulation System on page 178.
- Compatibility aspects and possible naming conflicts:
  - Keep in mind the compatibility aspects. Refer to Compatibility of VEOS 5.2 on page 40.

- If a model implementation contains custom C code with symbol prefixes and names that cause naming conflicts, you cannot integrate it in an offline simulation. For more information, refer to Avoiding Naming Conflicts when Integrating Custom C Code (SIL Testing Overview 🕮).

Method

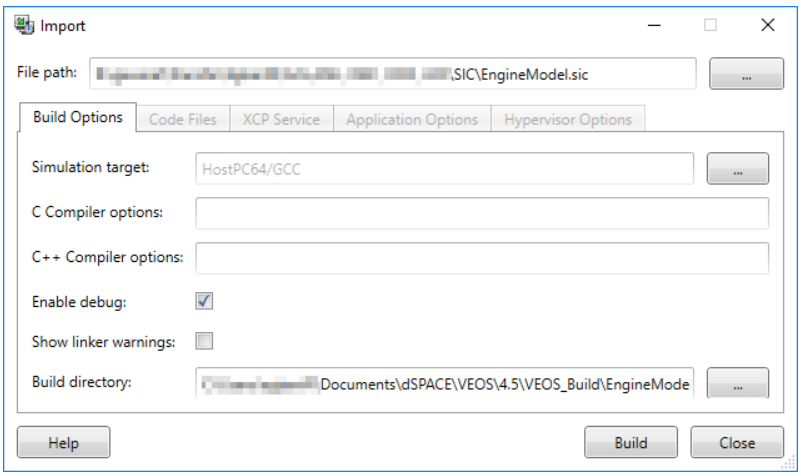
To import a Simulink implementation

1. On the **Home** ribbon, select **Import**.

   VEOS Player opens the **Select File for Import** dialog.

2. In the **Select File for Import** dialog, select an SIC file and click **Open**.

   The **VEOS Player** opens the **Import** dialog. (The following example shows the **Import** dialog for an SIC file that supports TRC file generation.)



3. In the **Import** dialog, specify the settings for building the Simulink implementation, such as compiler options.

4. Click **Build** to build the Simulink implementation and import the resulting VPU to the simulation system.

   The **Build output** dialog opens. It displays the processing steps and messages of the build output.

5. After the build process, click **Close** to close the **Build output** dialog.

6. Click **Close** to close the **Import** dialog.

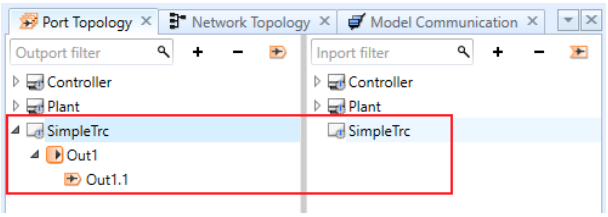
7. Click 💾 to save the modified simulation system.

Result

An imported Simulink implementation container appears in the **Port Topology** pane and the **Network Topology** pane with one of the following VPU icons:

- Environment VPU (imported with a TRC/SDF file)
- Controller VPU (imported with an A2L file)

Outgoing and incoming signals are displayed in the **Port Topology** pane. The illustration below shows an example.



Next steps

You can now connect the signals. Refer to Basics on Signal Connection Dependencies on page 100.

Related topics

Basics

Basics on Importing Simulink Implementation Containers (SICs) 70
Compatibility of VEOS 5.2 40
Elements Generated for Simulink Implementation Containers 73

References

Compiler Options 166
Import 152
Simulation Target Manager 163
VEOS Build Console Command Reference 318

Elements Generated for Simulink Implementation Containers

Element mapping

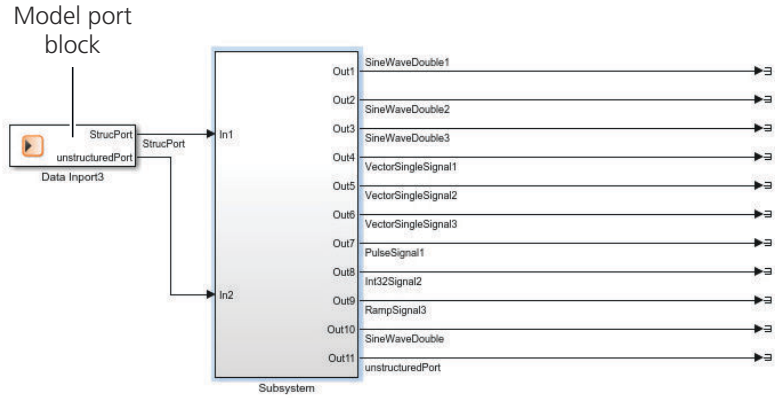
The following table shows the mapping of elements in a Simulink implementation container file to elements in VEOS Player:

Element in the Container File	Element in the VEOS Player	Element Symbol
Model subsystem	Port group	
Model port block	Port	▪ (Port of outgoing signal groups or signals) ▪ (Port of incoming signal groups or signals)
Structured model port/bus	Signal group	▪ (Signal group of outgoing signals) ▪ (Signal group of incoming signals)

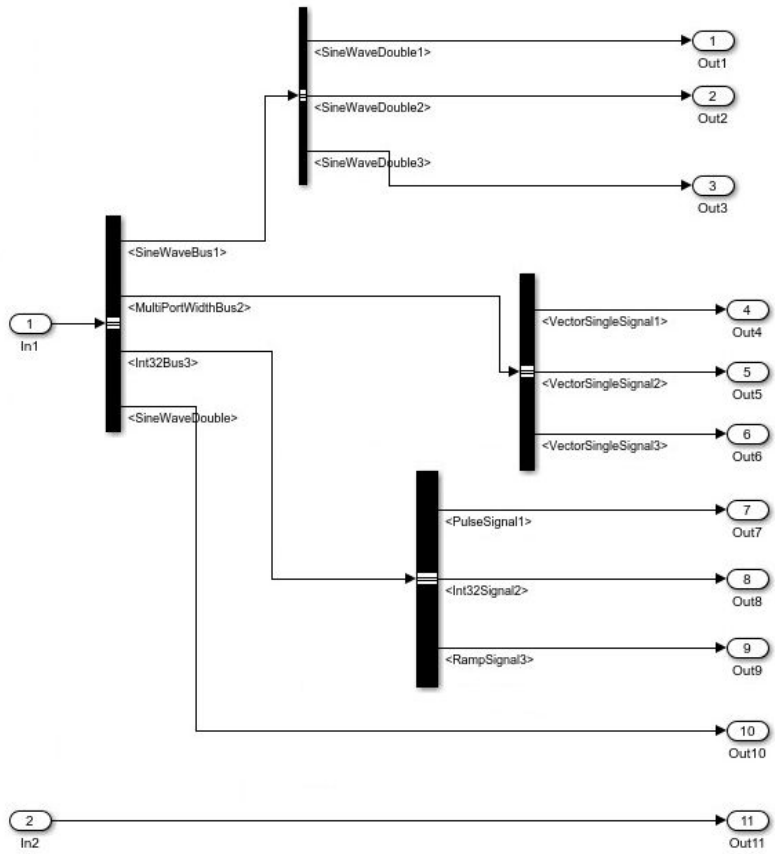
Element in the Container File	Element in the VEOS Player	Element Symbol
Unstructured model port	Signal	▪ (outgoing signal) ▪ (incoming signal)

Example

Simulink model with model port block The following illustration shows an excerpt from a Simulink model with a model port block connected to a subsystem. The model port block has one structured port and one unstructured port.

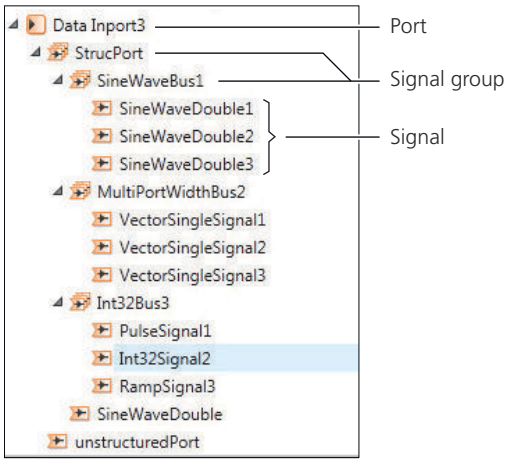


Model port structure The following illustration shows the internal structure of the two ports:



Display in the VEOS Player When you import a Simulink implementation to the VEOS Player, the VEOS Player creates ports for the model port blocks.

The following illustration of the VEOS Player shows the hierarchical structure of the ports of the example model port block shown above.



VEOS Player displays the dimension of a signal via the **Data width** property in the **Properties** pane.

Related topics

Basics

Basics on Importing Simulink Implementation Containers (SICs)..... 70

HowTos

How to Import Simulink Implementations..... 71

Importing Bus Simulation Containers (BSCs)

Where to go from here

Information in this section

Basics on Importing Bus Simulation Containers (BSCs)..... 77
VEOS lets you import bus simulation container (BSC) files to integrate them in a simulation system.

How to Import Bus Simulation Containers..... 78
VEOS Player lets you import bus simulation container (BSC) files into a simulation system.
A bus simulation container contains elements of a communication matrix that are assigned to a bus configuration.

Elements Generated for Bus Simulation Containers.....80
Information on the mapping of elements in a bus simulation container to elements in VEOS Player.

Basics on Importing Bus Simulation Containers (BSCs)

Introduction

VEOS lets you import bus simulation container (BSC) files to integrate them in a simulation system.

Bus simulation container (BSC) A container that contains an implementation of bus communication which is configured with the Bus Manager.

- BSC files can be imported to VEOS Player to implement the configured bus communication in offline simulation application and perform offline simulation on VEOS.
- BSC files can be imported to ConfigurationDesk to implement the configured bus communication in a real-time application and perform real-time simulation on SCALEXIO or a MicroAutoBox III.

Support of Windows DLLs and Linux SOs VEOS supports BSCs that include Windows dynamic link libraries (DLLs) and Linux shared objects (SOs).

Importing and building bus simulation containers

Importing a bus simulation container involves two steps:

1. The VEOS System Builder builds the BSC file to be imported.
   - The DsDAQ measurement service is integrated in the VPU. The service lets you measure and calibrate VPU variables using dSPACE experiment software such as ControlDesk.
   - The Real-Time Testing service is integrated. Real-Time Testing lets you, for example, stimulate variables of the VPU during offline simulation.
2. The VPU resulting from the build process is imported to the simulation system.

Related topics

Basics

Compatibility of VEOS 5.2 40
Elements Generated for Bus Simulation Containers 80

HowTos

How to Import Bus Simulation Containers 78

References

VEOS Build Console Command Reference 318

How to Import Bus Simulation Containers

Objective

VEOS Player lets you import bus simulation container (BSC) files into an offline simulation application.

Preconditions

An OSA file must be open in VEOS Player.

Refer to How to Open an OSA File on page 33.

Restrictions

Observe the following restrictions:

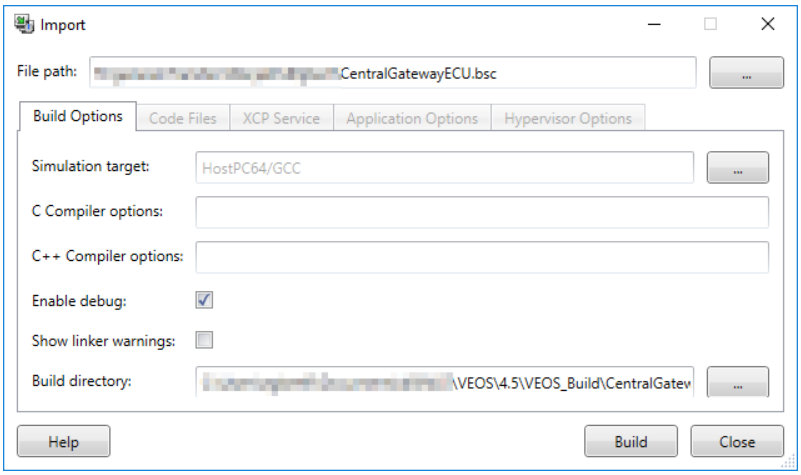
- Keep in mind the compatibility aspects.
  Refer to Compatibility of VEOS 5.2 on page 40.
- There are some limitations for integrating BSC files.
  Refer to Limitations for Integrating the Simulation System on page 178.

Method

To import a bus simulation container

1. On the Home ribbon, select Import.
   VEOS Player opens the Select File for Import dialog.
2. In the Select File for Import dialog, select a BSC file and click Open.

VEOS Player opens the **Import** dialog.



3 In the **Import** dialog, specify the settings for building the BSC, such as compiler options.

4 Click **Build** to build the BSC and import the resulting bus VPU to the simulation system.

The **Build output** dialog opens. It displays the processing steps and messages of the build output.

5 After the build process, click **Close** to close the **Build output** dialog.

6 Click **Close** to close the **Import** dialog.

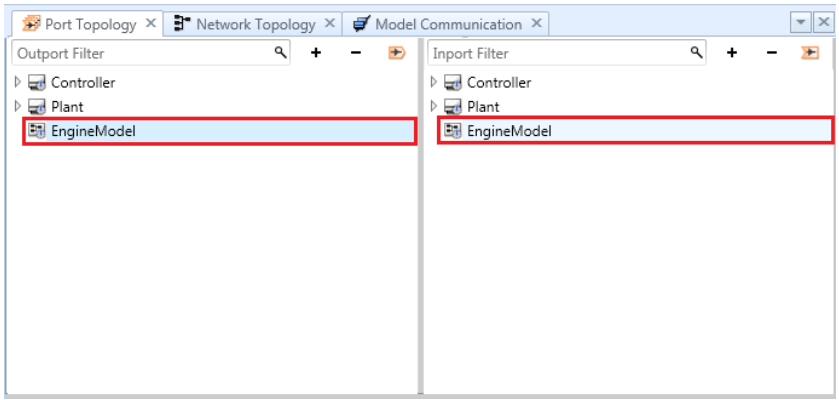
7 Click to save the modified simulation system.

Result

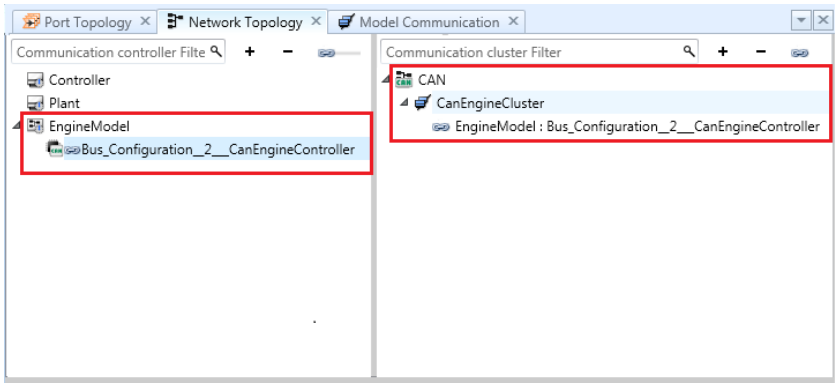
A bus VPU resulting from importing a BSC appears in the **Port Topology** pane and the **Network Topology** pane with the following icon:

The communication controllers and connections to communication clusters appear in the **Network Topology** pane, too.

Example The following illustration shows the VPUs of an imported example BSC in the **Port Topology** pane.



The illustration below shows the connected communication controller and cluster of an imported example BSC in the **Network Topology** pane.



Next steps

You can now connect communication controllers to communication clusters if required. Refer to Connecting Communication Controllers to Communication Clusters on page 111.

Related topics

Basics

Basics on Importing Bus Simulation Containers (BSCs) 77
Elements Generated for Bus Simulation Containers 80

References

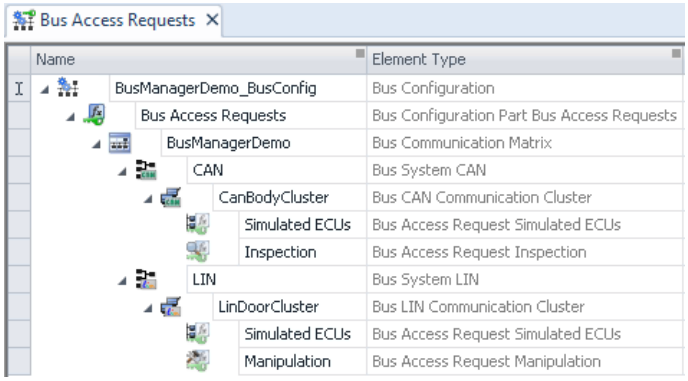
Compiler Options 166
Import 152
Simulation Target Manager 163
VEOS Build Console Command Reference 318

Elements Generated for Bus Simulation Containers

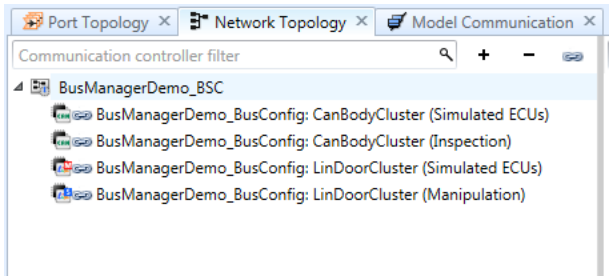
Elements in the Network Topology pane

For each *bus access request* in the imported bus simulation container, the VEOS Player displays a communication controller in the **Network Topology** pane.

The following illustration shows the Bus Manager with some bus access requests as an example:



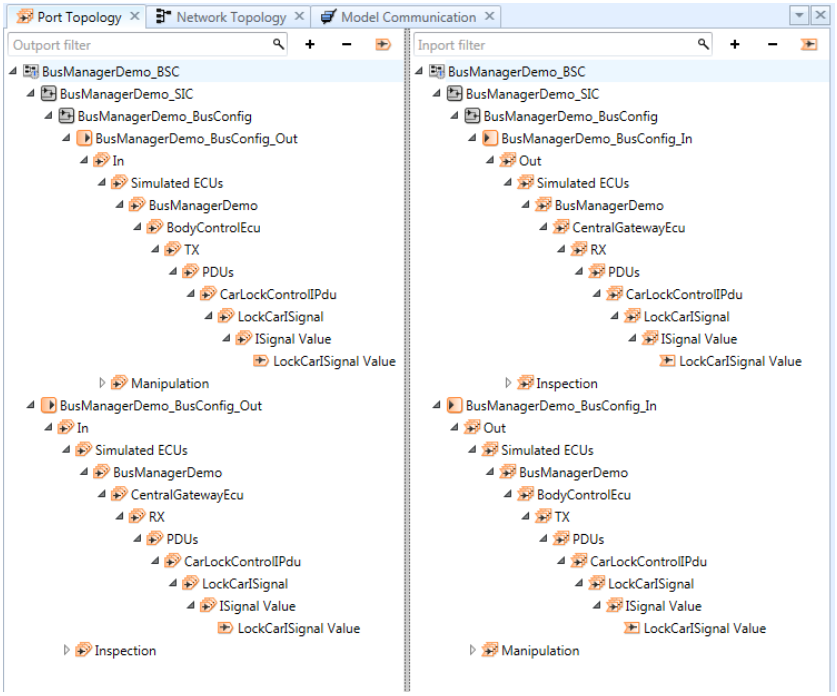
The following illustration shows the resulting communication controllers in the Network Topology pane:



Controller-to-cluster connection In the example above, the VEOS Player automatically connects all the communication controllers to communication clusters during BSC file import. Each connection is represented by a icon. In some cases, automatic connection is not possible, for example, for bus access requests created for frame gateways in the imported bus simulation container. In these cases, you have to connect the related controllers to clusters manually. Refer to Connecting Communication Controllers to Communication Clusters on page 111.

Elements in the Port Topology pane

Depending on the configuration in the Bus Manager, the bus simulation container provides ports that the VEOS Player displays in the Port Topology pane. The following illustration shows an example:



The following table shows the mapping of elements in a bus simulation container to elements in the VEOS Player:

Element in the Container File	Element in the VEOS Player	Element Symbol
Model subsystem	Port group	
Model port block	Port	▪ (Port of outgoing signal groups or signals) ▪ (Port of incoming signal groups or signals)
Structured model port/bus	Signal group	▪ (Signal group of outgoing signals) ▪ (Signal group of incoming signals)
Unstructured model port	Signal	▪ (outgoing signal) ▪ (incoming signal)

For more information on the port types in a bus simulation container that result in signals in the VEOS Player, refer to:

- Bus Manager in ConfigurationDesk: Basics on Bus Simulation Containers (ConfigurationDesk Bus Manager Implementation Guide)
- Bus Manager (stand-alone): Basics on Bus Simulation Containers (Bus Manager (Stand-Alone) Implementation Guide)

Related topics

Basics

Basics on Bus Simulation Containers (ConfigurationDesk Bus Manager Implementation Guide 🕮)
Basics on Bus Simulation Containers (Bus Manager (Stand-Alone) Implementation Guide 🕮)
Basics on Importing Bus Simulation Containers (BSCs)..... 77
Connecting Communication Controllers to Communication Clusters..... 111

HowTos

How to Import Bus Simulation Containers..... 78

Importing Functional Mock-Up Units (FMUs)

Where to go from here

Information in this section

Definition of the FMI Standard and FMUs..... 84
Functional Mock-up Interface (FMI) is a tool-independent standard that supports both the co-simulation and the exchange of dynamic models by using archive files containing a combination of XML files and C functions provided in source and/or binary form. The FMI standard defines the interface to be implemented by a Functional Mock-up Unit (FMU).

Basics on Importing Functional Mock-Up Units (FMUs)..... 85
Functional Mock-up Units (FMUs) are built according to a tool-independent standard and are used for simulation purposes via the Functional Mock-up Interface (FMI).

Specifying the FMU Simulation Behavior..... 87
You can use the standard dSPACE simulation behavior for a VPU that is based on an FMU. As an alternative, you can use the FMI-compliant simulation behavior according to the FMI specification.

How to Import FMUs..... 90
The VEOS Player lets you import FMUs into a simulation system.

Variable Description (TRC) File Entries for FMU Variables..... 92
TRC file entries are created for the variables of an imported FMU.

Definition of the FMI Standard and FMUs

Functional Mock-up Interface

Functional Mock-up Interface (FMI) is a tool-independent standard that supports both the co-simulation and the exchange of dynamic models by using archive files containing a combination of XML files and C functions provided in source and/or binary form. The FMI standard defines the interface to be implemented by a Functional Mock-up Unit (FMU).

FMI for Co-Simulation interface The FMI for Co-Simulation interface is designed both for coupling simulation tools (simulator coupling, tool coupling), and for coupling with subsystem models, which have been exported by their simulators together with its solvers as runnable code.

FMI for Model Exchange interface The FMI for Model Exchange interface defines an interface to the model of a dynamic system described by differential, algebraic and discrete-time equations. Via the FMI for Model Exchange interface, these equations can be evaluated as needed in different simulation environments.

VEOS FMI compatibility For information on VEOS FMI compatibility, refer to Compatibility of VEOS 5.2 on page 40.

Functional Mock-up Unit

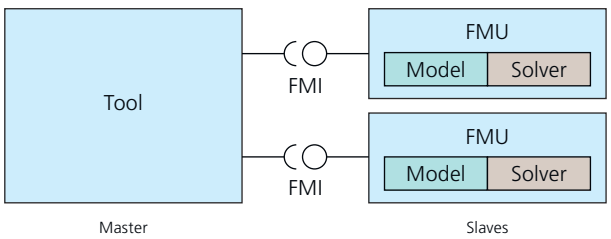
A Functional Mock-up Unit (FMU) implements the Functional Mock-up Interface (FMI). An FMU describes and implements the functionality of a model. It is an archive file with the FMU file name extension. An FMU file contains:

- The functionality defined in C source code files or binary files (e.g., DLL files) or both
- The `modelDescription.xml` file with the description of the interface data
- Additional resources needed for simulation

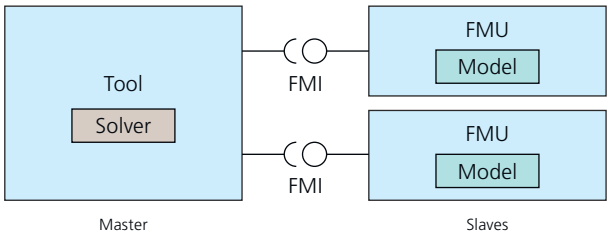
Correlation between the FMI standard and FMUs

The illustrations below show how FMUs correlate with the FMI standard.

FMI for Co-Simulation interface The illustration below shows two FMUs and a tool interconnected via the FMI for Co-Simulation interface.



FMI for Model Exchange interface The illustration below shows two FMUs and a tool interconnected via the FMI for Model Exchange interface.



Note

VEOS supports only the *FMI for Co-Simulation interface*, but not the FMI for Model Exchange interface.

Related topics

Basics

Basics on Importing Functional Mock-Up Units (FMUs) 85

HowTos

How to Import FMUs 90

References

Variable Description (TRC) File Entries for FMU Variables 92

Basics on Importing Functional Mock-Up Units (FMUs)

Introduction

The VEOS Player lets you import Functional Mock-up Units (FMUs). For each imported FMU, the VEOS Player builds a VPU, which you can connect to other VPUs of the simulation system. The connected FMUs and VPUs can then be used for simulation in SIL mode.

Preconditions for using FMUs in VEOS

FMUs you want to import in an offline simulation application must fulfill the following preconditions:

- VEOS supports Functional Mock-up Units (FMUs) that comply with the FMI 2.0 standard.
- VEOS supports only the *FMI for Co-Simulation interface*, but not the FMI for Model Exchange interface.

You can integrate an FMU file in a VEOS simulation on Windows and in a VEOS simulation on Linux.

For detailed and up-to-date compatibility information on dSPACE FMI support, refer to:

http://www.dspace.com/go/FMI-Compatibility.

- FMUs must provide their C functions as source files or as a DLL file compiled for Windows.

  If the FMU to be imported contains C source code files and a DLL compiled for Windows, VEOS Player uses the latter for the build process.
- FMUs must not require additional tools for simulation.
- FMUs must define a default start time equal to 0.0.

Importing and building FMUs

Importing and building FMUs in a single step The VEOS Player lets you import FMUs. During import, the VEOS Player invokes the build process.

The VEOS System Builder builds the FMU to be imported.

When an FMU is built:

- The DsDAQ measurement service is integrated in the VPU. The service lets you measure and calibrate VPU variables using dSPACE experiment software such as ControlDesk.
- The Real-Time Testing service is integrated. Real-Time Testing lets you, for example, stimulate variables of the environment VPU during offline simulation.

For instructions on importing and building FMUs, refer to How to Import FMUs on page 90.

Note

There is a limitation for building FMUs with the MSVC compiler. Refer to Limitations for Integrating the Simulation System on page 178.

Logging of FMU-internal messages You can enable logging of FMU-internal messages during import. You have to specify a specific compiler option for this. For more information, refer to Compiler Options on page 166.

FMU represented as an environment VPU VEOS represents FMUs as environment VPUs. This means, FMUs are displayed with the same icon as an environment VPU in the VEOS Player.

SDF/TRC files are generated. For information on TRC file contents, refer to Variable Description (TRC) File Entries for FMU Variables on page 92.

Signals for FMU inputs and outputs VEOS creates a signal for each input and output whose data type is supported. The following data types are supported:

- Real
- Integer
- Boolean
- Enumeration

Connecting FMUs

After you import an FMU, you can connect the FMU signals to other signals in the simulation system.

For instructions, refer to Basics on Signal Connection Dependencies on page 100.

Related topics

Basics

Compatibility of VEOS 5.2 40
Definition of the FMI Standard and FMUs 84
Specifying the FMU Simulation Behavior 87

HowTos

How to Import FMUs 90

References

Variable Description (TRC) File Entries for FMU Variables 92
VEOS Build Console Command Reference 318

Specifying the FMU Simulation Behavior

Introduction

You can use the *standard dSPACE simulation behavior* for a VPU that is based on an FMU. As an alternative, you can use the *FMI-compliant simulation behavior according to the FMI specification*.

Standard dSPACE simulation behavior

When the computation of, for example, an SIC-based VPU is triggered at t_i, the simulation result is also assigned to t_i. This the *standard dSPACE simulation behavior* since it is used for all the dSPACE-specific model implementation formats (BSC, SIC, and VECU).

Hence, when an FMU is sumulated according to the standard dSPACE simulation behavior, and when the `fmi2DoStep` function of that FMU is called at communication point t_i, the FMU output is considered to be the result for the same communication point t_i.

Note

Using the standard dSPACE simulation behavior, the FMU is not simulated according to the FMI specification.
However, VEOS uses the standard dSPACE simulation behavior for FMUs *by default* for consistency with the simulation behavior of the dSPACE-specific model implementation formats.
As an alternative, you can enable the FMI-compliant simulation behavior.

FMI-compliant simulation behavior

According to the FMI specification, the `fmi2DoStep` function, which is called when an FMU computation step is triggered, performs the following:

1. The function samples the inputs from the communication point t_i.
2. The function performs an integration to the next communication point t_{i+1}.
3. It provides the FMU output for the communication point t_{i+1}.

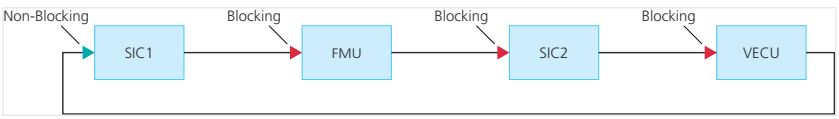
Enabling the FMI-compliant simulation behavior You can enable this behavior by using the `DSFMUComplianceMode` compiler option.

For more information, refer to Enabling FMI-compliant simulation behavior on page 167.

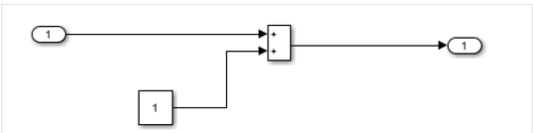
Example

The following example shows the differences between the two simulation behaviors supported for FMU simulation.

Example simulation system The example simulation system consists of the following four interconnected VPUs:



Example model The following illustration shows a schematic of the model that is implemented in each of the four VPUs. The VPU input is incremented by 1 in each sample step.



Simulation results with the standard dSPACE simulation behavior (default simulation behavior) When you use the standard dSPACE simulation behavior for the FMU, the `fmi2DoStep` function that is called with t_i as the current simulation time calculates the FMU outputs for a communication point t_i by using the inputs from the same communication point.

For example, at t_1 = 0.01 s the `fmi2DoStep` function uses the output of SIC1 at t_1 as the input.

Time [s]	VPU Output			
	SIC1	**FMU**	**SIC2**	**VECU**
t_0 = 0	1	2	3	4
t_1 = 0.01	5	6	7	8
t_2 = 0.02	9	10	11	12
t_3 = 0.03	13	14	15	16
t_4 = 0.04	17	18	19	20

Simulation results with the FMI-compliant simulation behavior When you use the FMI-compliant simulation behavior for the FMU, the `fmi2DoStep` function that is called with t_{i-1} as the current simulation time calculates the FMU outputs for a communication point t_i by using the inputs from the previous communication point t_{i-1}.

For example, at t_1 = 0.01 s the `fmi2DoStep` function uses the output of SIC1 at t_0 as the input.

Time [s]	VPU Output			
	SIC1	**FMU**	**SIC2**	**VECU**
t_0 = 0	1	0[1)]	1	2
t_1 = 0.01	3	2	3	4
t_2 = 0.02	5	4	5	6
t_3 = 0.03	7	6	7	8
t_4 = 0.04	9	8	9	10

[1)] At communication point t_0 = 0, the `fmi2DoStep` function is not executed.

Related topics

Basics

Basics on Importing Functional Mock-Up Units (FMUs) 85
Timing in Offline Simulation with VEOS 24

HowTos

How to Import FMUs 90

References

Compiler Options 166

How to Import FMUs

Objective

The VEOS Player lets you import Functional Mock-up Units (FMU files) into an offline simulation application.

Preconditions

An OSA file must be opened in the VEOS Player.

Refer to How to Open an OSA File on page 33.

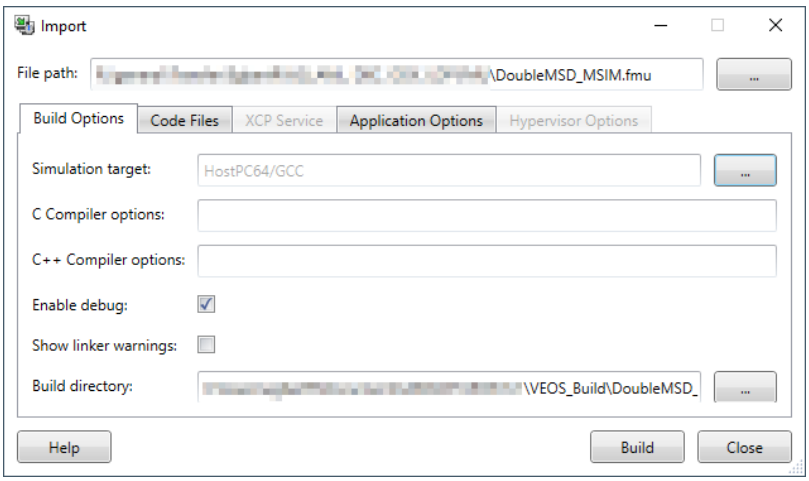
Restrictions

- There are some limitations for integrating FMU files. Refer to Limitations for Integrating the Simulation System on page 178.
- Compatibility aspects and possible naming conflicts:
  - Keep in mind the compatibility aspects. Refer to Compatibility of VEOS 5.2 on page 40.
  - If a model implementation contains custom C code with symbol prefixes and names that cause naming conflicts, you cannot integrate it in an offline simulation. For more information, refer to Avoiding Naming Conflicts when Integrating Custom C Code (SIL Testing Overview 🕮).

Method

To import FMUs

1. On the Home ribbon, select Import.
   VEOS Player opens the Select File for Import dialog.
2. In the Select File for Import dialog, select an FMU file and click Open.
   The VEOS Player opens the Import dialog.



3. In the Import dialog, specify the settings for building the FMU such as compiler options.
4. Click Build to build the FMU and import the resulting VPU to the simulation system.

The **Build output** dialog opens. It displays the processing steps and messages of the build output.

If the FMU to be imported contains C source code files and a DLL compiled for Windows, VEOS Player uses the latter for the build process.

5 After the build process, click **Close** to close the **Build output** dialog.

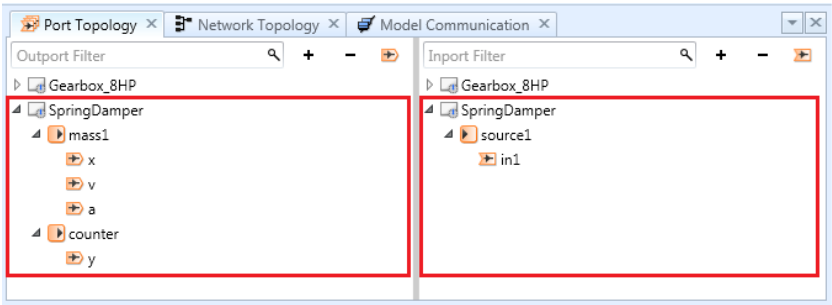
6 Click **Close** to close the **Import** dialog.

7 Click to save the modified simulation system.

Result

The imported FMUs including the outgoing and incoming ports appear in the **Port Topology** pane as an environment VPU with the following icon:

The illustration below shows an example.



Next steps

You can now connect the signals. Refer to Basics on Signal Connection Dependencies on page 100.

Related topics

Basics

Basics on Importing Functional Mock-Up Units (FMUs) 85
Compatibility of VEOS 5.2 40
Definition of the FMI Standard and FMUs 84
Specifying the FMU Simulation Behavior 87

References

Compiler Options 166
Import 152
Simulation Target Manager 163
Variable Description (TRC) File Entries for FMU Variables 92
VEOS Build Console Command Reference 318

Variable Description (TRC) File Entries for FMU Variables

Introduction

TRC file entries are created for the variables of an imported FMU.

FMU variable entries in variable description files

FMU variables for which TRC file entries are generated The following list gives an overview on the FMU variable types (possible combination of variability and causality) for which TRC file entries are generated during the build process:

- Fixed parameter

> **Note**
>
> According to the Functional Mock-up Interface (FMI) standard, changing the value of a fixed parameter via the corresponding TRC variable does not immediately affect the simulation results. The effect occurs when the simulation state changes from `STOP` to `RUN`.
>
> The simulation state is `STOP` in the following cases:
>
> - Directly after the offline simulation application is downloaded to VEOS
> - When you stop the simulation
>
> The simulation state changes to `RUN` when you start the simulation.

- Tunable parameter
- Fixed dependent parameter
- Tunable dependent parameter
- Discrete input
- Continuous input
- Constant output
- Discrete output
- Continuous output
- Constant local
- Fixed local
- Tunable local
- Discrete local
- Continuous local
- Continuous independent

Generated TRC file entries For each supported variable, a TRC file entry is generated like this:

```
<symbol>
{
  type:     <dataType>
  alias:    "<variableName>"
  flags:    <flags>
  value:    <initialValue>     // optional, only for tunable parameters
  range:   < <min>; <max> >    // optional, only if min and max value are defined
  desc:     "<variability> <causality>: <variableDesc>"
  addr:     <address>
}
```

The table below describes the TRC file entry generated for an FMU variable:

Variable	Description
`<symbol>`	Symbol name of global variable
`<dataType>`	Data type in TRC file syntax for variables of type: ▪ Double: flt(64,IEEE) ▪ Integer: int(32) ▪ Boolean: platform-dependent (e.g.,uint(8) for SCALEXIO, int(32) for VEOS)
`<variableName>`	▪ If `variableNamingConvention="flat"`, it is the name property of the corresponding `ScalarVariable` element in the `modelDescription.xml` file. ▪ If `variableNamingConvention="structured"`, it is the last part of the name property. Double quotes are replaced by single quotes.
`<flags>`	Depend on variability and causality of the variable. ▪ For fixed parameters: PARAM|INITONLY ▪ For tunable parameters: PARAM ▪ For fixed dependent parameters and tunable dependent parameters: PARAM|READONLY ▪ For all other variables: OUTPUT|READONLY
`<initialValue>`	Start property of the related `ScalarVariable` element in the `modelDescription.xml` file. Keyword value is only generated for variables of type `tunable parameter`. If the start value is not provided by the FMU, `initialValue` is 0 (or 0.0, depending on the data type).
`<min>` or `<max>`	Range property of the related `ScalarVariable` element in the `modelDescription.xml` file. A keyword range is generated if `<min>` and `<max>` values are provided.
`<variability>`	Variability property of the related `ScalarVariable` element in the `modelDescription.xml` file.
`<causality>`	Causality property of the related `ScalarVariable` element in the `modelDescription.xml` file.
`<variableDesc>`	Description property of the related `ScalarVariable` element in the `modelDescription.xml` file. Double quotes are replaced by single quotes. New lines are replaced by `\n`.
`<address>`	Memory base address of the variable.

TRC file entries for a hierarchical FMU variable structure

If an FMU provides its variables with `variableNamingConvention="structured"` (i.e., hierarchy information is provided for the variables), the generated TRC file hierarchy reflects this structure.

The last hierarchy level is used as variable name and one variable entry is generated as described in FMU variable entries in variable description files.

For all the other hierarchy levels, one TRC group is generated.

Example Suppose the `modelDescription.xml` file of an FMU contains the following variables:

```
Main.DoubleMSD.Constant.k                // parameter
Main.DoubleMSD.Gain1.k                   // parameter
Main.DoubleMSD.Subsystem.Int2.k          // parameter
Main.DoubleMSD.Subsystem.Int2.u(t)       // local
DoubleMSD_r(2)                           // local
der(displacementMassA_MSIM)              // local
displacementMassB_MSIM                   // output
force_MSIM                               // input
```

This results in the following TRC file entries:

```
group “Model Root”
  group “Main”
    group “DoubleMSD”
      group “Constant”
        variable “k”
      group “Gain1”
        variable “k”
      group “Subsystem”
        group “Int2”
          variable “k”
          variable “u(t)”
  variable “displacementMassB_MSIM”
  variable “force_MSIM”
  variable “DoubleMSD_r(2)”
  variable “der(displacementMassA_MSIM)”
```

TRC file entries for a flat FMU variable structure

If an FMU provides its variables with `variableNamingConvention="flat"` (i.e., no hierarchy information is available for the variables), a flat TRC hierarchy with one variable entry is generated as described in FMU variable entries in variable description files:

```
group “Model Root”
  group “Outputs”     // contains all variables with causality “output”
  group “Inputs”      // contains all variables with causality “input”
  group “Parameters”  // contains all variables with causality “parameter” or “calculatedParameter”
  group “Locals”      // contains all variables with causality “local” and “independent”
```

Example Suppose the `modelDescription.xml` file of an FMU contains the following variables:

```
Main.DoubleMSD.Constant.k                // parameter
Main.DoubleMSD.Gain1.k                   // parameter
Main.DoubleMSD.Subsystem.Int2.k          // parameter
Main.DoubleMSD.Subsystem.Int2.u(t)       // local
DoubleMSD_r(2)                           // local
der(displacementMassA_MSIM)              // local
displacementMassB_MSIM                   // output
force_MSIM                               // input
```

This results in the following TRC file entries:

```
group “Model Root”
  group “Outputs”
    variable “displacementMassB_MSIM”
  group “Inputs”
    variable “force_MSIM”
  group “Parameters”
    variable “Main.DoubleMSD.Constant.k”
    variable “Main.DoubleMSD.Gain1.k”
    variable “Main.DoubleMSD.Subsystem.Int2.k”
  group “Locals”
    variable “Main.DoubleMSD.Subsystem.Int2.u(t)”
    variable “DoubleMSD_r(2)”
    variable “der(displacementMassA_MSIM)”
```

Related topics

Basics

Basics on Importing Functional Mock-Up Units (FMUs)..... 85
Definition of the FMI Standard and FMUs..... 84

HowTos

How to Import FMUs..... 90

Importing System Model Containers (SMCs)

How to Import System Model Containers

Objective

VEOS Player lets you import system model containers (SMC files) into an empty offline simulation application.

Basics

SMC file Abbreviation of system model container (SMC) file.

A file archive that contains an arbitrary number of the following containers, which belong to a specific system model:

- Bus simulation container (BSC)
- Functional Mock-up Unit (FMU)
- Simulink implementation container (SIC)
- V-ECU implementation (VECU)

An SMC file additionally contains the following information:

- Description of the data flow connections between the components of the system model

- Description of the bus communication connections between the components of the system model

An SMC file represents a system model container (SMC).

Compiler for building a system model container In contrast to importing other model implementation file types, you cannot specify the compiler during SMC file import.

To build a system model container, VEOS uses one of the following compilers when you import an SMC file:

- The compiler specified in the system model container
- The default compiler as specified in the **Simulation Target Manager** if no compiler is specified in the system model container

For more information, refer to Simulation Target Manager on page 163.

Preconditions

An empty OSA file must be opened in the **VEOS Player**.

Refer to New on page 159.

Restrictions

Compatibility aspects and possible naming conflicts:

- Keep in mind the compatibility aspects. Refer to Compatibility of VEOS 5.2 on page 40.
- If a model implementation contains custom C code with symbol prefixes and names that cause naming conflicts, you cannot integrate it in an offline simulation. For more information, refer to Avoiding Naming Conflicts when Integrating Custom C Code (SIL Testing Overview).

Method

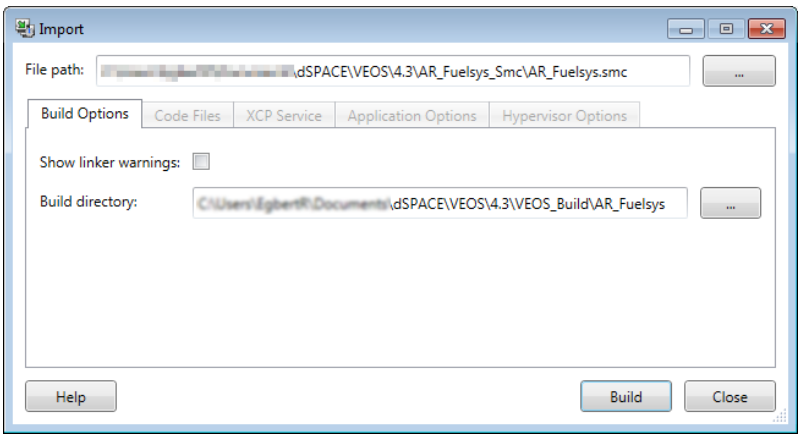
To import a system model container

1. On the **Home** ribbon, select **Import**.

   VEOS Player opens the **Select File for Import** dialog.

2. In the **Select File for Import** dialog, select an SMC file and click **Open**.

   The **VEOS Player** opens the **Import** dialog.



3 In the **Import** dialog, specify the settings for building the system model container, such as the build directory.

4 Click **Build** to build the system model container and import the resulting elements to the simulation system.

The **Build output** dialog opens. It displays the processing steps and messages of the build output.

5 After the build process, click **Close** to close the **Build output** dialog.

6 Click **Close** to close the **Import** dialog.

7 Click to save the modified simulation system.

Result

The imported system model container appears in the following panes:

- **Port Topology** pane
- **Network Topology** pane
- **Model Communication** pane.

Next steps

You can now perform the next steps as described in the following chapters:

- How to Select a Remote Simulator on page 186
- How to Load and Run an Offline Simulation Application on page 188

Related topics

Basics

Compatibility of VEOS 5.2 40

References

Import 152
Simulation Target Manager 163

Importing VPUs

How to Import VPUs

Objective

The VEOS Player lets you import VPUs into a simulation system.

Basics

The VEOS Player lets you import VPUs (OSA files) into an offline simulation application.

When you import one or more container files, e.g., V-ECU implementations and Simulink implementation containers, or VPUs that are already in the simulation system, the VEOS Player lets you *update the VPUs* while keeping existing signal connections if possible.

Preconditions

An OSA file must be opened in the VEOS Player.

Refer to How to Open an OSA File on page 33.

Restrictions

Compatibility aspects and possible naming conflicts:

- Keep in mind the compatibility aspects. Refer to Compatibility of VEOS 5.2 on page 40.
- If a model implementation contains custom C code with symbol prefixes and names that cause naming conflicts, you cannot integrate it in an offline simulation. For more information, refer to Avoiding Naming Conflicts when Integrating Custom C Code (SIL Testing Overview 🕮).

Method

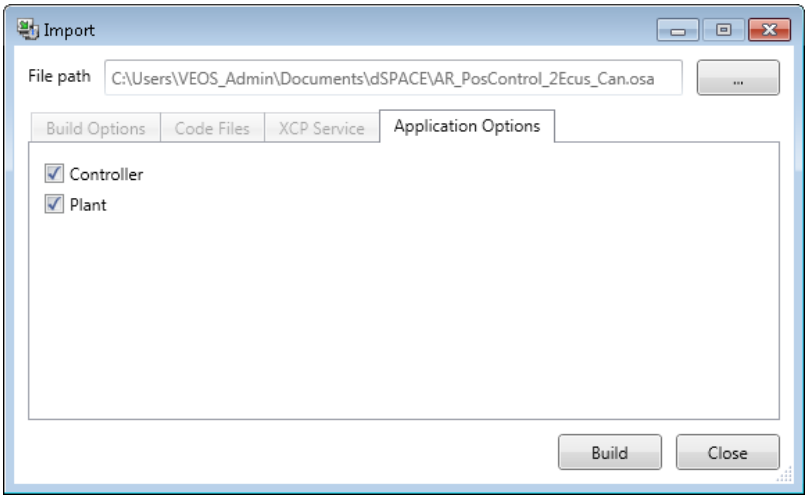
To import VPUs

1. On the **Home** ribbon, select **Import**.

   VEOS Player opens the **Select File for Import** dialog.
2. In the **Select File for Import** dialog, select an OSA file and click **Open**.

   The VEOS Player opens the **Import** dialog.

   All the VPUs of the specified OSA file are displayed in the dialog.
3. In the **Import** dialog:
   - Select the VPU(s) to be imported to the simulation system.
   - If the VPUs to be imported are connected to buses, specify whether to import bus information.



4 In the **Import** dialog, click **Build** to close the dialog and import the selected VPU(s) to the simulation system.

5 Click to save the modified simulation system.

Result

Imported VPUs appear in the **Port Topology** pane as V-ECUs with the following icon:

If the following items exist, they also appear in the panes:

- Outgoing and incoming signals of an imported VPU appear in the **Port Topology** pane.
- Communication controllers, communication clusters and their connections of an imported VPU appear in the **Network Topology** pane.

Next steps

You can now connect the signals. Refer to Basics on Signal Connection Dependencies on page 100.

Related topics

Basics

Basics on Signal Connection Dependencies..... 100
Compatibility of VEOS 5.2..... 40

References

Import..... 152
Replace..... 161
Update..... 166

Connecting VPUs

Where to go from here

Information in this section

Basics on Signal Connection Dependencies..... 100
During offline simulation, data exchange between VPUs is performed via interconnected signals. The VEOS Player lets you establish these connections.

Ways to Connect VPU Signals..... 102
The VEOS Player supports different ways to connect VPUs.

Examples for Connecting VPUs by Dragging Structural Elements..... 104
Possible approaches of connecting signals in an example offline simulation application

Specifying Blocking or Non-Blocking Communication..... 109
VEOS supports blocking and non-blocking communication between VPUs.

Connecting Communication Controllers to Communication Clusters..... 111
The VEOS Player lets you connect communication controllers to communication clusters.

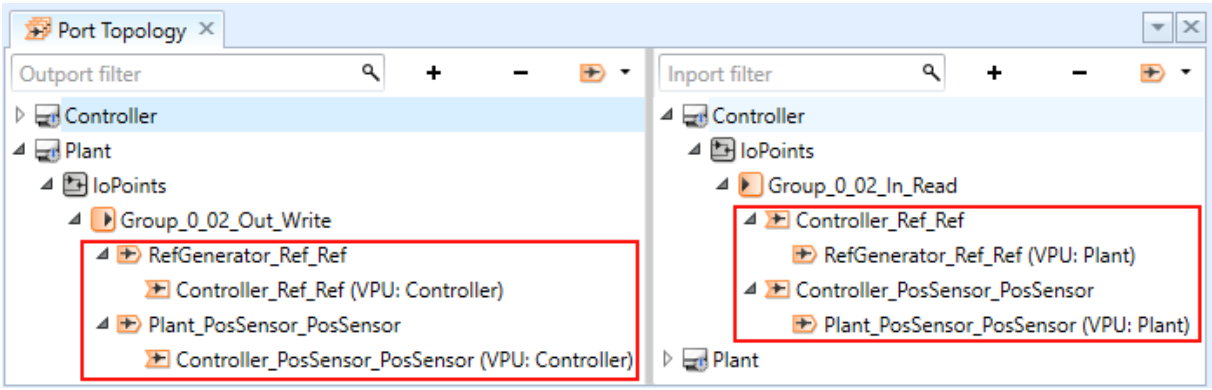
Basics on Signal Connection Dependencies

Introduction

During offline simulation, data exchange between VPUs is performed via interconnected *signals*. The VEOS Player lets you establish these connections.

The **Port Topology** pane displays a connection as a child of a VPU signal. The signal connection displays the name of the connected VPU in brackets.

The following illustration shows an example of two outport signals of the Plant VPU connected to two inport signals of the Controller VPU.



Signal connection dependencies

When you connect signals, you have to consider some signal connection dependencies.

The following table gives an overview of the possible signal connections depending on the signal type and on the signal direction:

Outport Signal Type	Inport Signal	Preconditions	Remarks
Fixed-size signal	Fixed-size signal	–	▪ Different data widths[1)] ▪ Casting in case of different data types[2)]
	Variable-size signal	Data width precondition[3)]	▪ Data width of inport signal[4)] ▪ Casting in case of different data types[2)]
	Binary signal	▪ Data width precondition[3)] ▪ Outport data type precondition[5)]	Data width of inport signal[4)]
Variable-size signal	Variable-size signal	Data width precondition[3)]	▪ Data width of inport signal[4)] ▪ Casting in case of different data types[2)]
	Binary signal	▪ Data width precondition[3)] ▪ Outport data type precondition[5)]	–
Binary signal	Variable-size signal	▪ Data width precondition[3)] ▪ Inport data type precondition[6)]	Data width of inport signal[4)]
	Binary signal	–	–

[1)] If the data width of the outport signal is greater than the data width of the inport signal, not all the array elements of the outport signal are transmitted. This leads to incomplete data transmission.

[2)] If the data type of the outport and inport signals differ, VEOS performs type casting individually for each array element of the signal. This can lead to inexact data transmission.

Outport Signal Type	Inport Signal	Preconditions	Remarks

[3] The data width of the outport signal must be smaller than or equal to the data width of the inport signal.
[4] The data width of the inport signal is set to the data width of the outport signal.
[5] The data type of the outport signal must be uint8 or int8.
[6] The data type of the inport signal must be uint8 or int8.

Signals and sampling intervals

If you use blocking communication, make sure that the sampling interval of an incoming signal is an integer multiple of the sampling interval of the connected outgoing signal. Otherwise, there might be communication issues.

Refer to Specifying Blocking or Non-Blocking Communication on page 109.

Related topics

Basics

Specifying Blocking or Non-Blocking Communication..... 109
Ways to Connect VPU Signals..... 102

References

Disconnect all Child Connections..... 143
Signal..... 129

Ways to Connect VPU Signals

Introduction

The VEOS Player supports different ways to connect VPUs.

Auto-connecting VPUs

The VEOS Player lets you automatically connect VPUs by using the Autoconnect command. The VEOS Player automatically connects all the signals of a VPU with the same name in all the structural elements of the offline simulation application.

The VEOS Player cannot connect VPUs if the VPU with output signals contains a structure with two or more signals with the same name as the VPU with input signals, i.e., if the signal name is not unique.

However, if the signal name is not unique on the VPU with output signals, the VEOS Player connects the input signal to the output signal whose structural elements and their names are identical to those of the input signal.

Refer to Autoconnect on page 138.

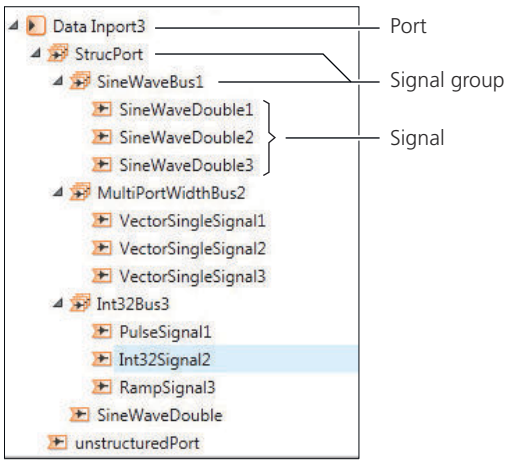
Connecting VPUs by dragging structural elements

Hierarchical display of signals The VEOS Player displays the signals of a VPU hierarchically, i.e., together with following possible structural elements:

- Port group
- Port
- Signal group

These structural elements result from elements of the underlying model.

The following illustration shows the hierarchy of signals of an offline simulation application as an example.



Dragging structural elements You can connect signals by dragging one structural element to another. The VEOS Player then automatically connects all the signals of the first structure to signals of the second structure if the signals have the same name.

The VEOS Player cannot connect VPUs if the VPU with output signals contains a structure with two or more signals with the same name as the VPU with input signals, i.e., if the signal name is not unique.

However, if the signal name is not unique on the VPU with output signals, the VEOS Player connects the input signal to the output signal whose structural elements and their names are identical to those of the input signal.

Possible combinations of draggable elements When you drag an element to another, the VEOS Player indicates whether drag & drop is possible.

Drag & Drop Possible	Drag & Drop Impossible
Port: Plant.Group_0_02_Out_Write Port: Plant.Group_0_02_In_Read	Port: Plant.Group_0_02_Out_Write -- Cannot drop here. --

The following table shows the combinations for which drag & drop is possible:

Element in One Tree View	Element in the Other Tree View
VPU[1)]	VPU
Port group[1)] ( )	Port group ( )
Port[1)] (,)	Port (,)

Element in One Tree View	Element in the Other Tree View
Signal group[1] (,)	Signal group (,)
Signal[1] (,)	Signal[1] (,)

[1] You can drag one outgoing element to several incoming elements. Dragging n outgoing elements to one or m incoming elements is not supported.

For examples, refer to Examples for Connecting VPUs by Dragging Structural Elements on page 104.

Connecting signals by dragging them individually

You can connect individual signals by dragging one to another, independent of the signal names.

You can connect one outgoing signal to multiple incoming signals. n:1 or n:m connections are not supported.

Connecting via the automation interface

You can connect VPU signals by using the VEOS Player automation interface. Refer to Automating the Connection of VPUs on page 172.

Related topics

Basics

Automating the Connection of VPUs..... 172
Basics on Signal Connection Dependencies..... 100

Examples

Examples for Connecting VPUs by Dragging Structural Elements..... 104

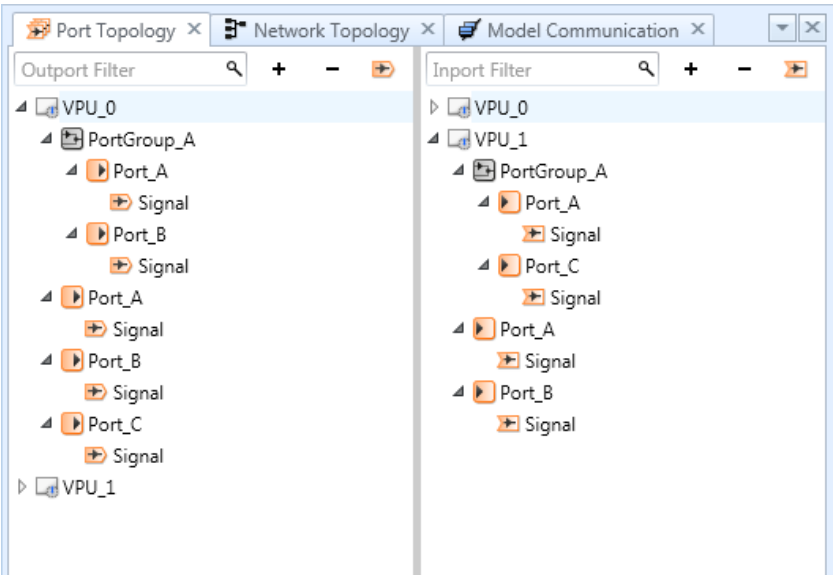
References

Autoconnect..... 138
Signal..... 129

Examples for Connecting VPUs by Dragging Structural Elements

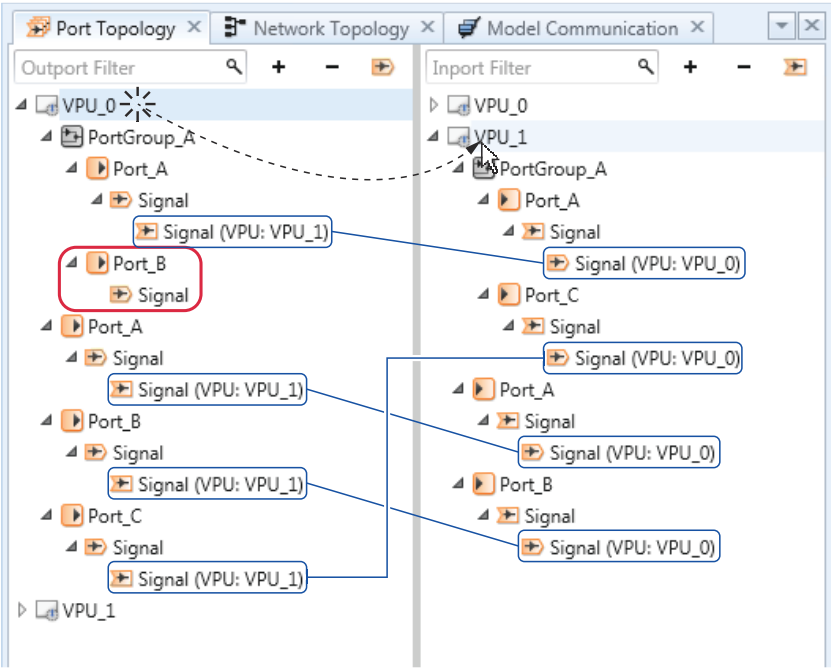
Example OSA

The following illustration shows an example offline simulation application (OSA). This example OSA is used to demonstrate possible approaches to connecting signals by dragging structural elements:



Dragging a VPU

You can drag a VPU to another to connect their signals. The following illustration shows how this works for the example OSA:



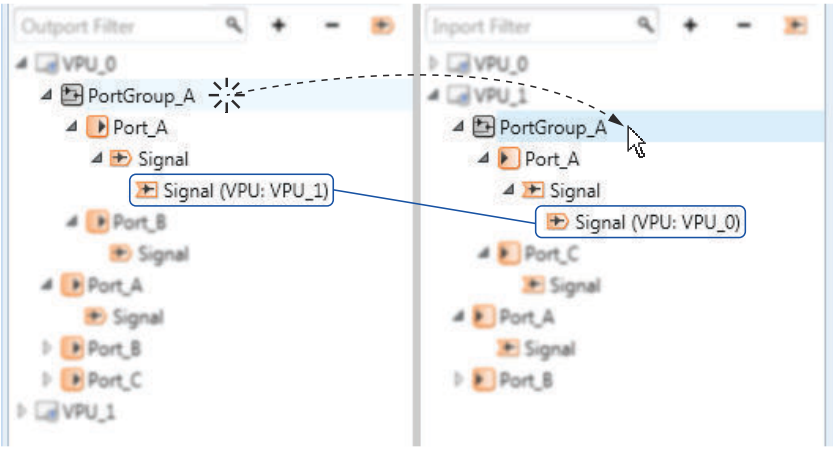
VEOS Player connects the following signals:

- VPU_0.PortGroup_A.Port_A.Signal to VPU_1.PortGroup_A.Port_A.Signal
- VPU_0.Port_A.Signal to VPU_1.Port_A.Signal
- VPU_0.Port_B.Signal to VPU_1.Port_B.Signal
- VPU_0.Port_C.Signal to VPU_1.PortGroup_A.Port_C.Signal

VEOS Player connects the signals with identical names. If a signal name is not unique, then the signal group names, port names and port group names are also taken into account and the longest match is used. This is why the signal VPU_0.PortGroup_A.Port_B.Signal is not connected in the example above.

Dragging a port group

You can drag a port group to another to connect their signals. The following illustration shows how this works for the example OSA:



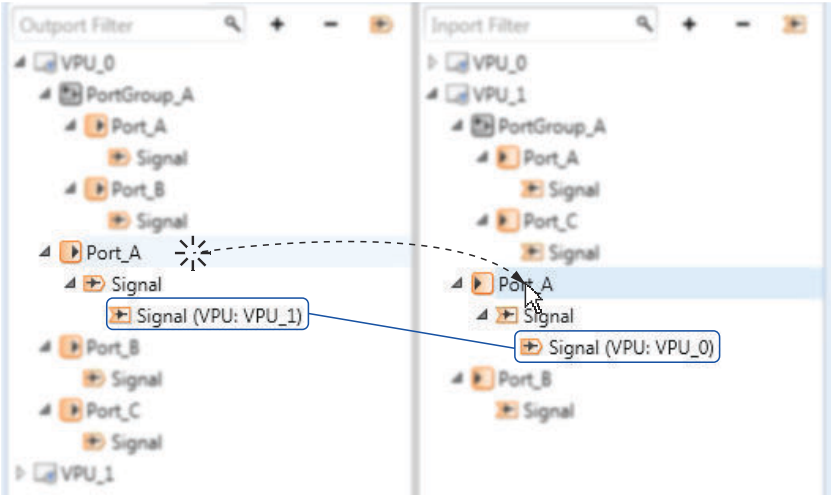
VEOS Player connects the following signals:

- VPU_0.PortGroup_A.Port_A.Signal to VPU_1.PortGroup_A.Port_A.Signal

VEOS Player connects the signals with identical names. If a signal name is not unique, then the port names and port group names are also taken into account and the longest match is used. This is why the signal VPU_0.PortGroup_A.Port_B.Signal, for example, is not connected in the example above.

Dragging a port

Identical port names, same hierarchies You can drag a port to another port with the same name to connect their signals. The following illustration shows how this works for the example OSA:

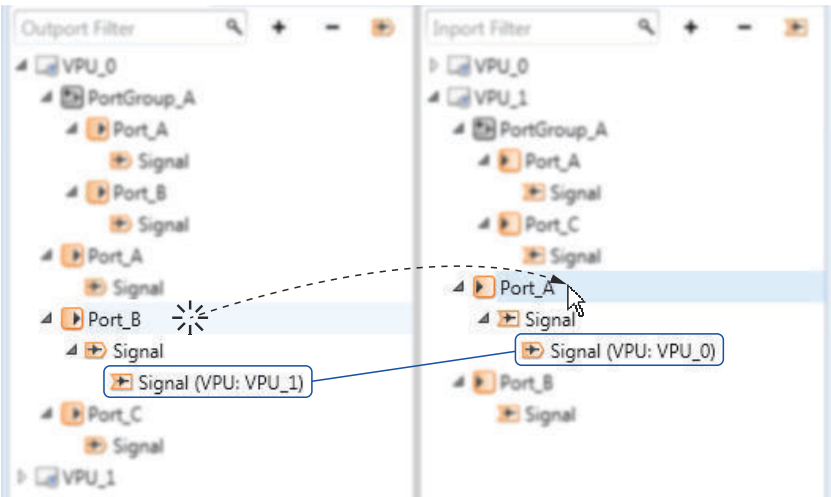


VEOS Player connects the following signals:

- Port_A.Signal to Port_A.Signal

VEOS Player connects these two signals because there is only one signal with the same name in the port group.

Different port names, same hierarchies You can drag a port to another port with a different name to connect their signals. The following illustration shows how this works for the example OSA:

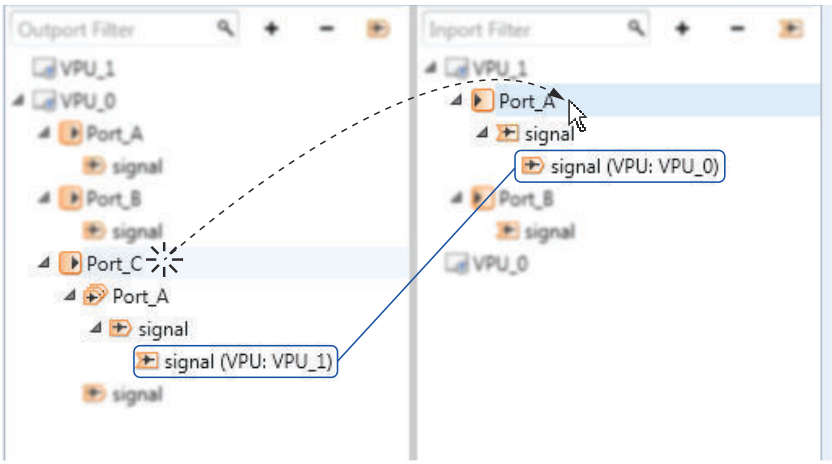


VEOS Player connects the following signals:

- Port_A.Signal to PortGroup_A.Port_C.Signal

VEOS Player connects these two signals because there is only one signal with the same name in the two ports.

Identical port names, different hierarchies You can drag a port to another port with a different name, which contains a subgroup with the same name to connect their signals. The following illustration shows how this works for the example OSA:

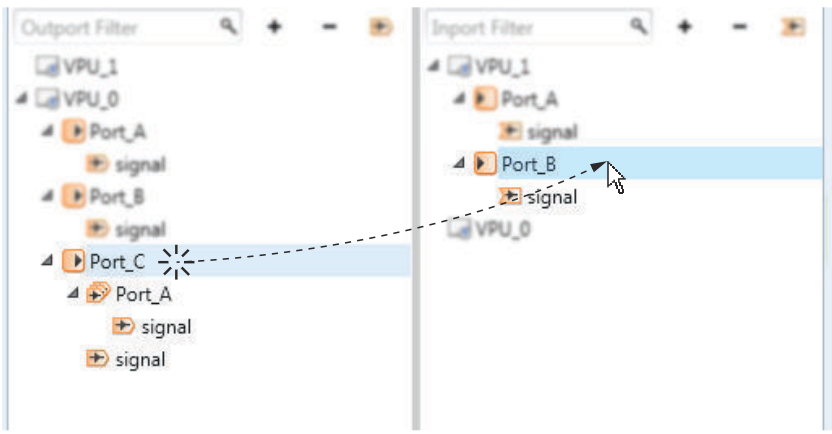


Port_C contains two signals with the same name: one directly below Port_C and one below Port_A. VEOS Player connects the following signals:

- Port_C.Port_A.signal to Port_A.signal

VEOS Player connects these two signals because there is only one signal in combination with the port above it that have the same names

Different port names, different hierarchies You can drag a port to another port with a different name, which contains subgroups with different names to connect their signals. The following illustration shows how this works for the example OSA:



Port_C contains two signals with the same name: one directly below Port_C and one below Port_A. VEOS Player cannot uniquely identify any signal, because their port names differ. VEOS Player connects no signal and shows a warning.

Related topics

Basics

Basics on Signal Connection Dependencies..... 100

Specifying Blocking or Non-Blocking Communication

Introduction

VEOS supports blocking and non-blocking communication between VPUs.

Blocking and non-blocking communication

VEOS lets you select between the *blocking* and *non-blocking* communication protocol types for the communication between VPUs.

Blocking communication A communication protocol type in which the VEOS Simulator transfers data between VPUs in the same (virtual) sampling step. If the VPU that provides the data has not calculated the data yet, the VPU that receives the data must wait until the other provides the data.

Note

If you use blocking communication, make sure that the sampling interval of an incoming signal is an integer multiple of the sampling interval of the connected outgoing signal. Otherwise, there might be communication issues.
VEOS monitors deadlock situations that result from blocking communication.

Non-blocking communication A communication protocol type in which the VEOS Simulator transfers data between VPUs after a (virtual) sampling step.

Depending on the configuration of your simulation models, the different communication protocol types can affect your simulation result. In general, you should choose non-blocking communication for fast, parallel simulation of your models. If your models show unstable behavior when you use non-blocking communication, use blocking communication at those points where a strong coupling between submodels exists, or where an immediate reaction is required in the same simulation step.

Blocking communication improves numerical correctness by waiting for up-to-date input signals. However, it increases waiting times and the number of context switches.

By default, VEOS assumes *non-blocking communication* between VPUs.

Specifying the communication protocol type

You can individually specify the communication protocol type for each port. To do so, you can assign ports with signal connections to a specific *communication*

package. Each communication package either has the blocking or non-blocking communication protocol type.

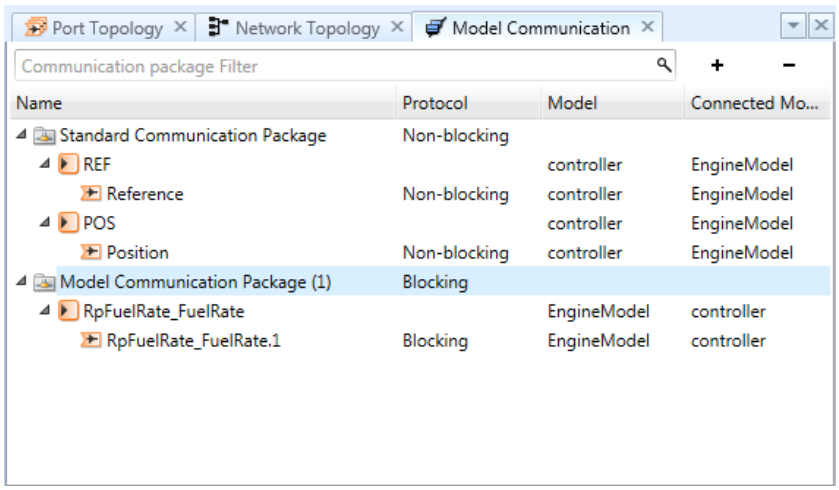
Communication package types VEOS provides the following communication package types:

Communication Package Type	Communication Protocol	Description
Standard communication package	Blocking or non-blocking	Default communication package that cannot be deleted
User communication package	Blocking or non-blocking	User-defined communication package. You can define as many model communication packages as required.

When you connect signals, VEOS automatically assigns the related incoming port to the *standard communication package*. If required, you can assign the port to any model communication package in the offline simulation application.

Model Communication pane VEOS Player's Model Communication pane lets you view and edit the communication packages of the offline simulation application. It also lets you manage the assignment of ports to communication packages.

The illustration below shows the Model Communication pane with the Standard Communication Package and a Model Communication Package as an example:



Tip

When you select a port in the Model Communication pane, the selection remains when you switch to the Port Topology pane. This lets you identify *ports with the same name*.

Related topics

References

Model Communication Pane..... 116

Connecting Communication Controllers to Communication Clusters

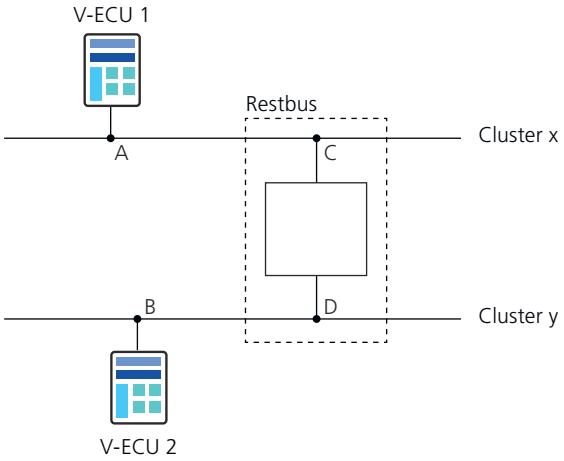
Introduction

The VEOS Player lets you connect communication controllers to communication clusters.

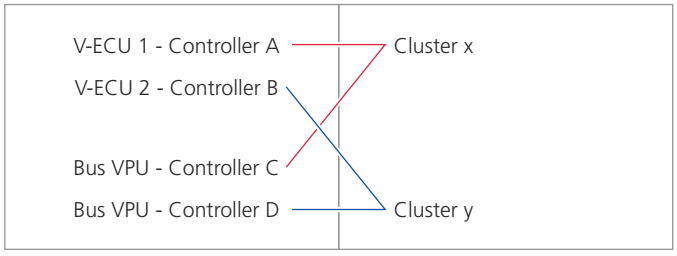
Example of connected controllers and clusters

The following example shows a simulation system that consists of two V-ECUs and a restbus.

- V-ECU 1 is connected to the communication cluster x via controller A.
- V-ECU 2 is connected to the communication cluster y via controller B.
- The restbus is connected to the communication cluster x via controller C and to the communication cluster y via controller D.



The following illustration shows a schematic of the simulation system and the interconnections in the **Network Topology** (refer to Network Topology Pane on page 118) pane.



Automatic connection

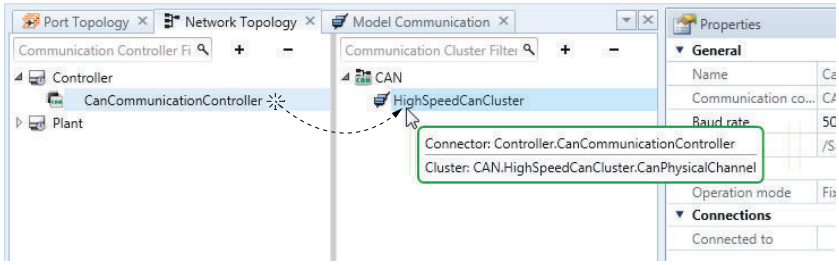
You can let VEOS Player automatically connect communication controllers to communication clusters by using the Autoconnect command.

Refer to Autoconnect on page 138.

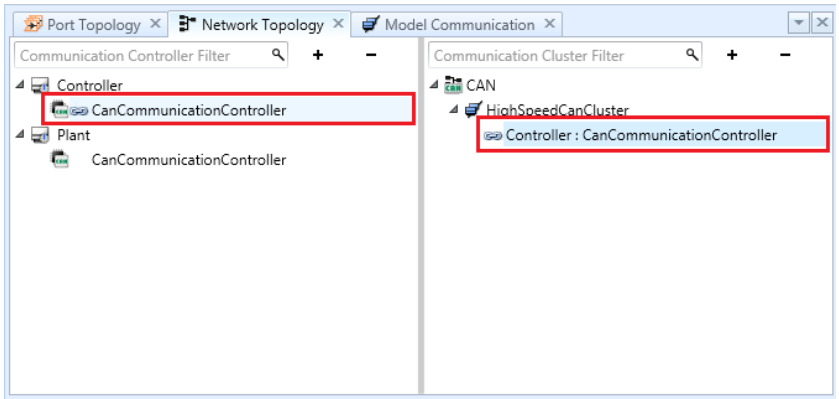
Manual connection

You can connect communication controllers to communication clusters by dragging the controllers from the left tree view to clusters in the right tree view of the Network Topology pane.

The following illustration shows an example:



The following illustration shows the result:



- A chain ( ) symbol is added to the source communication controller in the left tree view.
- A connection with a chain ( ) symbol is displayed below the target communication cluster.

Assigning controllers to channels

VEOS lets you *assign controllers to channels*.

Assigning controllers to channels in VEOS is equivalent to assigning bus access requests to bus accesses in the Bus Manager of ConfigurationDesk.

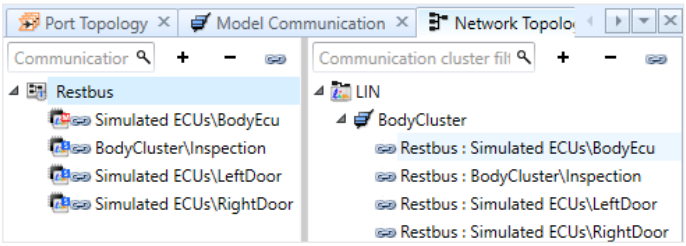
Application example *LIN frame collisions*, i.e., collisions of frames sent via two different cluster nodes, are an application example for assigning controllers to channels in VEOS:

When you create a LIN frame collision restbus model with the Bus Manager, the resulting BSC file can contain the following bus access requests:

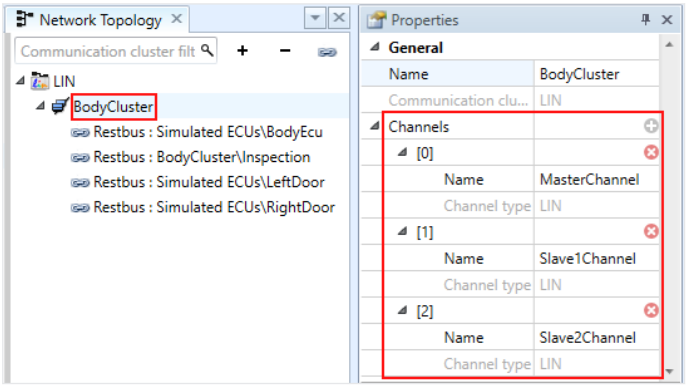
- One bus access request representing the **BodyEcu** LIN master ECU
- Two bus access requests representing the **LeftDoor** and **RightDoor** LIN slave ECUs
- One bus access request for inspection

For each bus access request in the BSC file, the VEOS Player creates a communication controller during import. All the controllers are connected to the **BodyCluster**.

The following illustration shows the VEOS **Network Topology** pane after BSC file import:



The **BodyCluster** has three LIN channels.



To simulate LIN frame collisions on VEOS between the two LIN slave ECUs, you have to assign these controllers to *different* channels of the cluster. Otherwise, if you assign the controllers to the *same* channel, no LIN frame collisions would occur because all the LIN frames would be sent by the same run-time channel.

The following table shows the channel assignment of the four controllers:

Controller	Assigned Channel
Simulated ECUs\BodyEcu	MasterChannel
BodyCluster\Inspection	MasterChannel
Simulated ECUs\LeftDoor	Slave1Channel
Simulated ECUs\RightDoor	Slave2Channel

Connection via tool automation

Connecting communication controllers to communication clusters is also possible via the `ConnectToCluster` method of the IBusConnector interface.

Refer to IBusConnector on page 349.

Related topics

Basics

Elements Generated for Bus Simulation Containers..... 80

References

Autoconnect..... 138
Communication Cluster..... 123
Communication Controller..... 124
Disconnect all Child Connections..... 143
IBusConnector..... 349
Network Topology Pane..... 118

Exporting System Model Containers

How to Export System Model Containers

Objective

The VEOS Player lets you export the open system model as a system model container (SMC). This allows you to exchange a system model on the basis of the related source code.

Basics

SMC file Abbreviation of system model container (SMC) file.

A file archive that contains an arbitrary number of the following containers, which belong to a specific system model:

- Bus simulation container (BSC)
- Functional Mock-up Unit (FMU)
- Simulink implementation container (SIC)
- V-ECU implementation (VECU)

An SMC file additionally contains the following information:

- Description of the data flow connections between the components of the system model

- Description of the bus communication connections between the components of the system model

An SMC file represents a system model container (SMC).

Version of exported SMC files For information on the version of exported SMC files, refer to Compatibility of VEOS 5.2 on page 40.

Preconditions

- The system model you want to export must be open in the **VEOS Player**. Refer to Open on page 159.
- An SMC file contains the implementations of the individual models of a specific system. VEOS therefore requires access to the model implementations files such as VECU or SIC during SMC file export. As a result, SMC file export is not possible if the system model to be exported results from an imported OSA file.

Restrictions

Compatibility aspects and possible naming conflicts:

- Keep in mind the compatibility aspects. Refer to Compatibility of VEOS 5.2 on page 40.
- If a model implementation contains custom C code with symbol prefixes and names that cause naming conflicts, you cannot integrate it in an offline simulation. For more information, refer to Avoiding Naming Conflicts when Integrating Custom C Code (SIL Testing Overview).

Method

To export a system model container

1. On the **Home** ribbon, select **Export**.
   The **Export SMC File** dialog opens.
2. In the **Export SMC File** dialog, specify the name and path of the export file, then click **Save**.

Result

You exported the open simulation system as a system model container (SMC).

Related topics

Basics

Compatibility of VEOS 5.2 40

References

Export 145

Reference Information

Where to go from here

Information in this section

Panes 116

Elements and Properties 122

Commands and Dialogs 134

Compiler Options 166

Panes

Where to go from here

Information in this section

Model Communication Pane 116
To display and edit the communication packages of the simulation system.

Network Topology Pane 118
To display and edit the bus simulation configuration of the offline simulation application.

Port Topology Pane 120
To display and edit the VPUs and signals of the offline simulation application.

Model Communication Pane

Purpose

To display and edit the communication packages of the simulation system.

Description

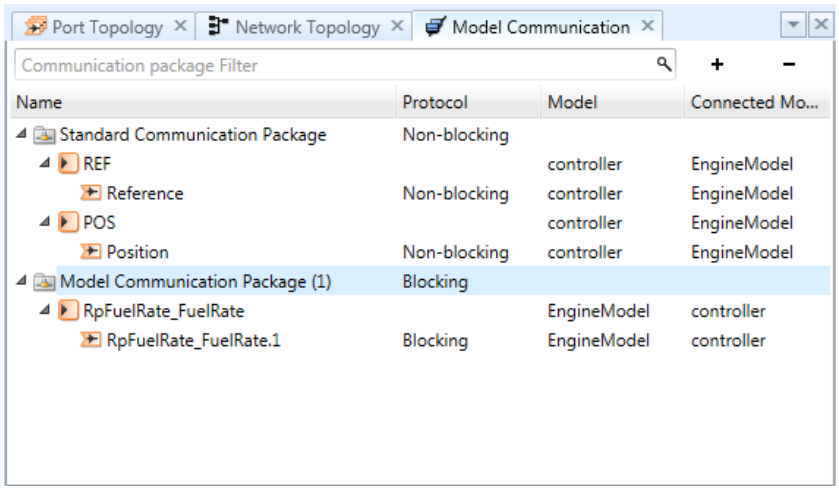
VEOS Player's Model Communication pane lets you view and edit the communication packages of the offline simulation application.

Communication package, communication protocol and ports A communication package specifies the communication protocol type (either blocking or non-blocking) of the assigned ports. You can assign ports to a

communication package. The ports get the communication protocol type of the communication package that they are assigned to.

For details on specifying the communication protocol type, refer to Specifying Blocking or Non-Blocking Communication on page 109.

Elements in the Model Communication pane The illustration below shows the Model Communication pane with the Standard Communication Package and a Model Communication Package as an example:



The Model Communication pane can display the following elements:

Symbol	Meaning
Communication package on page 126	
	Communication Package
Port on page 127	
	Port containing incoming signal groups or signals
Signal group on page 131	
	Signal group of incoming signals
Signal on page 129	
	Incoming signal

Tip

When you select a port in the Model Communication pane, the selection remains when you switch to the Port Topology pane. This lets you identify *ports with the same name*.

VEOS provides the following communication package types:

Communication Package Type	Communication Protocol	Description
Standard communication package	Blocking or non-blocking	Default communication package that cannot be deleted

Communication Package Type	Communication Protocol	Description
User communication package	Blocking or non-blocking	User-defined communication package. You can define as many model communication packages as required.

Filtering elements To reduce the number of elements displayed when working with large simulation systems, you can specify separate filter terms for each tree view in the Communication package Filter edit fields and use the Filter Editor.

Related commands

The Model Communication pane provides the following commands:

Purpose	Refer to
To reduce the number of displayed elements in the selected tree view.	Filtering Elements on page 148
To expand, collapse, and hide elements in the selected tree view.	Expanding and Hiding Elements on page 145
To remove the selected element from the open simulation system.	Delete on page 143
To open the Filter Editor to edit a filter rule.	Filter Editor on page 147

Related topics

Basics

Specifying Blocking or Non-Blocking Communication..... 109

References

Add New Model Communication Package..... 138
Communication Package..... 126
Delete..... 143
Expanding and Hiding Elements..... 145
Filter Editor..... 147
Filtering Elements..... 148

Network Topology Pane

Purpose

To display and edit the bus simulation configuration of the offline simulation application.

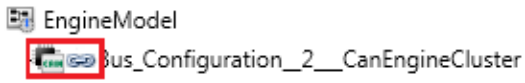
Description

Displays all the bus communication elements of the offline simulation application and their properties and lets you edit the bus simulation configuration.

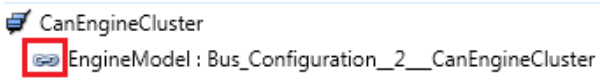
Network Topology pane elements The Network Topology pane can display the following elements:

Symbol	Meaning
VPU	
	Classic V-ECU
	Adaptive V-ECU
	Environment VPU
	Controller VPU
	Bus VPU
Bus simulation - communication elements	
	Communication controller (e.g., CAN)
	Bus type (e.g., CAN)
	Communication cluster
	Controller-to-cluster connection

Connected communication controllers appear with a chain symbol as children of the bus VPU or V-ECUs in the Network Topology pane. The controller name represents the bus configuration name and the connected communication cluster name. Refer to the following example.



Connected communication clusters appear as physical channels with a chain symbol in the Network Topology pane. The name of the channel represents the bus VPU name, the bus configuration name, and the cluster name. Refer to the following example.



Filtering elements To reduce the number of elements displayed when working with large simulation systems, you can specify separate filter terms for each tree view in the Communication controller Filter and the Communication cluster Filter edit fields.

Related commands

The Network Topology pane provides the following commands:

Purpose	Refer to
To reduce the number of displayed elements in the selected tree view.	Filtering Elements on page 148

Purpose	Refer to
To expand, collapse, and hide elements in the selected tree view.	Expanding and Hiding Elements on page 145
To remove the selected element from the open simulation system.	Delete on page 143
To disconnect all the selected child connections of the open simulation system.	Disconnect all Child Connections on page 143

Related topics

Basics

Connecting Communication Controllers to Communication Clusters..... 111

References

Delete..... 143
Disconnect all Child Connections..... 143
Expanding and Hiding Elements..... 145
Filtering Elements..... 148

Port Topology Pane

Purpose

To display and edit the VPUs and signals of the offline simulation application.

Description

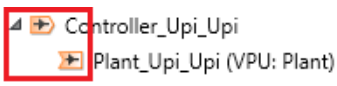
Displays VPUs in the offline simulation application and their properties. If the VPUs are configured with signals, the signals are also displayed.

Elements displayed in the Port Topology pane The Port Topology pane can display the following elements:

Symbol	Meaning
VPU	
	Classic V-ECU
	Adaptive V-ECU
	Environment VPU
	Controller VPU
	Bus VPU
Port group	
	Port group

Symbol	Meaning
Port	
	Port of outgoing signals
	Port of incoming signals
Signal	
	Signal group of outgoing signals
	Signal group of incoming signals
Signal	
	Outgoing signal
	Incoming signal

Connected signals appear as children with the name of the connected VPU in brackets. Refer to the following example.



Refer to Basics on Signal Connection Dependencies on page 100.

Filtering elements To reduce the number of elements displayed when working with large simulation systems, you can specify separate filter terms for each tree view in the **Outports Filter** and **Inports Filter** edit fields.

Related commands

The Port Topology pane provides the following commands:

Purpose	Refer to
To reduce the number of displayed elements in the selected tree view.	Filtering Elements on page 148
To expand, collapse, and hide elements in the selected tree view.	Expanding and Hiding Elements on page 145
To remove the selected element from the open simulation system.	Delete on page 143

Related topics

Basics

Basics on Signal Connection Dependencies..... 100

References

Delete..... 143
Expanding and Hiding Elements..... 145
Filtering Elements..... 148

Elements and Properties

Where to go from here

Information in this section

Bus Type 122
A structural element in the right tree view of the Network Topology pane. It contains communication clusters and their connections to communication controllers.

Communication Cluster 123
An element that specifies the topological connection of communicating ECUs.

Communication Controller 124
A hardware device of an ECU for sending information to and receiving information via a communication channel.

Communication Package 126
An element that specifies the communication protocol type (either blocking or non-blocking) of the assigned ports.

Controller-to-Cluster Connection 127
A connection between a communication controller and a communication cluster.

Port 127
A structural element of a VPU.

Port Group 128
A structural element of a VPU.

Signal 129
An element of a VPU that allows direct communication with other VPUs.

Signal Group 131
A structural element of a VPU.

VPU (V-ECU / Environment VPU / Controller VPU / Bus VPU) 131
Abbreviation of *virtual processing unit*. A VPU is part of an offline simulation application in VEOS. Each VPU runs in a separate process of the PC.

Bus Type

Definition

Bus type A structural element in the right tree view of the Network Topology pane. It contains communication clusters and their connections to communication controllers.

Element symbol

The element has the following symbols:

- (CAN bus)
- (LIN bus)
- (Idealized bus)
- (Ethernet bus)

General properties

Bus type Displays the bus type.

Related topics

References

Network Topology Pane..... 118

Communication Cluster

Definition

Communication cluster An element that specifies the topological connection of communicating ECUs. In a communication cluster, the communication controllers of different ECUs are interconnected.

Element symbol

Communication clusters are represented by the following symbol:

Communication cluster types

VEOS supports the following communication cluster types in bus simulation:

Cluster Type	Description
CAN	Cluster in a bus according to the CAN standard.
Idealized	Cluster in an idealized bus used to simulate models with bus configurations according to unsupported standards such as FlexRay.
LIN	Cluster in a bus according to the LIN standard.
Ethernet	Cluster in a bus according to the Ethernet standard.

General properties

Name Lets you change the name of the cluster.

Communication cluster type Displays the bus type the cluster is specified for.

Channels Lets you specify the channels of the selected communication cluster.

After you connect a communication controller to the selected communication cluster, you can assign one of the channels to be used for bus communication.

Related topics

Basics

Connecting Communication Controllers to Communication Clusters..... 111

References

Network Topology Pane..... 118

Communication Controller

Definition

Communication controller A hardware device of an ECU for sending information to and receiving information via a communication channel.

Element symbol

The element has one of the following symbols:

- (Communication controller (connected))
- (Communication controller (unconnected))

General properties

Name Displays the name of the selected communication controller.

Communication controller type Displays the bus type the controller is specified for.

Origin Displays the origin name of the bus configuration and the communication cluster.

Settings properties

Operation mode Displays the operation mode of the controller.

The following table explains the operation modes, which depend on the controller type:

Mode	Description
CAN	
FixedDataRate	The communication controller operation mode is *standard CAN*. You can specify the baud rate.
FlexibleDataRate	The communication controller operation mode is *CAN with Flexible Data Rate* (CAN FD). You can specify two baud rate values: ▪ Baud rate for the arbitration phase ▪ Baud rate for the data phase
Unknown	The communication controller operation mode is *unknown*. You cannot specify the baud rate for communication controllers with an unknown operation mode[1].
LIN	
Master	The communication controller operation mode is master.
Slave	The communication controller operation mode is slave.

[1] Bus simulation container (BSC) files can contain *CAN frame gateways* that are used to exchange CAN frames between two communication clusters. VEOS applies the *unknown* operation mode to the communication controllers created for bus access requests that result from CAN frame gateways, and it derives the baud rates of these communication controllers from the settings of the other communication controllers connected to the same communication cluster. For more information on CAN frame gatewaying, refer to Specifying CAN Gateways (Bus Manager (Stand-Alone) Implementation Guide).

Baud rate (Not available for the Unknown operation mode) Lets you specify the transmission rate in baud.

Data phase baud rate (Available for the FlexibleDataRate operation mode only) Lets you specify the transmission rate for the CAN FD data phase in baud.

Connections properties

Connected to Displays the name of the communication cluster to which the communication controller is connected.

Channel Name Lets you select the name of one of the channels of the communication cluster the communication controller is connected to.

You can assign multiple controller requests to one channel.

Communication Matrix properties

Name Displays the name of the communication matrix the selected communication controller is part of.

Content hash Displays a unique ID for the communication matrix the selected communication controller is part of.

Bus Configuration properties

Name Displays the bus VPU name of the selected communication controller.

Part Displays a characterization of the communication matrix extract.

Related topics

Basics

Connecting Communication Controllers to Communication Clusters..... 111

References

Network Topology Pane..... 118

Communication Package

Definition

Communication package An element that specifies the communication protocol type (either blocking or non-blocking) of the assigned ports.

Element symbol

The element has the following symbol:

General properties

Name Displays the package name.

Protocol Lets you specify the blocking and non-blocking communication protocol type of the communication package.

For more information, refer to Specifying Blocking or Non-Blocking Communication on page 109.

Package type Displays the type of package.

For more information, refer to Specifying Blocking or Non-Blocking Communication on page 109.

Related topics

Basics

Specifying Blocking or Non-Blocking Communication 109

References

Model Communication Pane 116

Controller-to-Cluster Connection

Definition

Controller-to-cluster connection A connection between a communication controller and a communication cluster.

Element symbol

Controller-to-cluster connections are represented by a chain symbol ( ).

General properties

Communication cluster type Displays the type of the communication cluster, i.e., the bus type.

Communication controller type Displays the bus type the controller is specified for.

Related topics

Basics

Connecting Communication Controllers to Communication Clusters 111

References

Network Topology Pane 118

Port

Definition

Port A structural element of a VPU.

A port can contain the following elements:

- Signal groups
- Signals

Element symbol

The element has one of the following symbols:

- (Port of outgoing signal groups or signals)
- (Port of incoming signal groups or signals)

General properties

Name Displays the name of the port.

Related topics

References

Communication Package..... 126
Model Communication Pane..... 116
Network Topology Pane..... 118
Port Topology Pane..... 120

Port Group

Definition

Port group A structural element of a VPU.

A port group can contain the following elements:

- Port groups
- Ports

Element symbol

The element has the following symbol:

General properties

Name Displays the name of the port group.

Related topics

References

Communication Package..... 126
Model Communication Pane..... 116
Network Topology Pane..... 118
Port Topology Pane..... 120

Signal

Definition

Signal An element of a VPU that allows direct communication with other VPUs.

There are different signal types:

- Fixed-size signals
- Variable-size signals
- Binary signals

Element symbol

The element has one of the following symbols:

- (Incoming signal)
- (Outgoing signal)

General properties

Description Displays a description of the selected signal.

Name Displays the name of the selected signal.

Definition properties

Direction Displays the transfer directions of the array elements of the selected signal. The possible values are In, Out.

Origin Displays the configurations (e.g., DAP module configuration created in SystemDesk) from which the selected signal originates. For a signal originating from a Simulink port, the original name of the Simulink port is displayed.

Data width Displays the width of the selected signal.

- For scalar signals, the width is 1.
- For array signals, the width is >1.

  For fixed-size signals, the number of array elements is constant during run time.

  For variable-size signals, the number of array elements varies during run time in the range 0 ... (data width).

Unit Displays the physical units of the array elements of the selected signal. The unit value is empty if the value is raw.

Initial value (Available only for incoming signals) Lets you enter an initial value for the selected incoming signal.

For array signals, you can enter an initial value for each array element in a separate line.

Note

When you import a BSC or an SIC file to VEOS, initial values for model port blocks that were set during generation in ConfigurationDesk or the Model Interface Package for Simulink are ignored and set to *0*.

Data type Displays the data type of the selected signal.

Signals with the *binary* data type are *binary signals*.

MIME Type (Available only for signals of the binary data type, i.e., for binary signals) Displays the MIME type of the selected signal.

Sampling interval Displays the sampling interval of the selected signal in seconds.

Connections properties

(Available only for connected signals)

Receives from Displays the names of the sending signals the selected signal is connected to.

Sends to Displays the names of the receiving signals the selected signal is connected to.

Communication Package properties

(Available only for incoming, connected signals)

Communication Package Lets you select the communication package of the signal.

If you select another communication package for the selected signal, the entire port is assigned to this communication package.

Name Displays the package name.

Protocol Lets you specify the blocking and non-blocking communication protocol type of the communication package.

For more information, refer to Specifying Blocking or Non-Blocking Communication on page 109.

Package type Displays the type of package.

For more information, refer to Specifying Blocking or Non-Blocking Communication on page 109.

Related topics

Basics

Basics on Signal Connection Dependencies..... 100

Specifying Blocking or Non-Blocking Communication..... 109
Ways to Connect VPU Signals..... 102

References

Communication Package..... 126
Model Communication Pane..... 116
Port Topology Pane..... 120

Signal Group

Definition

Signal group A structural element of a VPU.

A signal group can contain the following elements:

- Signal groups
- Signals

Element symbol

The element has one of the following symbols:

- (Signal group of outgoing signals)
- (Signal group of incoming signals)

General properties

Name Displays the name of the signal group.

Related topics

References

Network Topology Pane..... 118
Port Topology Pane..... 120

VPU (V-ECU / Environment VPU / Controller VPU / Bus VPU)

Definition

VPU Abbreviation of *virtual processing unit*. A VPU is part of an offline simulation application in VEOS. Each VPU runs in a separate process of the PC.

VPU is also the generic term for:

- V-ECUs
- Environment VPUs

- Controller VPUs
- Bus VPUs

Adaptive V-ECU A V-ECU that allows for the *dynamic* integration of new applications and changes to the network communication at run time, which is not possible with a *statically* configured classic V-ECU. Communication is service-oriented via Ethernet.

An example are V-ECUs developed according to the AUTOSAR Adaptive Platform.

Classic V-ECU A V-ECU whose resources, such as the operating system and network communication, are statically configured. Therefore, in contrast to an adaptive V-ECU, you cannot add applications or make changes to the network communication at run time. Communication is mainly signal-based, as in network communication via CAN or FlexRay, but later classic V-ECUs also support service-oriented communication via Ethernet and SOME/IP.

An example are V-ECUs developed according to the AUTOSAR Classic Platform.

Environment VPU The executable of an environment model built for the VEOS platform. An environment VPU is part of an offline simulation application (OSA).

Controller VPU The executable of a controller model built for the VEOS platform. A controller VPU is part of an offline simulation application (OSA).

Bus VPU The executable of a bus model built for the VEOS platform. A bus VPU is part of an offline simulation application (OSA).

Element symbols

The element has one of the following symbols:

- (Classic V-ECU)
- (Adaptive V-ECU)
- (Environment VPU)
- (Controller VPU)
- (Bus VPU)

General properties

Author Displays the author of the selected VPU.

Description Displays the description for the selected VPU.

Memory Size (MBytes) (Available only for adaptive V-ECUs) Displays the memory (RAM) size used by the selected VPU.

Name Displays the name of the selected VPU.

Number of CPUs (Available only for adaptive V-ECUs) Displays the number of CPUs used by the selected VPU.

Step Time (Available only for adaptive V-ECUs) Displays the length of a simulation step of the selected VPU.

Container info properties

Source format Displays the source format of the VPU, such as V-ECU Implementation.

Format version Displays the version number of the source format in digits.

Exporting tool Displays the tool that was used to export the container, such as ConfigurationDesk.

Content information Displays information about the container content.

Build date Displays the date with day and time when the container was imported.

Model creation date Displays date with day and time when the model was created.

Simulation time options properties

Co-simulation For future use.

Process ID Displays the application ID for the selected VPU. The ID is displayed when the offline simulation application is loaded to VEOS.

You can use the application ID to identify the related VPU process when you perform source code debugging in offline simulation. For further details on source code debugging, refer to Basics on Debugging Source Code in an Offline Simulation on page 202.

Timer drift Lets you specify the timer drift value. The value specifies the drift between the local VPU time and the global simulation time.

The timer drift value must be greater than - 1.

- A drift value less than 0 means that the local VPU timer is slower than the global simulation timer. For example, if the timer drift value is - 0.5, the global simulation time runs twice as fast as the local VPU time.
- If the timer drift value is 0, the local VPU time and the global simulation time are identical.
- With a drift value greater than 0, the local timer runs faster than the global timer. For example, if the timer drift value is 1, the local VPU time advances twice as fast as the global simulation time.

Timer offset Lets you specify the timer offset value. The value specifies the time offset between the local VPU time and the global simulation time. Specifying a timer offset allows you to shift the local VPU time linearly on the global time axis. A positive timer offset value, for example, means that the global simulation time is in the future in relation to the local VPU time.

The relation between the local VPU time t_{local} and the global simulation time t_{global} is calculated like this:

$t_{global} = (t_{local} + \text{offset}) / (1 + \text{drift})$

Build time options properties

Contains debug information Displays whether the selected VPU contains debug information or not. For example, when you import a V-ECU implementation, you can enable source code debugging. Refer to Import on page 152.

For details on debugging source code in offline simulation, refer to Basics on Debugging Source Code in an Offline Simulation on page 202.

Target simulation module identifier Displays:

- The target the VPU was generated for (`HostPC` for SIL)
- The compiler used to generate the VPU

XCP on Ethernet port 1 (Available only for classic V-ECUs and controller VPUs (SIC + A2L)) Displays the number of the port used by VEOS to access the generated VPU via XCP on Ethernet.

XCP on Ethernet port 2 (Available only for classic V-ECUs and controller VPUs (SIC + A2L)) Displays the number of the port used by an external tool such as ControlDesk to access the generated VPU via XCP on Ethernet.

Hypervisor options properties

Hypervisor Connection Type Displays the type of the connection used by the hypervisor to access the network.

Hypervisor MAC-Range Displays the MAC-address range of the hypervisor.

Related topics

References

Port Topology Pane..... 120

Commands and Dialogs

Where to go from here

Information in this section

Add New Communication Cluster..... 136
To add a new communication cluster to the offline simulation application.

Add New Model Communication Package..... 138
To add a new model communication package to the offline simulation application.

Autoconnect..... 138
To automatically connect all unconnected incoming signals to outgoing signals with the same name. And: To automatically connect all unconnected communication controllers to communication clusters with the same bus type, and optionally according to the controller baud rates.

Clear Recently Exported Files..... 140
To clear the list of recently exported files.

Clear Recently Imported Files..... 140
To clear the list of recently imported files.

Clear Recently Used Files..... 141
To clear the list of recently used files.

Close..... 142
To close the open simulation system.

Collapse All Nodes..... 142
To collapse all the elements in the tree view(s) of the VEOS Player.

Delete..... 143
To remove the selected element from the open simulation system.

Disconnect all Child Connections..... 143
To disconnect all the selected child connections of the open simulation system.

Expand All Nodes..... 144
To expand all the elements in the tree view(s) of the VEOS Player.

Expanding and Hiding Elements..... 145
To expand, collapse, and hide elements in the selected tree view.

Export..... 145
To export the open simulation system as a system model container (SMC).

Export the Variable Description Files (A2L or TRC) when the OSA File is Saved..... 146
To export variable description files after saving the simulation system via the Save or Save As command.

Filter Editor..... 147
To open the Filter Editor to edit a filter rule.

Filtering Elements..... 148
To reduce the number of displayed elements in the selected tree view.

Hide Connected Communication Clusters..... 149
To hide the connected communication clusters in the right tree view of the Network Topology pane.

Hide Connected Communication Controllers..... 150
To hide the connected communication controllers in the left tree view of the Network Topology pane.

Hide Connected Incoming Ports..... 150
To hide the connected incoming ports in the Port Topology pane.

Hide Connected Outgoing Ports..... 151
To hide the connected outgoing ports in the Port Topology pane.

Import..... 152
To import model implementation container files, e.g., V-ECU implementations and Simulink implementation containers, and VPUs into a simulation system.

Info..... 158
To display information about the opened offline simulation application (OSA) file.

New..... 159
To create a new OSA file.

Open..... 159
To open a simulation system from an offline simulation application (OSA) file.

Recently Used..... 160
To open one of the most recently opened OSA files in the VEOS Player.

Replace..... 161
To replace the model container file of the selected VPU.

Save..... 161
To save the open simulation system as an offline simulation application (OSA) file.

Save As..... 162
To save the open simulation system as an offline simulation application (OSA) file under a new name.

Simulation Target Manager..... 163
To specify simulation targets and make them available to the VEOS build process.

Update..... 166
To update the model container file of the selected VPU.

Add New Communication Cluster

Access

You can access this command via:

Ribbon	Home - Home
Context menu of	None

Shortcut key	None
Icon	

Purpose

To add a new communication cluster to the offline simulation application.

Description

Typically, communication clusters are part of the open offline simulation application or the container files you imported to the VEOS Player.

If a communication cluster in the offline simulation application is accidentally deleted, you do not need to reimport the original file. As an alternative, the VEOS Player lets you add a new communication cluster and connect the appropriate communication controllers manually to it.

VEOS supports the following communication cluster types in bus simulation:

Cluster Type	Description
CAN	Cluster in a bus according to the CAN standard.
Idealized	Cluster in an idealized bus used to simulate models with bus configurations according to unsupported standards such as FlexRay.
LIN	Cluster in a bus according to the LIN standard.
Ethernet	Cluster in a bus according to the Ethernet standard.

Result

A communication cluster is added to the offline simulation application in the right tree view of the Network Topology pane.

Related topics

Basics

Connecting Communication Controllers to Communication Clusters..... 111

References

Network Topology Pane..... 118

Add New Model Communication Package

Access

You can access this command via:

Ribbon	Home – Home
Context menu of	None
Shortcut key	None
Icon	

Purpose

To add a new model communication package to the offline simulation application.

Description

Each model communication package either has the blocking or non-blocking communication protocol type. To specify the communication protocol type of a specific port, assign the group to the related communication package.

For details on blocking and non-blocking communication and specifying the communication type, refer to Specifying Blocking or Non-Blocking Communication on page 109.

Related topics

Basics

Specifying Blocking or Non-Blocking Communication 109

References

Model Communication Pane 116

Autoconnect

Access

You can access this command via:

Ribbon	Home - Home
Context menu of	None
Shortcut key	None
Icon	

Purpose

- To automatically connect all unconnected incoming signals to outgoing signals with the same name.
- To automatically connect all unconnected communication controllers to communication clusters with the same bus type, and optionally according to the controller baud rates.

Result

The VEOS Player connects the unconnected elements, if possible.

Description

- If you click the main icon part of the **Autoconnect** button, VEOS Player does the following:
  - All previously unconnected incoming and outgoing signals with identical names are connected.
  - All previously unconnected communication controllers are connected to communication clusters with the same bus type.
- If you click the arrow part of the **Autoconnect** button, you can select one of the following commands:

Command	Description
	▪ To automatically connect all unconnected incoming signals to outgoing signals with the same name. ▪ To automatically connect all unconnected communication controllers to communication clusters with the same bus type. If no communication cluster with the same bus type is available in the current simulation system, VEOS Player creates a new one and establishes the connection afterwards.
	To automatically connect all unconnected incoming signals to outgoing signals with the same name.
	To automatically connect all unconnected communication controllers to communication clusters with the same bus type. If no communication cluster with the same bus type is available in the current simulation system, VEOS Player creates a new one and establishes the connection afterwards.
	To automatically connect all unconnected communication controllers to communication clusters with the same bus type, and according to the baud rates of the controllers: Controllers with the same baud rate are grouped, and each controller group is connected to a separate cluster with the same bus type. If no communication cluster with the same bus type is available in the current simulation system, VEOS Player creates a new one and establishes the connection afterwards.

Related topics

Basics

Basics on Signal Connection Dependencies..... 100

Connecting Communication Controllers to Communication Clusters..... 111

Clear Recently Exported Files

Access

You can access this command via:

Ribbon	Home - Export
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To clear the list of recently exported files.

Result

The list of recently exported files is cleared.

Description

You can clear either the entire list by clicking the command's button or remove one of the items in the list by clicking ✕.

Related topics

References

Export..... 145

Clear Recently Imported Files

Access

You can access this command via:

Ribbon	Home - Import
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To clear the list of recently imported files.

Result

The list of recently imported files is cleared.

Description

You can clear either the entire list by clicking the command's button or remove one of the items in the list by clicking ✕.

Related topics

References

Import..... 152

Clear Recently Used Files

Access

You can access this command via:

Ribbon	▪ File - Recently Used ▪ Home - Open
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To clear the list of recently used files.

Description

You can clear the entire list by clicking the command's button or remove one of the items in the list by clicking ✕.

Result

The list of recently used files is cleared.

Related topics

References

Open..... 159
Recently Used..... 160

Close

Access

You can access this command via:

Ribbon	File
Context menu of	None
Shortcut key	`Ctrl` + `W`, `Ctrl` + `F4`
Icon	None

Purpose

To close the open simulation system.

Result

The active simulation system is closed. If the simulation system was changed since the last save operation, you are asked whether you want to save it.

Related topics

HowTos

How to Open an OSA File.....33

Collapse All Nodes

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Port Topology pane ▪ Network Topology pane ▪ Model Communication pane
Shortcut key	None
Icon	–

Purpose

To collapse all the elements in the tree view(s) of the VEOS Player.

Result

All the elements in tree view(s) are collapsed.

Related topics

References

Expand All Nodes..... 144

Delete

Access

You can access this command via:

Ribbon	None
Context menu of	▪ VPU (**Port Topology** pane) ▪ Signal connections (**Port Topology** pane) ▪ Communication cluster (**Network Topology** pane) ▪ Communication package (**Model Communication** pane)
Shortcut key	**`Del`**
Icon	None

Purpose

To remove the selected element from the open simulation system.

Result

The selected element is removed from the simulation system.

Related topics

References

Import..... 152

Disconnect all Child Connections

Access

You can access this command via:

Ribbon	None
Context menu of	▪ VPU (**Port Topology** pane) ▪ Port group (**Port Topology** pane) ▪ Port (**Port Topology** pane)

	▪ Signal group (Port Topology pane) ▪ Signal (Port Topology pane) ▪ Connection (Network Topology pane) ▪ Communication controller (Network Topology pane)
Shortcut key	**Del**
Icon	None

Purpose

To disconnect all the selected child connections of the open simulation system.

Result

All the selected child connections of the simulation system are disconnected.

Related topics

Basics

Connecting Communication Controllers to Communication Clusters..... 111

Expand All Nodes

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Port Topology pane ▪ Network Topology pane ▪ Model Communication pane
Shortcut key	None
Icon	+

Purpose

To expand all the elements in the tree view(s) of the VEOS Player.

Result

All the elements in the tree view(s) are expanded.

Related topics

References

Collapse All Nodes..... 142

Expanding and Hiding Elements

Purpose

To expand, collapse, and hide elements in the selected tree view.

Available commands

You can use the following commands:

Nodes

- Collapse All Nodes on page 142
- Expand All Nodes on page 144

Ports

- Hide Connected Incoming Ports on page 150
- Hide Connected Outgoing Ports on page 151

Communication controllers and clusters

- Hide Connected Communication Clusters on page 149
- Hide Connected Communication Controllers on page 150

Related topics

References

Collapse All Nodes..... 142
Expand All Nodes..... 144
Hide Connected Communication Clusters..... 149
Hide Connected Communication Controllers..... 150
Hide Connected Incoming Ports..... 150
Hide Connected Outgoing Ports..... 151

Export

Access

You can access this command via:

Ribbon	Home - Home
Context menu of	None
Shortcut key	`Ctrl` + `E`
Icon	

Purpose

To export the open simulation system as a system model container (SMC).

Description

The Export SMC File dialog is opened for you to save the currently opened simulation system as an SMC file.

Note

An SMC file contains the implementations of the individual models of a specific system. VEOS therefore requires access to the model implementations files such as VECU or SIC during SMC file export. As a result, SMC file export is not possible if the system model to be exported results from an imported OSA file.

Export SMC File dialog

File name Lets you specify the name and path of the export file.

Save as type Displays the SMC export file type.

Save Lets you export the SMC file.

Related topics

HowTos

How to Export System Model Containers..... 114

References

Clear Recently Exported Files..... 140

Export the Variable Description Files (A2L or TRC) when the OSA File is Saved

Access

You can access this command via:

Ribbon	File – Preferences
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To export variable description files after saving the simulation system via the Save or Save As command.

Result

Each time you save the simulation system, the variable description files are exported.

Related topics

References

Preferences..... 408
Save..... 161
Save As..... 162

Filter Editor

Access

You can access this command via:

Ribbon	None
Context menu of	Column header in: ▪ **Model Communication** pane ▪ Select Simulator dialog ▪ Select Simulator
Shortcut key	None
Icon	

Purpose

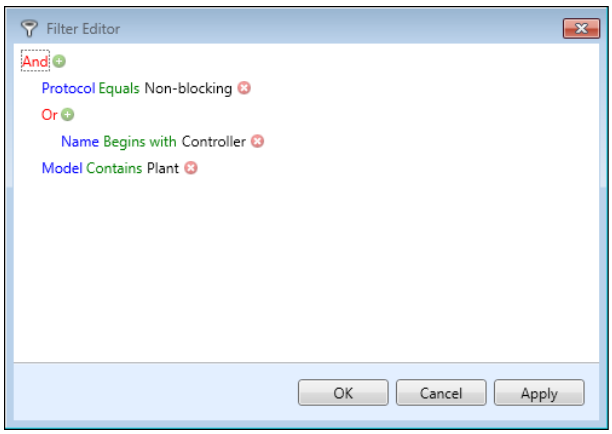
To open the Filter Editor to edit a filter rule.

Result

The **Filter Editor** is opened to create and apply a filter rule.

Filter Editor

The **Filter Editor** provides a tree view for specifying a hierarchical filter rule, for example, by using Boolean operators.



A filter can consist of several conditions that can be combined by a logical operation. In a condition, a column header is compared with a specified value. To specify a filter, the colored words can be modified.

Red word Click the red word to select a logical operator or a command for combining conditions. The following logical operators are available:

Logical Operation	Description
And	All combined conditions must be true.
Or	At least one of the combined condition must be true.
NotAnd	Not all of the combined conditions must be true.
NotOr	Neither of the combined conditions must be true.

The following commands are available:

Command	Description
Add Condition, , +, or `Insert`	Lets you add a condition.
Add Group	Lets you add a condition group.
Clear All	Lets you clear all conditions.
	Lets you clear a condition.
Remove Group	Lets you remove a group.

Blue word Click the blue word to select a column of the table whose value is then compared with a specified value.

Green word Click the green word to select a relational operator.

You can apply a filter rule by pressing the OK or Apply button. Pressing the OK button closes the editor.

Related topics

References

Clear Filter 391
Model Communication Pane 116
Select Simulator 219

Filtering Elements

Access

You can access this command via:

Ribbon	None
Context menu of	None

Shortcut key	None
Icon	

Purpose

To reduce the number of displayed elements in the selected tree view.

Description

You can specify separate filter terms for each tree view in the edit fields of the VEOS Player's panes.

The filter is applied as soon as you click or press **Enter**.

The filter is not case-insensitive. This means that it does not differentiate between lower and upper case letters when filtering, both cases are included.

Related topics

References

Panes..... 116

Hide Connected Communication Clusters

Access

You can access this command via:

Ribbon	None
Context menu of	None
Shortcut key	None
Icon	(right tree view of **Network Topology** pane)

Purpose

To hide the connected communication clusters in the right tree view of the Network Topology pane.

Description

If your simulation system contains a lot of connected communication clusters, it can be helpful to hide them for a better overview.

Result

The connected communication clusters are hidden in the tree view and the **Hide Connected Communication Clusters** icon changes to .

Related topics

References

Expanding and Hiding Elements..... 145

Hide Connected Communication Controllers

Access

You can access this command via:

Ribbon	None
Context menu of	None
Shortcut key	None
Icon	(left tree view of **Network Topology** pane)

Purpose

To hide the connected communication controllers in the left tree view of the Network Topology pane.

Description

If your simulation system contains a lot of connected communication controllers, it is helpful to hide them for a better overview.

Result

The connected communication controllers are hidden in the tree view and the **Hide Connected Communication Controllers** icon changes to .

Related topics

References

Expanding and Hiding Elements..... 145

Hide Connected Incoming Ports

Access

You can access this command via:

Ribbon	None
Context menu of	None

Shortcut key	None
Icon	

Purpose

To hide the connected incoming ports in the Port Topology pane.

Description

If your simulation system contains a lot of connected ports, it can be helpful to hide them for a better overview.

Result

The connected incoming ports are hidden in the tree view and changes to .

Related topics

References

Hide Connected Outgoing Ports..... 151

Hide Connected Outgoing Ports

Access

You can access this command via:

Ribbon	None
Context menu of	None
Shortcut key	None
Icon	

Purpose

To hide the connected outgoing ports in the Port Topology pane.

Description

If your simulation system contains a lot of connected ports, it can be helpful to hide them for a better overview.

Result

The connected outgoing ports are hidden in the tree view and changes to .

Related topics

References

Hide Connected Incoming Ports..... 150

Import

Access

You can access this command via:

Ribbon	Home - Home
Context menu of	None
Shortcut key	`Ctrl` + `I`
Icon	

Purpose

To import model implementation container files, e.g., V-ECU implementations and Simulink implementation containers, and VPUs into a simulation system.

Description

The VEOS Player opens the recently imported files list if you click the arrow part of the Import button. If you click the main icon part, the VEOS Player opens the Select File for Import dialog.

In both cases you can select a file of one of the following groups of file formats:

Import File Format	Description
Container File Formats	
BSC	When you import a BSC file, the VEOS Player builds the implementation of the bus configuration. Then the resulting VPU is imported to the simulation system.
VECU	When you import a VECU file, the VEOS Player builds the V-ECU implementation (for a classic V-ECU), or imports it (for an adaptive V-ECU). The resulting VPU then is imported to the simulation system.

Import File Format	Description
FMU	When you import an FMU file, the VEOS Player builds the functional mock-up unit. The resulting VPU then is imported to the simulation system. **Note** ▪ VEOS supports Functional Mock-up Units (FMUs) that comply with the FMI 2.0 standard. ▪ VEOS supports only the *FMI for Co-Simulation interface*, but not the FMI for Model Exchange interface. You can integrate an FMU file in a VEOS simulation on Windows and in a VEOS simulation on Linux. For detailed and up-to-date compatibility information on dSPACE FMI support, refer to: http://www.dspace.com/go/FMI-Compatibility.
SIC	When you import an SIC file, the VEOS Player builds the Simulink implementation. The resulting VPU then is imported to the simulation system.
SMC	When you import an SMC file, the VEOS Player imports each source container within the SMC container and connects their ports and bus connectors according to the connections specified by the system model. The resulting VPUs then are imported to the simulation system.
OSA File Format	
OSA	When you import an OSA file, the VEOS Player lets you select the VPU(s) to be imported to the simulation system.

Import dialog After you select a file to be imported, the VEOS Player opens the Import dialog. Depending on the format of the file to be imported, you can specify import settings in this dialog.

The dialog provides the following controls and pages:

- File path control (Import dialog) on page 154
- Application Options page (Import dialog) on page 154
- Build Options page (Import dialog) on page 154
- Code Files page (Import dialog) on page 155
- Hypervisor Options page (Import dialog) on page 155
- XCP Service page (Import dialog) on page 157
- Build/Start buttons on page 157

Updating and replacing VPUs VEOS lets you update and replace a VPU while keeping existing signal connections if possible.

Refer to:

- Replace on page 161
- Update on page 166

File path control (Import dialog)

File path Displays the name and path of the file to be imported. To select another file to be imported, click the Browse button.

Application Options page (Import dialog)

(Available for the following file formats: FMU, OSA)

The dialog lets you select one or more VPUs contained in the specified OSA file.

Step size (Available for the FMU file format)

Lets you specify the simulation step size in seconds.

VPU list (Available for the OSA file format)

Lets you select the VPU(s) to be imported to the open simulation system. The VPUs in the list that are to be updated by the import are marked.

Build Options page (Import dialog)

(Available for the following file formats: BSC, VECU, FMU, SIC, SMC)

Simulation target Lets you specify the simulation target for the build process. The default simulation target is preselected. To select a different simulation target, click the Browse button to open the **Simulation Target Manager** on page 163.

C Compiler options Lets you specify options for compiling C code in the model implementation to be imported. For a list of supported compiler options, refer to the compiler documentation.

For more information on compiler options, refer to Compiler Options on page 166.

C++ Compiler options Lets you specify options for compiling C++ code in the model implementation to be imported. For a list of supported compiler options, refer to the compiler documentation.

For more information on compiler options, refer to Compiler Options on page 166.

Enable debug Lets you enable source code debugging.

When source code debugging is enabled, debug symbol files are added to the generated VPU. As a result, you can debug the source code during offline simulation.

Note

- Do not enable source code debugging if you want to preserve the software's intellectual property. Disabling source code debugging prevents debug symbol files from being added to the generated VPU.
- Do not enable source code debugging in connection with the MSVC compiler if the model implementation to be imported contains third-party libraries that were built without debug information.

For more details on source code debugging, refer to Basics on Debugging Source Code in an Offline Simulation on page 202.

Show linker warnings Lets you specify when to display linker warnings:

- If enabled, linker warnings are displayed even if the build process is successful.
- If disabled, linker warnings are displayed only if the build process is not successful.

Build directory Lets you specify the folder the build results are located in.

Note

When using MATLAB and VEOS in parallel, the **Build directory** for the build results of VEOS must not be a folder on the MATLAB search path.

Code Files page (Import dialog)

(Available for the following file formats: VECU, FMU)

Code descriptors list Displays a list of code descriptors used during the build process of the model implementation to be imported.

Using code descriptors allows you to add additional, simulation-specific code files to the build process of a model implementation.

The code files can be source code, object code, or library files, e.g., patches or specific code that is relevant only to the simulation. The code is not part of the imported model implementation and only used for simulation with VEOS.

Add Lets you add a code descriptor to the list. A standard **Open** dialog opens for you to specify the code descriptor to be added.

Remove Lets you delete the selected code descriptor from the list.

Hypervisor Options page (Import dialog)

(Available for VECU files of adaptive V-ECUs)

The hypervisor options let you configure the connection to the hypervisor on which the adaptive V-ECU to be imported is executed.

For more information, refer to Basics on Simulating Adaptive V-ECUs on page 236.

Hypervisor Connection Lets you select the connection type of the hypervisor that hosts the adaptive V-ECU to be imported. The connection type determines if and how the adaptive V-ECU can communicate with network participants outside the simulation.

Note

Only one adaptive V-ECU per simulation can have a hypervisor connection other than Disconnected.

The following connection types and corresponding virtual network adapters are available:

Connection Type	Corresponding Virtual Network Adapter		Description
	VMware	Hyper-V	
Disconnected	–	–	The adaptive V-ECU is not connected to the network resources of the host PC.
HostOnly[1]	VMnet1[2]	vEthernet (dSPACE_VEOS_5.2_Switch_Internal)[3]	The adaptive V-ECU can be accessed by the host PC from a private network configured by the hypervisor. Data packets of the virtual machine remain in the private network.
Bridged[1]	VMnet0[2]	vEthernet (dSPACE_VEOS_5.2_Switch_Internal)[3], [4]	The adaptive V-ECU can be accessed from the host PC's physical network by sharing the Ethernet adapter of the host PC.[5]
NAT[1]	VMnet8[2]	-	The adaptive V-ECU hosted by the hypervisor can be accessed from the host PC's physical network via network address translation (NAT). [5] This means that the virtual machine hosting the V-ECU shares the host PC's IP address and is therefore not seen as a separate entity in the network.

[1] Accessing an adaptive V-ECU from outside the offline simulation influences the simulation state and can change the simulation results. Furthermore, these accesses are not synchronized with the offline simulation.
[2] The virtual network adapter is created when the VMware Workstation Player is installed.
[3] The virtual network adapter is created during the first simulation of an adaptive V-ECU using Hyper-V.
[4] Refer to Enabling bridging on page 240 for instructions on how to enable bridging for Hyper-V.
[5] The virtual Ethernet adapter of the adaptive V-ECU is bridged into the host PC's physical network when you select this connection type.

Hypervisor MAC-Range Lets you specify a range of MAC addresses (first four bytes) that is used for the virtual machine hosted by the hypervisor.

When you select a connection type other than Disconnected, the specified range is used to generate MAC addresses for the hypervisor Ethernet adapter in the local network of the host PC.

Note

The specified MAC range is *not* used for the (virtual) Ethernet adapter of the adaptive V-ECU. The Ethernet adapter of the adaptive V-ECU must be specified in the Target Machine Configuration (`Tmc`) module configuration of the adaptive V-ECU implementation.
For more information, refer to Tmc Module (SystemDesk Manual 🕮).

XCP Service page (Import dialog)

(Available for the VECU file format and the SIC file format if the SIC file to be imported supports A2L file generation)

Assign port automatically Lets you assign the XCP port automatically.

XCP port (available if Assign port automatically is not selected) Lets you specify an XCP port to access the generated VPU via XCP on Ethernet. You can specify a port number in the range 1024 … 65535. It is recommended not to use the range 49152 ... 65535. If you do not specify a port number, a random port number is generated during the build process.

Build/Start buttons

Build (Available for the VECU, FMU, BSC and SIC file formats) Lets you start the build process. The Build log (refer to Build log dialog on page 157) is opened.

Start (Available for the OSA file format) Lets you start the import.

Build log dialog

(Available for the VECU, FMU, BSC and SIC file formats)

Progress bar To show the progress of the build process.

Build log To provide information on the build process, such as compiler and linker messages.

Messages To provide a history of all the errors, warnings, and info messages related to the build process.
The messages are also displayed in VEOS Player's Messages Pane on page 401.

Related topics

Basics

Compatibility of VEOS 5.2 40

HowTos

How to Import Bus Simulation Containers 78
How to Import FMUs 90
How to Import Simulink Implementations 71
How to Import System Model Containers 95

How to Import V-ECU Implementations....67
How to Import VPUs....97

References

Clear Recently Imported Files....140
Simulation Target Manager....163

Info

Access

You can access this command via:

Ribbon	File
Context menu of	None
Shortcut key	None
Icon	

Purpose

To display information about the opened offline simulation application (OSA) file.

General properties

Author Lets you specify the author of the offline simulation application.

Build date Displays the date when the offline simulation application was built.

Description Lets you enter a description for the offline simulation application.

Simulation system name Displays the name of the simulation system referenced by the offline simulation application.

File path Displays the path to the offline simulation application (OSA) file.

Related topics

HowTos

How to Open an OSA File....33

New

Access

You can access this command via:

Ribbon	▪ File ▪ Home - Home
Context menu of	None
Shortcut key	**Ctrl** + **N**
Icon	

Purpose

To create a new OSA file.

Description

A new OSA file is created. The file is not automatically saved. You have to save it manually.

Result

The VEOS Player lets you edit a new empty OSA file.

Related topics

References

Save..... 161
Save As..... 162

Open

Access

You can access this command via:

Ribbon	▪ File ▪ Home - Home
Context menu of	None
Shortcut key	**Ctrl** + **O**
Icon	

Purpose

To open a simulation system from an offline simulation application (OSA) file.

Result

The VEOS Player opens the recently opened files if you click the arrow part of the **Open** button. If you click the main icon part, the VEOS Player opens a standard **Open** dialog for you to select an OSA file.

The selected OSA file is opened in the **VEOS Player**. The VPUs and signals contained in the simulation system are displayed in the **VEOS Player** panes.

Related topics

HowTos

How to Open an OSA File..... 33

Recently Used

Access

You can access this command via:

Ribbon	**File**
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To open one of the most recently opened OSA files in the VEOS Player.

Result

If another file is already opened, it is closed. Then the file you selected is opened.

Tip

You can click Clear Recently Used Files on page 141 to clear the list or remove a single item from the list.

Related topics

References

Clear Recently Used Files..... 141

Replace

Access

You can access this command via:

Ribbon	None
Context menu of	VPU (**Port Topology** pane)
Shortcut key	None
Icon	None

Purpose

To replace the model container file of the selected VPU.

Description

The VEOS Player opens the **Replace** dialog for you to select a model container file. You can select a model container file with the same or a different file type than the model container file to be replaced.

When you replace a model container file, e.g., a V-ECU implementation or a Simulink implementation container, existing signal connections are kept, if possible. The VPU name is also kept.

The **Replace** dialog provides the same dialog pages as the **Import** dialog. For more information on the dialog settings, refer to Import on page 152.

Related topics

References

Import..... 152
Update..... 166

Save

Access

You can access this command via:

Ribbon	▪ **File** ▪ **Home - Home**
Context menu of	None
Shortcut key	**Ctrl + S**
Icon	

Purpose

To save the open simulation system as an offline simulation application (OSA) file.

Result

The currently opened offline simulation application (OSA) file is saved to the file system.

Depending on the **Export the Variable Description Files (A2L or TRC) when the OSA File is Saved** setting, the variable description files are also exported.

Related topics

References

Export the Variable Description Files (A2L or TRC) when the OSA File is Saved..... 146

Save As

Access

You can access this command via:

Ribbon	**File**
Context menu of	None
Shortcut key	**F12**
Icon	None

Purpose

To save the open simulation system as an offline simulation application (OSA) file under a new name.

Result

A standard **Save As** dialog is opened for you to save the currently opened offline simulation application (OSA) file under a new name.

Related topics

References

Save..... 161

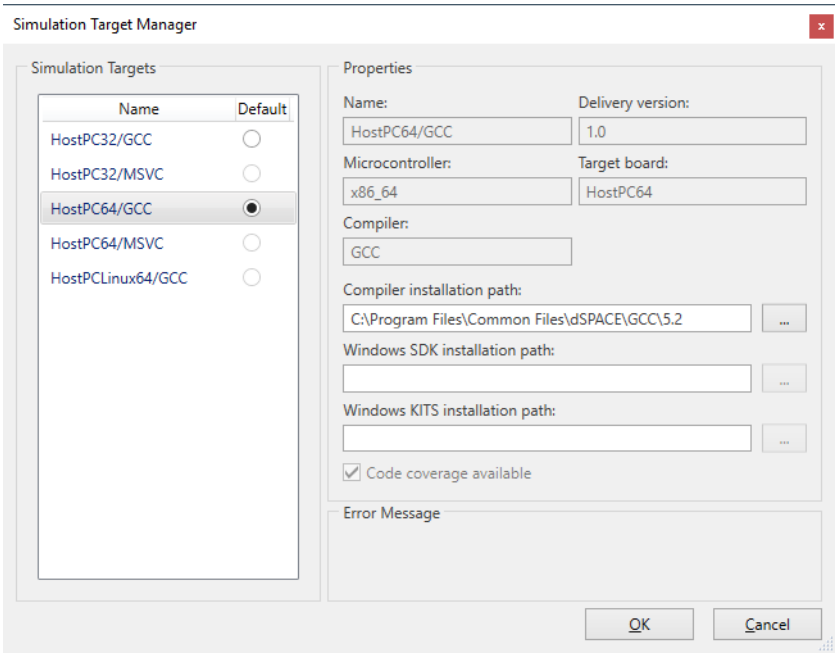
Simulation Target Manager

Access

You can access the **Simulation Target Manager** as follows:

- Via the **Open Simulation Target Manager** control of the **Preferences** page. Refer to Preferences on page 408.
- Via the `VeosSimulationTargetManager.exe` file located in the `\Bin` subfolder of the VEOS installation folder.
- Via the **Simulation target** browse button on the **Build Options** page of the **Import** dialog.

The illustration below shows the **Simulation Target Manager**:



Purpose

To specify simulation targets and make them available to the VEOS build process.

Description

VEOS supports the following simulation targets:

Simulation Target		Abbreviation	Version(s)
Target	**Compiler**		
HostPC32	GNU GCC C Compiler	HostPC32/GCC	5.2[1)]
HostPC32	Microsoft Visual C/C++ Compiler	HostPC32/MSVC[2)]	14.1x[3)]
HostPC64	GNU GCC C Compiler	HostPC64/GCC[4)]	5.2[1)]

Simulation Target		Abbreviation	Version(s)
Target	**Compiler**		
HostPC64	Microsoft Visual C/C++ Compiler	HostPC64/MSVC[2]	14.1x[3]
HostPCLinux64[5]	GNU GCC C Compiler (cross compiler)	HostPCLinux64/GCC	5.5[1]

[1] Provided by VEOS 5.2.
[2] For limitations in connection with the MSVC compiler, refer to Limitations for Integrating the Simulation System on page 178.
[3] Provided by Microsoft Visual Studio 2017. A version of the Windows 10 SDK is required.
[4] When you import and build a model implementation with the HostPC64/GCC simulation target, the build process might fail. For troubleshooting information, refer to Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target on page 176.
[5] To build an offline simulation application for VEOS on Linux. For more information, refer to How to Build an Offline Simulation Application for Linux with the VEOS Player on page 298.

Dialog settings

List of simulation targets Displays the simulation targets that you can select for the build process.

Add (available from the context menu of the list of simulation targets) Lets you add a simulation target to the list of simulation targets for the build process. The Simulation Target Manager opens the Select TSMZ file dialog to select a TSMZ file.

Remove (available from the context menu of the list of simulation targets) Lets you remove the selected simulation target from the list of simulation targets.

Set as default (available from the context menu of the list of simulation targets) Lets you specify the selected simulation target as the *default simulation target*.

Name Displays the target board and compiler abbreviation of the selected simulation target, e.g., HostPC32/GCC.

Delivery version Displays the delivery version of the selected simulation target.

Microcontroller Displays the microcontroller supported by the selected simulation target, e.g., i86.

Target board Displays the target board supported by the selected simulation target, e.g., HostPC32.

Compiler Displays the compiler abbreviation of the selected simulation target, e.g., GCC.

Compiler installation path Lets you specify the folder where the compiler of the selected simulation target is installed.

For MSVC compilers, this Browse button opens the **Choose Compiler Path dialog** dialog.

Windows SDK installation path Lets you specify the folder where the SDK is installed. Specifying the SDK installation path is required by some compilers. For MSVC compilers, the SDK installation path is displayed only.

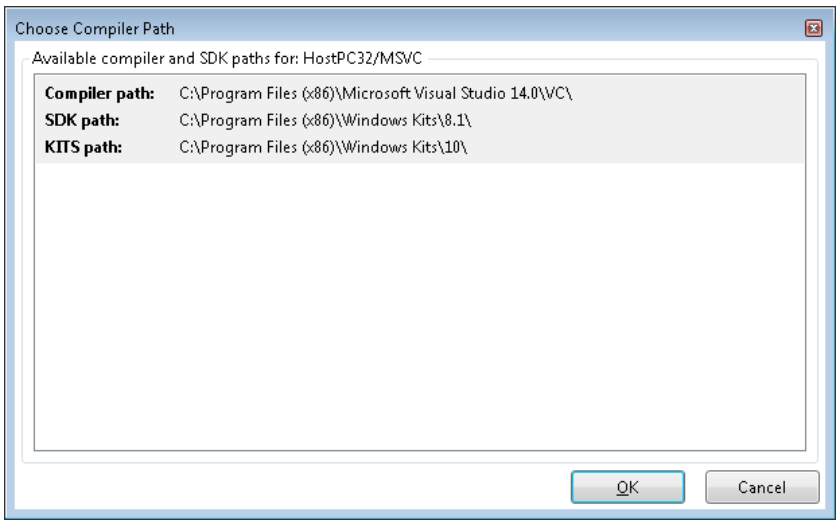
Windows KITS installation path Displays the KITS installation path if **MSVC** is selected as the compiler.

Code coverage available Displays if code coverage analysis of the C code you want to build is enabled.

Code coverage analysis is performed by interfacing with the *Test Coverage Analyzer CTC++* from Testwell. The resulting code coverage report can be displayed via an HTML file.

Choose Compiler Path dialog

For the MSVC compilers installed on the PC, this dialog lets you select a compiler installation path (including the related Windows SDK installation path).



The dialog lists only the MSVC compilers that are registered in the Windows registry.

Related topics

References

Import..... 152

Update

Access

You can access this command via:

Ribbon	None
Context menu of	VPU (**Port Topology** pane)
Shortcut key	None
Icon	None

Purpose

To update the model container file of the selected VPU.

Description

Reimports the model container file of the selected VPU using the path of the initially imported model container file.

When you update a model container file, e.g., a V-ECU implementation or a Simulink implementation container, existing signal connections are kept, if possible.

Related topics

References

Import..... 152
Replace..... 161

Compiler Options

Compiler Options

Disabling warnings

In certain situations, it is useful to disable the warnings during the VEOS build process.

- To disable warnings in connection with the GNU GCC C Compiler, enter `-w`.
- Disabling warnings in connection with the Microsoft Visual C/C++ Compiler is not possible.

Specifying the optimization level

Do not specify optimization levels higher than the default.

Do not specify compiler options that have an effect on data widths and memory alignments.

For example, do not use the following options:

- GNU GCC C Compiler:
  - `-fpack-struct`
  - `-fshort-enums`
  - Do not use the `__attribute__ ((__packed__))` directive for struct and enum definitions within your source code for variables that will be included in the variable description file.

Deactivating the VEOS data type check

During the download of an offline simulation application, VEOS automatically checks the data type width of enumerations and pointers. VEOS expects a data type width of 4 bytes. If it detects another width, VEOS issues an error message and stops the download of the offline simulation application.

VEOS performs this check:

- Independent of the container format
- Independent of the compiler

You can deactivate the check of the data type width and prevent VEOS from stopping the application download by specifying the following preprocessor defines:

Define	Description
`VEOS_NO_DTSIZE_CHECK`	To deactivate the check for enumerations and pointers.
`VEOS_NO_DTSIZE_CHECK_ENUM`	To deactivate the check for enumerations only.
`VEOS_NO_DTSIZE_CHECK_PTR`	To deactivate the check for pointers only.

Note

Keep in mind that deactivating the check of the data type width is useful only if enumerations and pointers are not part of the application's variable description.

FMU-specific compiler options

Enabling FMU-internal message logging To enable logging of FMU-internal messages, enter one of the following compiler options:

- Microsoft Visual C/C++ Compiler: `/DDSFMULogging=1`
- GNU GCC C Compiler: `-DDSFMULogging=1`

Enabling FMI-compliant simulation behavior You can enable FMI-compliant simulation behavior. For more information, refer to Specifying the FMU Simulation Behavior on page 87.

To enable FMI-compliant simulation behavior, enter one of the following compiler options:

- Microsoft Visual C/C++ Compiler: `/DDSFMUComplianceMode=1`
- GNU GCC C Compiler: `-DDSFMUComplianceMode=1`

Related topics

Basics

Basics on Debugging Source Code in an Offline Simulation..... 202
Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target..... 176

References

Import..... 152
Simulation Target Manager..... 163

Automation

Where to go from here

Information in this section

Automating the Opening and Saving of an Offline Simulation Application 169
You can automate opening an offline simulation, and saving an offline simulation.

Automating the Import and Removal of VPUs 170
You can automate the import of VPUs and model implementations such as a V-ECU implementations, and the removal of VPUs.

Automating the Connection of VPUs 172
You can automate the connection of VPUs.

Automating the Opening and Saving of an Offline Simulation Application

Introduction

You can automate:

- Opening an offline simulation
- Saving an offline simulation

The program listings below consist of excerpts from the Python demo scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: [...].

Opening an offline simulation application (OSA) file

The following listing shows how you can open the `AR_PosControl_2Ecus_Bus.osa` offline simulation application file in the VEOS Player.

```
def OpenOfflineSimulationApplication(player):
    [...]
    demoFile = os.path.join(demoDirectory, "AR_PosControl_2Ecus_Bus.osa")
    application = player.Projects.Open(demoFile)
    return application
```

Saving the offline simulation application

The following listing shows how you can save the offline simulation application to the `ThrottleControl_AfterDemoRun.osa` file.

```
def SaveOfflineSimulationApplication(player, application):
  [...]
  demoFile = os.path.join(demoDirectory, "ThrottleControl_AfterDemoRun.osa")
  application.Save(demoFile)
```

Related topics

HowTos

How to Open an OSA File 33

References

IApplication 343
IProjects 363

Automating the Import and Removal of VPUs

Introduction

You can automate:

- The import of VPUs
- The import of a model implementation such as a V-ECU implementation
- The removal of VPUs

The program listings below consist of excerpts from the Python demo scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: [...].

Importing VPUs

Importing all the VPUs from an OSA file The following listing shows how you can import all the VPUs from the `ImportVPUs.osa` file to the offline simulation application loaded in the VEOS Player.

```
def ImportVpus(application, importConnectedChannels):
    importFile = os.path.join(demoDirectory, "ImportVPUs.osa")
    application.ImportVpus(importFile, importConnectedChannels)
```

Importing a specific VPU from an OSA file The following listing shows how you can import a list of VPUs from the `ImportVPUs.osa` file to the offline simulation application loaded in the VEOS Player.

```
def ImportVpusByName(application, list, importConnectedChannels):
    importFile = os.path.join(demoDirectory, "ImportVPUs.osa")
    for vpuName in list:
        application.ImportNamedVpu(importFile, vpuName, importConnectedChannels)
```

Importing V-ECU implementations

The following listing shows how you can import a V-ECU implementation contained in a VECU file to the offline simulation application loaded in the VEOS Player.

```
def ImportVEcuImplementation(application):
    print("Import VECU compiled with HostPC32/GCC and XCP Ports 30000 and 30001")
    importSettings = application.CreateNewImportSettings()
    importSettings.ImportFilePath = GetSourceArchive()
    importSettings.BoardName= "HostPC32"
    importSettings.CompilerName = "GCC"
    importSettings.CompilerOptions = ""
    importSettings.EnableLogging = False
    importSettings.XcpPort = "30000"
    application.Import(importSettings)
```

Importing Simulink implementations

The following listing shows how you can import a Simulink implementation contained in an SIC file to the offline simulation application loaded in the VEOS Player.

```
def ImportSic(application):
    print("Import SIC compiled with HostPC32/GCC")
    importSettings = application.CreateNewImportSettings()
    importSettings.ImportFilePath = GetSourceArchive()
    importSettings.BoardName= "HostPC32"
    importSettings.CompilerName = "GCC"
    importSettings.CompilerOptions = ""
    application.Import(importSettings)
```

Importing FMUs

The following listing shows how you can import a Functional Mock-up Unit contained in an FMU file to the offline simulation application loaded in the VEOS Player.

```
def ImportFmu(application):
    print("Import FMU compiled with HostPC32/GCC")
    importSettings = application.CreateNewImportSettings()
    importSettings.ImportFilePath = GetSourceArchive()
    importSettings.BoardName= "HostPC32"
    importSettings.CompilerName = "GCC"
    importSettings.CompilerOptions = ""
    application.Import(importSettings)
```

Removing VPUs

The following listing shows how you can remove a VPU from an offline simulation application.

```
def RemoveVpu(application, vpuName):
    application.RemoveVpu(vpuName)
```

Related topics

HowTos

How to Import VPUs.....97

References

IProject.....359

Automating the Connection of VPUs

Introduction

You can automate the connection of VPUs.

The program listings below consist of excerpts from the Python demo scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: [...].

Auto-connecting VPU ports

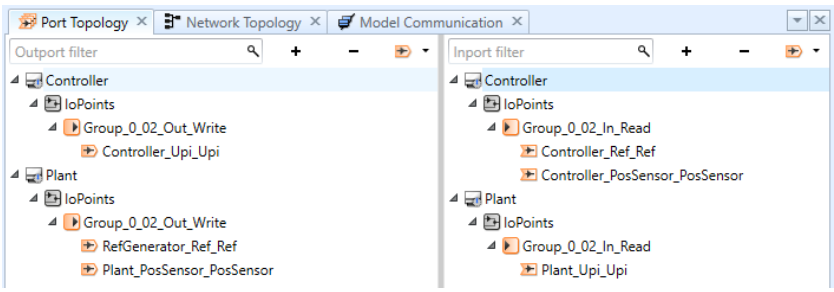
The following listing shows how you can connect all outgoing and incoming VPU ports with identical names in the offline simulation application loaded in the VEOS Player.

```
def AutoConnectVpus(application):
    application.ConnectUnconnectedPortsByName()
```

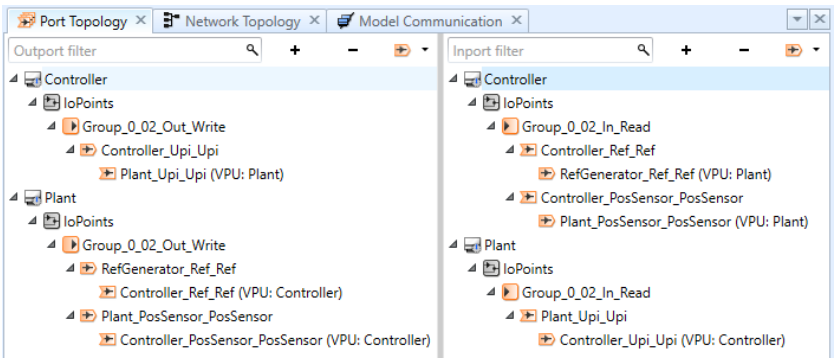
For more information on the auto-connect feature, refer to Ways to Connect VPU Signals on page 102.

Example use case

The following illustration shows an example use case with three signal pairs to be connected:



The following illustration shows the application after the connections have been established:



The following listings show you how to establish these connections in two ways:

- Connecting VPU ports by using a connection list on page 173
- Connecting VPU ports (without a connection list) on page 173

Connecting VPU ports by using a connection list

The following listing shows how to create a connection list with signals to be connected.

> **Tip**
>
> This method is useful for connecting a large number of signals.

Creating a connection list Use the `CreateNewConnectionList` method in the `IProject` interface to create a connection list:

```
connectionList = application.CreateNewConnectionList()
```

Adding connections to the list Use the `AddConnection` method and/or the `AddVpuConnection` method in the `IConnectionList` interface to add the signals to be connected to the connection list:

```
connectionList.AddConnection("Plant", "/[...]/IoPoints/Group_0_02_Out/Group_0_02_Out_Write/RefGenerator_Ref_Ref",
  "Controller", "/[...]/IoPoints/Group_0_02_In/Group_0_02_In_Read/Controller_Ref_Ref")
connectionList.AddConnection("Plant", "/[...]/IoPoints/Group_0_02_Out/Group_0_02_Out_Write/Plant_PosSensor_PosSensor",
  "Controller", "/[...]/IoPoints/Group_0_02_In/Group_0_02_In_Read/Controller_PosSensor_PosSensor")

plant = GetVpu(application, "Plant")
plantPort = controller.GetVpuPortByOrigin("/[...]/IoPoints/Group_0_02_In/Group_0_02_In_Read/Plant_Upi_Upi")
controller = GetVpu(application, "Controller")
controllerPort = controller.GetVpuPortByOrigin("/[...]/IoPoints/Group_0_02_Out/Group_0_02_Out_Write/Controller_Upi_Upi")

connectionList.AddVpuConnection(controllerPort, plantPort)
```

Connecting signals Use the `ConnectVpuPortList` method in the `IConnectionResult` interface to connect the signals in the connection list:

```
result = application.ConnectVpuPortList(connectionList)
```

Connecting VPU ports (without a connection list)

The following listing shows how you can connect signals without using a connection list.

> **Note**
>
> This method is not useful for connecting a large number of signals.

Getting VPUs The following listing shows how you can get VPUs in the offline simulation application loaded in the VEOS Player.

```
def GetVpu(application, vpuName):
    value = application.Item(vpuName)
    if (not value):
        raise Exception("No VPU is named like '%s'" % vpuName)
    return value
```

Getting VPU ports The `VpuPorts` property in the `IVpu` interface lets you access only the VPU ports that are at the root level of a VPU. Accessing VPU ports at deeper levels is not possible using this approach.

Instead, you can use one of the following approaches:

- The `IVpuPorts` and the `IVpuPortGroups` interfaces provide an `Item` method that lets you return the underlying `IVpuPort` and `IVpuPortGroup` interfaces.

- As an alternative, the `IVpu` and the `IVpuPortGroup` interfaces provide an `Item` method that lets you return underlying generic `IVpuItem` interfaces.

  Via the `VpuItemType` property in the `IVpuItem` interface, you can get the item type, either a VPU port or a VPU port group.

The following listing shows how you can get VPU ports in the offline simulation application loaded in the VEOS Player.

```
def GetVpuPort(vpu, vpuPortGroupArray, vpuPortName):
    vpuPortParentGroup = vpu
    for vpuPortGroup in vpuPortGroupArray:
        vpuPortParentGroup = vpuPortParentGroup.Item(vpuPortGroup)
    value = vpuPortParentGroup.Item(vpuPortName)
    if (not value):
        raise Exception("No VPU port in VPU '%s' is named like '%s' and has the group structure '%s'" % (vpu.Name,
vpuPortName, vpuPortGroupArray))
    return value
```

Connecting VPU ports The following listing shows how to connect VPU ports.

```
def ConnectVpuPorts(sourceVpu, sourceGroupArray, sourcePortName, destVpu, destGroupArray, destPortName):
    sourcePort = GetVpuPort(sourceVpu, sourceGroupArray, sourcePortName)
    destinationPort = GetVpuPort(destVpu, destGroupArray, destPortName)
    sourcePort.AddConnection(destinationPort)
def ConnectVpus(application):
    controller = GetVpu(application, "Controller")
    plant = GetVpu(application, "Plant")
    ConnectVpuPorts(
        controller, ["IoPoints", "Group_0_02_Out_Write"], "Controller_Upi_Upi",
        plant,      ["IoPoints", "Group_0_02_In_Read"], "Plant_Upi_Upi")
    ConnectVpuPorts(
        plant,      ["IoPoints", "Group_0_02_Out_Write"], "RefGenerator_Ref_Ref",
        controller, ["IoPoints", "Group_0_02_In_Read"], "Controller_Ref_Ref")
    ConnectVpuPorts(
        plant,      ["IoPoints", "Group_0_02_Out_Write"], "Plant_PosSensor_PosSensor",
        controller, ["IoPoints", "Group_0_02_In_Read"], "Controller_PosSensor_PosSensor")
```

Specifying blocking or non-blocking VPU port communication

To specify blocking or non-blocking VPU port communication, you have to add communication packages to the offline simulation application. Then, you can assign VPU port connections to the packages.

Before you can perform this step, VPU port connections must be established. In the following listings, the VPU port connections established in the previous step (refer to Connecting VPU ports on page 174) are used as an example.

Adding communication packages The following listing shows how you can add a communication package with the non-blocking communication type to the application. The package is named *Non-blocking package*.

```
application.CommunicationPackages.Add(0, "Non-blocking package")
```

The following listing shows how you can add a communication package with the blocking communication type to the application. The package is named *Blocking package*.

```
application.CommunicationPackages.Add(1, "Blocking package")
```

Assigning VPU port connections to packages The following listing shows how you can assign all the VPU port connections of the *IoPoints/Group_0_02_In_Read* plant VPU port group to the *Non-blocking package*.

```
application.ConnectVpuPortGroupToCommunicationPackage(
                            "Plant",
                            "IoPoints/Group_0_02_In_Read",
                            "Non-blocking package")
```

The following listing shows how you can assign all the VPU port connections of the *IoPoints/Group_0_02_In_Read* controller VPU port group to the *Blocking package*.

```
application.ConnectVpuPortGroupToCommunicationPackage(
                            "Controller",
                            "IoPoints/Group_0_02_In_Read",
                            "Blocking package")
```

Disconnecting VPU ports

The following listing shows how you can remove all the VPU port connections in an offline simulation application loaded in the VEOS Player.

```
def DisconnectAllVpuPorts(application):
    for vpu in application.Vpus.Elements:
        vpu.RemoveAllVpuPortConnections()
```

Related topics

Basics

Basics on Signal Connection Dependencies..... 100
Ways to Connect VPU Signals..... 102

References

ICommunicationPackages..... 351
IConnectionList..... 353
IConnectionResult..... 353
IProject..... 359
IVpu..... 368
IVpuItem..... 370
IVpuPort..... 372
IVpuPortGroup..... 374
IVpuPortGroups..... 375
IVpuPorts..... 376
VpuItemType..... 379

Troubleshooting

Where to go from here

Information in this section

Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target..... 176
When you import and build a model implementation with the HostPC64/GCC simulation target, the build process might fail.

Problem When Building a Model Implementation with a 64-Bit Simulation Target..... 177
When you import and build a model implementation with a 64-bit simulation target, the simulation might lead to unexpected results or crash.

Problem when Importing and Building a Model Implementation with the HostPC64/GCC Simulation Target

Problem

When you import and build a model implementation with the HostPC64/GCC simulation target, the **Build log** dialog may display an error message similar to the following:

```
             ...<File_name>.o: too many sections
Fatal error: ...<File_name>.o: File too big
```

Solution

Use one of the following simulation targets:

- HostPC32/GCC
- HostPC32/MSVC

- HostPC64/GCC. On the **Build Options** page of the VEOS **Import** dialog, specify the `-mcmodel=small` compiler option.

  Note

  Do not specify the `-mcmodel=small` compiler option to import and build a V-ECU implementation (VECU). Use one of the other simulation targets instead.

- HostPC64/MSVC

Related topics

References

Import 152
Simulation Target Manager 163

Problem When Building a Model Implementation with a 64-Bit Simulation Target

Problem

When you import and build a model implementation with a 64-bit simulation target, the simulation might lead to unexpected results or crash.

Description

Up to and including VEOS 4.5, the simulation of 64-bit applications was limited to the lower 32-bit address range. As of VEOS 5.0, simulations of 64-bit applications use the entire 64-bit address range.

As a result, if the model implementation to build contains code that uses typecasting of memory addresses to 32 bits, the simulation might lead to unexpected results or crash if you build the model implementation with a 64-bit compiler.

Solution

To solve the problem, you have two options:

- Build the model implementation with a 32-bit compiler. This is not supported for Linux operating systems.
- Adapt the code contained in the model implementation to a 64-bit simulation target, then build the model implementation with a 64-bit compiler.

Related topics

References

Simulation Target Manager 163

Limitations

Limitations for Integrating the Simulation System

Limitations for BSC integration

There are some limitations for BSC integration:

- The following limitation applies if the BSC file contains a Simulink implementation container (SIC file):

  VEOS does not support external events. As a consequence, VEOS cannot call runnable functions in a Simulink implementation container if the functions are triggered, for example, by I/O events.

  VEOS Player issues the following warning:

  ```
  No source event is set for task <Task_Name>. It is therefore
  considered to be untriggered.
  ```

  Importing a BSC file containing a Simulink implementation container with such runnable functions to the VEOS Player, however, is possible.
- VEOS does not support the triggering of multiple runnable functions by one event.
- When you import a BSC file to VEOS, initial values for model port blocks that were set during generation in ConfigurationDesk are ignored and set to *0*.
- See Limitation for VPUs built with the MSVC compiler on page 180.
- For more information on Bus Manager limitations, refer to:
  - Limitations for Using the Bus Manager (Bus Manager (Stand-Alone) Implementation Guide 🕮) (Bus Manager (stand-alone))
  - Limitations for Using the Bus Manager (ConfigurationDesk Bus Manager Implementation Guide 🕮) (Bus Manager in ConfigurationDesk)

Limitations for VECU integration

There are some limitations for VECU integration:

- The VEOS simulator does not support all the basic software modules of model-based V-ECUs. For an overview, refer to Basic Software Module Support for Classic V-ECUs (SIL Testing Overview 🕮).
- See Limitation for VPUs built with the MSVC compiler on page 180.

Limitations for FMU integration

There are some limitations for FMU integration:

- VEOS supports Functional Mock-up Units (FMUs) that comply with the FMI 2.0 standard.

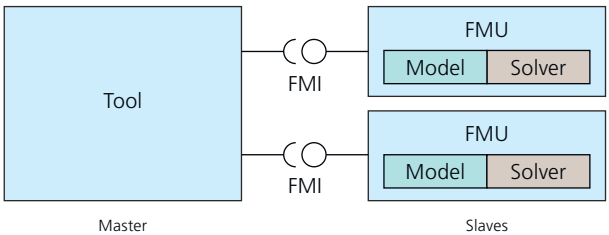
  For detailed and up-to-date compatibility information on dSPACE FMI support, refer to:

  http://www.dspace.com/go/FMI-Compatibility.
- VEOS supports only the *FMI for Co-Simulation interface*, but not the FMI for Model Exchange interface.

- The string data type of FMU inputs and outputs is not supported.
  - VEOS Player does not create VPU ports for these inputs/outputs.
  - VEOS Player does not generate TRC file entries for these inputs/outputs.
- Unit information provided for FMU variables in the `modelDescription.xml` file is ignored.
- The names of the models to be imported via FMU import must be unique in a simulation system. The model name is defined in the `modelDescription.xml` file.

  This limitation applies to the model names, not to the names of the FMU files.
- FMUs requiring additional tools for simulation are not supported.
- VEOS operates FMUs with a *fixed step size*, i.e., the master algorithm used by VEOS does not support variable step sizes for the communication steps regardless of whether the FMU can handle variable communication step sizes or not.

  However, this limitation *does not apply to the internal step size of the solver of an FMU*, i.e., VEOS supports variable step sizes within an FMU.



- Importing FMUs with static libraries is not possible.
- A2L variable description files included in FMUs are ignored during FMU import.
- See Limitation for VPUs built with the MSVC compiler on page 180.

Limitation for SIC integration

There are some limitations for SIC integration:

- You cannot precompile SIC files for the VEOS simulation platform.
- Importing an SIC file *precompiled for the SCALEXIO simulation platform* is possible only if the SIC file also contains the source code files for the related model.
- VEOS does not support external events. As a consequence, VEOS cannot call runnable functions in a Simulink implementation container if the functions are triggered, for example, by I/O events.

  VEOS Player issues the following warning:

  ```
  No source event is set for task <Task_Name>. It is therefore
  considered to be untriggered.
  ```

  Importing a Simulink implementation container with such runnable functions to the VEOS Player, however, is possible.
- See Limitation for VPUs built with the MSVC compiler on page 180.
- When you import an SIC file to VEOS, initial values for model port blocks that were set during generation in ConfigurationDesk or the Model Interface package for Simulink are ignored and set to *0*.

Limited C++ support

VEOS Player lets you import and build model implementations containing C and C++ source code for the PC target.

However, the following limitations apply to C++ support in connection with the MSVC compiler:

- Do not enable Simulink® Coder™'s C++ code generation when you generate an SIC or BSC file. However, you can integrate your C++ code in an S-function.
- The names of variables to be generated in the variable description must comply with the C symbol syntax.

Limitation for VPUs built with the MSVC compiler

If the VPU code contains a function that prints a floating-point variable (`printf("Variable: %f", variable)`) or a function that writes data to a string (`vsprintf("Variable: %f", variable)`), the following error message is issued for that VPU if it was built with the MSVC compiler.

```
Microsoft Visual C++ Debug Library
Debug error! Program: ...\bin\VpuHost.exe R6002 - floating point
support not loaded (Press Retry to debug the application)
```

Workaround

- To avoid the run-time error described above when you import an FMU:
  - Import the FMU as a DLL instead of importing it as source code.
  - Use the FMI-specific methods to print floating-point variables and write data to a string.

    These functions are defined in the Functional Mock-up Interface (FMI) standard.
- To avoid the run-time error described above when you import a V-ECU implementation, you can use the API of the simulator abstraction to display text messages during simulation.

  For additional information on using the Sab API, refer to Creating Code-Based V-ECUs 🕮 which you can find at www.dspace.com/go/CodeBasedVECU.
- There is no workaround to avoid the run-time error described above when you import a Simulink implementation container.

Limited number of VPUs in a simulation system

VEOS supports simulation systems with a maximum of 100 VPUs for which an SDF/TRC variable description is generated during the build process.

This applies to the following model container types:

- Bus simulation container (BSC)
- Functional Mock-up Unit (FMU)
- Simulink implementation container (SIC)

Limited number of signals in a simulation system

VEOS supports VPUs with a maximum of 65,536 (scalar) signals per direction.

HostPC64/GCC simulation target If you use the HostPC64/GCC simulation target, VEOS supports VPUs with a maximum of approx. 16,380 (scalar) signals per direction. When this limit is exceeded, the compiler issues the error message `as.exe: <file name> .o: too many sections`.

As a workaround, use the HostPC32/GCC or the HostPC64/MSVC simulation target.

Related topics

Basics

Limitations for Using the Bus Manager (Bus Manager (Stand-Alone) Implementation Guide 🕮)
Limitations for Using the Bus Manager (ConfigurationDesk Bus Manager Implementation Guide 🕮)

Loading and Running an Offline Simulation Application, and Debugging Source Code

Where to go from here

Information in this section

Basics and Instructions.....184
Reference Information.....212
Automation.....226
Limitations.....232

Basics and Instructions

Where to go from here

Information in this section

Loading and Running an Offline Simulation Application..... 184

Bus Simulation on VEOS..... 196

Debugging Source Code During Offline Simulation..... 201

Loading and Running an Offline Simulation Application

Where to go from here

Information in this section

Simulation on a Local or Remote Simulator..... 185
An offline simulation application can be executed on the *local simulator* or on a *remote simulator*.

How to Select a Remote Simulator..... 186
Before you can simulate an offline simulation application on a remote simulator, you have to select a remote simulator.

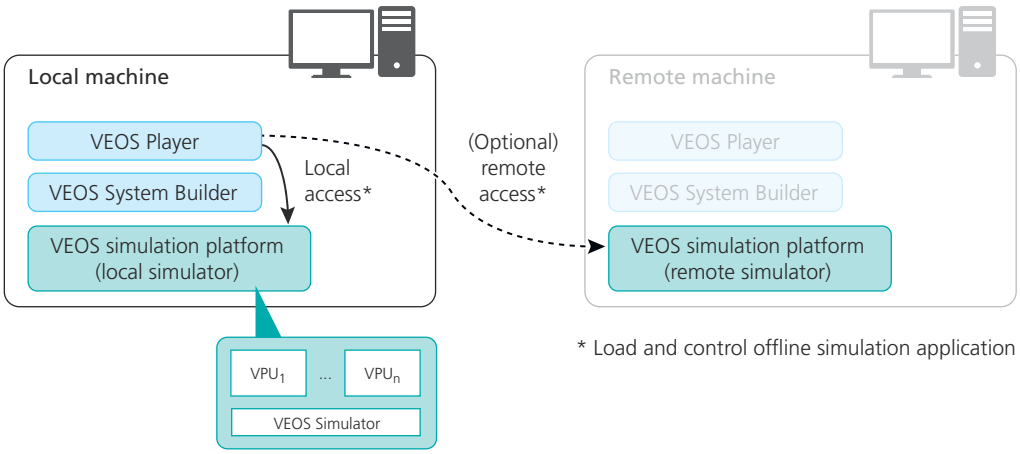
How to Load and Run an Offline Simulation Application..... 188
You can use the VEOS Player to load and run an offline simulation application.

Specific Variables for Simulation..... 189
When you build a simulation system, some simulation-specific variables are generated.

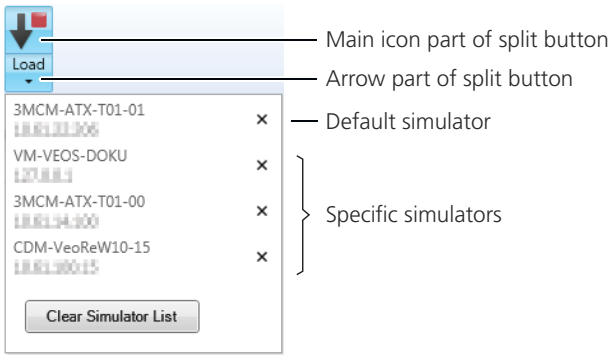
Simulation on a Local or Remote Simulator

Local and remote simulator

An offline simulation application can be executed on the *local simulator* or on a *remote simulator*.



The simulators to which you can load an offline simulation application are available via the **Load** split button in the VEOS Player, as shown in the following illustration.



Default simulator

The first entry in the simulator list is the *default simulator*. When you use the main icon part of the **Load** split button, the offline simulation application is loaded to the default simulator.

When you select a specific simulator in the list, VEOS Player makes this simulator to the default one.

If the simulator list contains no entry, VEOS Player automatically adds the local machine as the default simulator.

Related topics

HowTos

How to Load and Run an Offline Simulation Application..... 188
How to Select a Remote Simulator..... 186

How to Select a Remote Simulator

Objective

Before you can simulate an offline simulation application on a remote simulator, you have to select a remote simulator.

Preconditions

To simulate an offline simulation application on a remote simulator:

- VEOS must be installed on the remote simulator.

  The major and minor VEOS version numbers on the local and the remote simulator must be identical (e.g., VEOS 5.1: 5 = major version number, 1 = minor version number).
- Access to the remote simulator must be enabled.

  In a Command Prompt window on the remote simulator, the following commands must be used:

  1. **`VeosKernel.exe shutdown`**

     To stop the VEOS Simulator.
  2. **`VeosKernel.exe config --enable-remote-access`**

     To enable remote access to the VEOS Simulator.
  3. **`VeosKernel.exe`**

     To start the VEOS Simulator again.

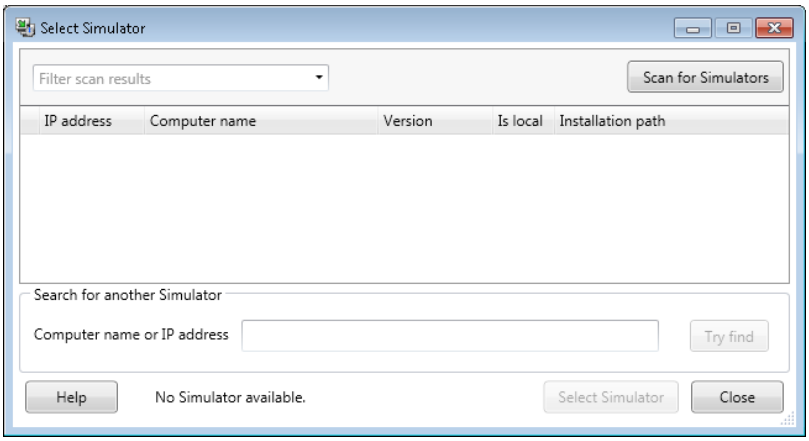
  To disable remote access to the VEOS Simulator later on, use the **`VeosKernel.exe config --disable-remote-access`** command.

Method

To select a remote simulator

1 On VEOS Player's Simulation ribbon on the local simulator, click Select Simulator.

The **Select Simulator** dialog opens.



2 In the **Select Simulator** dialog, select **Scan for Simulators** to scan for simulators.

Tip

If the remote simulator you are searching for is not listed in the simulator list after the scan, enter the name or the address in the **Computer name or IP address** edit field and click **Try find**. If the remote simulator can be found, it is added to the simulator list.

3 In the simulator list, select one remote simulator with the same major and minor VEOS version numbers as the VEOS version of the local simulator.

Click **Select Simulator** to save the selection and close the dialog.

Result

You selected a remote simulator for offline simulation.

Next steps

You can now load and run an offline simulation application to VEOS on a remote simulator. Refer to How to Load and Run an Offline Simulation Application on page 188.

Related topics

Basics

Simulation on a Local or Remote Simulator..... 185

HowTos

How to Load and Run an Offline Simulation Application..... 188

References

Select Simulator..... 219

How to Load and Run an Offline Simulation Application

Objective

You can use the VEOS Player to load and run an offline simulation application.

Preconditions

- An OSA file must be opened in VEOS Player.
  Refer to How to Open an OSA File on page 33.
- If you want to load and run the offline simulation application to a remote simulator, it must be available in the simulator list.
  Refer to How to Select a Remote Simulator on page 186.

Restrictions

VPU variables of a 64-bit application can currently not be measured and calibrated using dSPACE experiment software such as ControlDesk.

Method

To load and run an offline simulation application

1. On the Simulation ribbon, click the arrow on the Load split button.
   - To load the offline simulation application to the default simulator, click the default simulator from the list.
   - To load the offline simulation application to a specific simulator, click a specific simulator from the list.

   The offline simulation application is loaded to the selected simulator.
2. On the Simulation ribbon, click or press F5.

   VEOS Player starts the simulation run.

Result

You loaded an offline simulation application to VEOS. Then you started a simulation run.

If the application is running on a remote simulator, messages that are generated on the remote simulator during simulation are displayed on the local simulator.

In a simulation that contains adaptive V-ECUs, VEOS starts a virtual machine for each adaptive V-ECU. If you use the VMware Workstation hypervisor, a window is automatically opened for each virtual machine. If you work with the Hyper-V hypervisor, you have to use the Hyper-V Manager to display the virtual machines.

Next steps

Controlling the simulation You can control a running simulation using the following VEOS Player commands:

Command	Description
Pause	To pause the simulation running on the PC.
Step	To run simulations stepwise.
Stop	To stop the simulation running on the PC.

Experimenting with the offline simulation application You can use ControlDesk to experiment with an offline simulation application. Refer to ControlDesk (SIL Testing Overview 🕮).

Related topics

Basics

Basics on Simulating Adaptive V-ECUs..... 236
Simulation on a Local or Remote Simulator..... 185

HowTos

How to Select a Remote Simulator..... 186

References

Load..... 216
Pause..... 218
Start..... 221

Specific Variables for Simulation

Introduction

When you build a simulation system, one A2L variable description file is generated for each V-ECU in the system. In addition to A2L variables resulting from the model, the A2L file of a V-ECU also contains specific A2L variables generated for the simulation frame, for example, to simulate a bus breakdown.

To experiment with these variables, you can connect them to instruments in a ControlDesk experiment.

A2L variable for measuring the simulation time

The A2L file of a V-ECU contains a variable for displaying the time since the start of the simulation application.

The table below lists the A2L variable that you can use:

Note

The variables below are available only in an offline simulation.

A2L Variable	Purpose
`/Simulation_and_RTOS/Simulation/SimulationTime`	To display the time since the start of the simulation application.

Variable for monitoring the deviation between real time and virtual simulation time

The A2L file of a V-ECU and - if available - the SDF/TRC file of the related offline simulation application contain a variable for monitoring the deviation between real time and virtual simulation time.

The table below lists the variable that you can use:

Variable	Purpose
`/Diagnostics/ApplicationProcess/Runtime/RealTimeDeviation`	To monitor the deviation between real time and virtual simulation time. The variable value is either 0 or negative since real time is always faster than virtual simulation time.

A2L variables for delaying the startup of a V-ECU

A *real ECU* boots when the power is switched on, and when it receives a reset signal. Some ECUs are reset automatically when the power is switched on (KL 30). Others receive a reset signal when the ignition key is turned on (KL 15).

The A2L file of a V-ECU contains variables that allow you to simulate switching on the power and turning on the ignition key, and thus to delay the V-ECU startup.

The table below lists the A2L variables that you can use:

A2L Variable	Purpose
`/Simulation_and_RTOS/Simulation/Sab_PowerOn`	To enable (1 = ON) or disable (0 = OFF) KL30 (power).
`/Simulation_and_RTOS/Simulation/Sab_Ignition`	To enable (1 = ON) or disable (0 = OFF) KL15 (ignition).

A delayed V-ECU startup means that although the environment model is executed as soon as the simulation application is started, the startup function is not called for one or more V-ECUs until you switch the `Sab_PowerOn` and `Sab_Ignition` A2L variables for them. Each V-ECU's calibration service is active during the delay phase, so that you can change parameter values.

Note

- You can trigger the delayed startup only *once* after you download the simulation application. Once a V-ECU has executed its startup, it cannot be reset and restarted with this mechanism.
- When you start the real-time application, there is a delay between the start of the V-ECU application process(es) and the start of the environment model application process. The latter is started *a few milliseconds after* the V-ECU application process(es).
  This limitation does not apply if you control the V-ECU startup time by setting the `Sab_PowerOn` and `Sab_Ignition` A2L variables.

A2L variables for accessing the ECU State Manager

The A2L file of a V-ECU contains variables that allow you to observe and manipulate the ECU State Manager integrated in the V-ECU.

The table below lists the A2L variables that you can use:

A2L Variable	Purpose
`/BswModules/EcuM/CurrentState`	To display the current state of the ECU State Manager.
`/BswModules/EcuM/NextBootApplicationMode`	To display the application mode that the ECU State Manager will set the next time the ECU boots.
`/BswModules/EcuM/NextMode`	To send a request to the ECU State Manager to switch to the selected state.
`/BswModules/EcuM/NumberRunRequests`	To display the number of run requests to the ECU State Manager.
`/BswModules/EcuM/RunRequests`	To display the users of run requests to the ECU State Manager.

A2L variables for getting information on the OS tasks of a V-ECU

The A2L file of a V-ECU contains variables that provide information on the OS tasks of a V-ECU.

The table below lists the A2L variables that you can use:

A2L Variable	Purpose
`/Simulation_and_RTOS/Task_Information_Variables/<TaskName>/<TaskName>_TaskCallCount`	To display the number of invocations of a task since the start of the simulation application.
`/Simulation_and_RTOS/Task_Information_Variables/<TaskName>/<TaskName>_OverrunCount`	To display the number of task overruns. The number of task overruns is incremented each time the task is activated, although it is already scheduled.
`/Simulation_and_RTOS/Task_Information_Variables/<TaskName>/<TaskName>_TurnaroundTime`	To display the time from task activation (e.g., by `ActivateTask()` or by an alarm or by autostart) to task termination. The time includes the latency before the task enters the `RUNNING` state and the time for preemptions by higher priority tasks.[1)]

[1)] VEOS assumes zero execution time assumption. As a consequence, the value of this variable is always 0 in an offline simulation.

A2L variable for displaying the PDU router initialization state

The A2L file of a V-ECU contains a variable for displaying the initialization state of the PDU router.

The table below list the A2L variable that you can use:

A2L Variable	Purpose
`/BswModules/PduR/PduR_State`	To display the initialization state of the PDU router.

A2L variable for displaying the initialization state of the CanIf and LinIf modules

The A2L file of a V-ECU contains variables for displaying the initialization state of the CanIf/LinIf modules.

The table below lists the A2L variables that you can use:

Note

The variables below are available only in an offline simulation.

A2L Variable	Purpose
`/Diagnostics/State/CanIf_State` **Note** As of dSPACE Release 2021-B, this variable will no longer be supported.	To display the initialization state of the CanIf module.
`/Diagnostics/State/LinIf_State` **Note** As of dSPACE Release 2021-B, this variable will no longer be supported.	To display the initialization state of the LinIf module.

A2L variables for injecting bus failures

VEOS provides a mechanism to manipulate the behavior of a communication bus connecting the members of a simulation system. Manipulating the bus behavior means that you can simulate the breakdown of either the entire bus or single bus controllers, and disable the transmission of a specific IPDU.

Note

Bus behavior manipulation as described in this topic is supported only for offline simulation with VEOS. It is based on the manipulation of specific A2L variables that are part of the variable description of a V-ECU.

If your simulation system also contains a restbus, you can perform restbus simulation and manipulate restbus signals by using specific Bus Instruments of the ControlDesk Bus Navigator. You can then observe the effects of the manipulated bus communication on the ECUs under test.

The A2L file of a V-ECU contains variables for bus failure injection that allow you to manipulate the behavior of a communication bus.

The table below lists the A2L variables that you can use:

Note

The variables below are available only in an offline simulation.

A2L Variable	Purpose
General	
`/Diagnostics/Failure_Simulation/Simulator/BusConnectionState`	To connect (= `0`) or disconnect (= `1`) all the bus controllers of a V-ECU at once.

A2L Variable	Purpose
CAN Bus	
`/Diagnostics/Failure_Simulation/CAN/CAN_Controllers/Can_State_<controller name>` **Note** As of dSPACE Release 2021-B, this variable will no longer be supported.	To turn a specific CAN controller off (= `0`) or on (= `1`).
`/Diagnostics/Failure_Simulation/CAN/CAN_Messages/Can_State_<IPdu name>` **Note** As of dSPACE Release 2021-B, this variable will no longer be supported.	To turn the transmission of a specific CAN IPDU off (= `0`) or on (= `1`).
LIN Bus	
`/Diagnostics/Failure_Simulation/LIN/LIN_Controllers/Lin_State_<controller name>` **Note** As of dSPACE Release 2021-B, this variable will no longer be supported.	To turn a specific LIN controller off (= `0`) or on (= `1`).
`/Diagnostics/Failure_Simulation/LIN/LIN_Messages/Lin_State_<IPdu name>` **Note** As of dSPACE Release 2021-B, this variable will no longer be supported.	To turn the transmission of a specific LIN IPDU off (= `0`) or on (= `1`).

A2L variables for switching LIN schedule tables

The A2L file of a V-ECU configured as a LIN master for a specific channel contains variables that let you switch LIN schedule tables.

The following table lists the A2L variables that you can use:

A2L Variable	Purpose
`/BswModules/dSPACE/NetworkManagement/LIN_StateMgmt/<LinChannel>/NetMgmt_<LinChannel>_ActiveScheduleTable`	To display the currently active LIN schedule table.
`/BswModules/dSPACE/NetworkManagement/LIN_StateMgmt/<LinChannel>/NetMgmt_<LinChannel>_ChannelId`	To display the ID of the selected LIN channel.
`/BswModules/dSPACE/NetworkManagement/LIN_StateMgmt/<LinChannel>/NetMgmt_<LinChannel>_ChannelState`	To display the state of the selected LIN channel.
`/BswModules/dSPACE/NetworkManagement/LIN_StateMgmt/<LinChannel>/NetMgmt_<LinChannel>_SelectedScheduleTable`	To specify the LIN schedule table.

A2L Variable	Purpose
`/BswModules/dSPACE/NetworkManagement/LIN_StateMgmt/<LinChannel>/NetMgmt_<LinChannel>_SetBusAsleep`	To set the LIN bus to sleep mode when the ECU is in sleep mode. **Note** This is currently not supported.
`/BswModules/dSPACE/NetworkManagement/LIN_StateMgmt/<LinChannel>/NetMgmt_<LinChannel>_SwitchScheduleTableOnBusSleep`	To switch to the next LIN schedule table before the LIN bus is set to sleep mode. **Note** This is currently not supported.

A2L variables for manipulating VPU connections

The table below lists the A2L variables that you can use:

Note

The variables below are available only in an offline simulation.

A2L Variable	Purpose
`/Diagnostics/Failure_Simulation/Simulator/VpuConnectionState`	To connect (= `0`) or disconnect (= `1`) all the ports and all the bus controllers of a V-ECU at once.
`/Diagnostics/Failure_Simulation/Simulator/VpuPortsConnectionState`	To connect (= `0`) or disconnect (= `1`) all the ports of a V-ECU at once.

A2L variables for accessing data access points

The A2L file of a V-ECU contains variables that allow you to access data access points integrated in the V-ECU.

The table below lists the A2L variables that you can use:

Note

The variables below are available only in an offline simulation.

A2L Variable	Purpose
`/DataAccessPoints/<DapContext>/<DapBlock>/<DapAccessFunctionName`[1)]`>` ▪ `/DataAccessPoints/<DapContext>/<DapBlock>_TA/<DapAccessFunctionName`[1)]`>_TA_Factor`[2)] ▪ `/DataAccessPoints/<DapContext>/<DapBlock>_TA/<DapAccessFunctionName`[1)]`>_TA_Offset`[2)] ▪ `/DataAccessPoints/<DapContext>/<DapBlock>_TA/<DapAccessFunctionName`[1)]`>_TA_Input`[3)]	To display the value of the DAP signal, which represents a single data access point. To stimulate a data access point of an inport: e.g., when the inport is unconnected (open-loop stimulation). The following formula is applied: `outValue = TA_Input * TA_Factor + TA_Offset` ▪ To disable the inport, set the `TA_Factor` value to '0.0'.

A2L Variable	Purpose
	▪ To use the original DAP signal: i.e., disable stimulation, set the `TA_Factor` value to '1.0' and the `TA_Offset` value to '0.0'. When the simulation starts: ▪ The value of the `TA_Factor` variable is initialized with '1.0'. ▪ The value of the `TA_Offset` variable is initialized with '0.0'. **Note** This applies even if you specify another variable value: e.g., using ControlDesk before starting the simulation.

1) The name of the function used to access the value of the DAP signal.
2) The variable's data type is `Float64`, independent of the DAP signal's data type.
3) The value of this variable represents the original DAP signal value. You can use this variable to compare its value with the stimulated DAP signal value.

A2L variables for accessing RTE interventions V-ECU internally

The A2L file of a V-ECU contains variables that allow you to manipulate RTE interventions from within a V-ECU.

The table below lists the A2L variables that you can use:

A2L Variable	Purpose
`/InterventionService<n>/<name>_InterventionService<n>_input`	To access the input signal of an RTE intervention service.
`/InterventionService<n>/<name>_InterventionService<n>_output`	To access the output signal of an RTE intervention service.
`/RteInterventionServiceEnable<n>/RteInterventionServiceEnable<n>_enable`	To access the input enable signal of one or more RTE intervention services.

A2L variables for displaying NvM information

The A2L file of a V-ECU contains variables for displaying NvM information.

The table below lists the A2L variables that you can use:

A2L Variable	Purpose
`/BswModules/NVRAM/NvmBlock0/NvM_NvBlockData_NvmBlock0`	To display the status of the NvM operations `NvM_ReadALL`/`NvM_WriteALL`.
`/BswModules/NVRAM/NvmConfigId/NvM_NvBlockData_NvmConfigId`	To display the NvM configuration ID.
`/BswModules/NVRAM/NvMBlockDescriptor/NvM_NvBlockData_NvMBlockDescriptor`	To display the NV block content of a specific NvRAM block descriptor.

Related topics

References

Export the Variable Description Files (A2L or TRC) when the OSA File is Saved..... 146

Bus Simulation on VEOS

Where to go from here

Information in this section

Bus Communication Features for SIL Testing..... 196
In offline simulation on VEOS, bus communication is simulated on the PC.

Restbus Simulation in VEOS..... 199
VEOS supports restbus simulation.

Workflow for Simulating Bus Communication..... 199
VEOS supports the offline simulation of bus communication, i.e., communication between the (V-)ECUs connected to a bus.

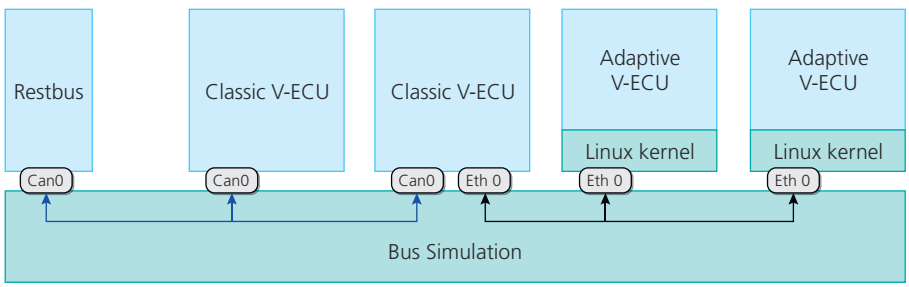
Bus Communication Features for SIL Testing

Introduction

In offline simulation on VEOS, bus communication is simulated on the PC.

Schematic of bus simulation on VEOS

The following illustration shows a schematic of bus simulation on VEOS.



General features of bus communication simulation

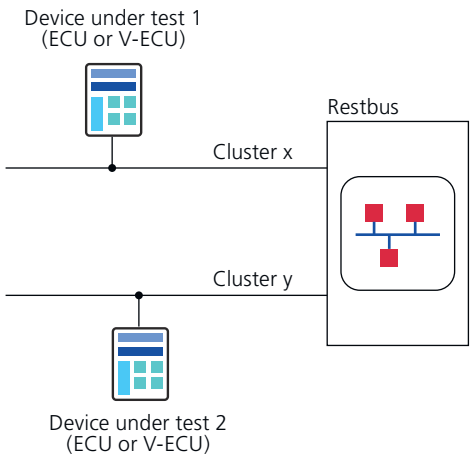
The following general bus communication features are supported independent of the bus type:

- Support of the AUTOSAR COM and PDU Router behavior for any bus type
- Cyclic and event-controlled transmission of IPDUs. Mixed transmission mode is supported for periods greater than zero.
- Delayed start-up of bus communication if V-ECU start-up is delayed.
- Support of restbus simulation

  Restbus simulation lets you test one or more ECUs while simulating the other ECUs of the related communication clusters. The restbus can simulate the bus communication of other ECUs of the simulation system. These restbus ECUs are combined and executed in a single application process, which simplifies the simulation system. The bus communication of the restbus can be manipulated, and the effects on the ECU(s) to be tested can be observed.

  Using the ControlDesk Bus Navigator, the bus communication of the restbus can be manipulated, and the effects on the ECUs to be tested can be observed via bus monitoring. You can also replay logged bus communication for CAN and Ethernet.

  The following illustration shows an example of restbus simulation: The two devices under test are connected to a restbus that simulates the bus communication of other ECUs of the simulation system.



- Global time synchronization support

  The concept of *global time synchronization* was introduced and standardized by AUTOSAR as a means of providing and distributing synchronized times across all ECUs in a vehicle.

Features of CAN bus communication simulation

The following CAN bus communication features are supported:

- Support of *CAN and CAN FD buses*. This includes CAN 2.0 A (characterized by the 11-bit message identifier length) and CAN 2.0 B (characterized by the 29-bit message identifier length).
- Displaying bus statistics during bus monitoring using the ControlDesk Bus Navigator.
- VEOS: Simulation of the *message transmission times*.

- VEOS: Calculation of the bus load during simulation.
- Support of message priorities for arbitrating different messages on a CAN bus.

Features of LIN bus communication simulation

The following LIN bus communication features are supported:

- Compliance with LIN bus transfer times
- Support of LIN message collisions on the bus
- Support of broadcast frames
- Displaying bus statistics during bus monitoring using the ControlDesk Bus Navigator.
- VEOS:
  - Simulation of LIN slave nodes
  - Simulation of multiple LIN master and slave nodes on one V-ECU
  - Simulation of the communication between LIN slave nodes

Features of Ethernet communication simulation

VEOS supports the simulation of Ethernet communication. This lets you simulate communication between classic and adaptive V-ECUs, for example.

The following Ethernet communication features are supported:

- Layer-2 hub-based simulation
- Support of broadcast frames
- Checksum offloading for TCP, UDP, IP, and ICMP

Bus communication log file

You can generate bus communication log files during bus simulation. The files log the entire bus communication and the bus load.

Note

Generating bus communication log files is supported only in an offline simulation on VEOS. Activating the generation of bus communication log files is possible only via the VEOS Player. You cannot activate it via ControlDesk, for example.

Refer to Generate Bus Log Files on page 214.

Related topics

Basics

Limitations (SIL Testing Overview 🕮)
Restbus Simulation in VEOS..... 199
Workflow for Simulating Bus Communication..... 199

References

Basic Software Module Support for Classic V-ECUs (SIL Testing Overview 🕮)

Restbus Simulation in VEOS

Introduction

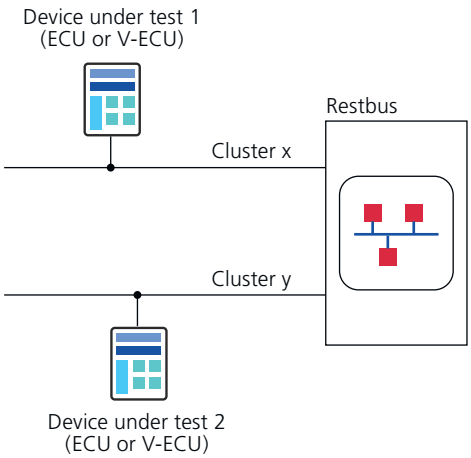
VEOS supports restbus simulation.

Overview

Restbus simulation in VEOS is a use scenario in which only some of the V-ECUs in a communication cluster are to be tested. The *restbus* simulates other V-ECUs of the communication cluster. The restbus is represented by one or more *bus VPUs*.

VEOS supports restbus simulation for CAN and LIN buses.

The following illustration shows an example of restbus simulation: The two devices under test are connected to a restbus that simulates the bus communication of other ECUs of the simulation system.



Related topics

Basics

Bus Communication Features for SIL Testing 196
Workflow for Simulating Bus Communication 199

Workflow for Simulating Bus Communication

Introduction

VEOS supports the offline simulation of bus communication, i.e., communication between the (V-)ECUs connected to a bus.

Preparing bus communication for offline simulation

Bus communication is prepared for offline simulation like this:

Bus communication in V-ECU implementations V-ECU implementation (VECU) files modeled with SystemDesk usually contain bus communication.

For details, refer to Basics on the Network Topology (SystemDesk Manual 🕮).

Bus communication in bus simulation containers Bus simulation container (BSC) files modeled with the Bus Manager contain bus communication.

For details, refer to the Bus Manager documentation.

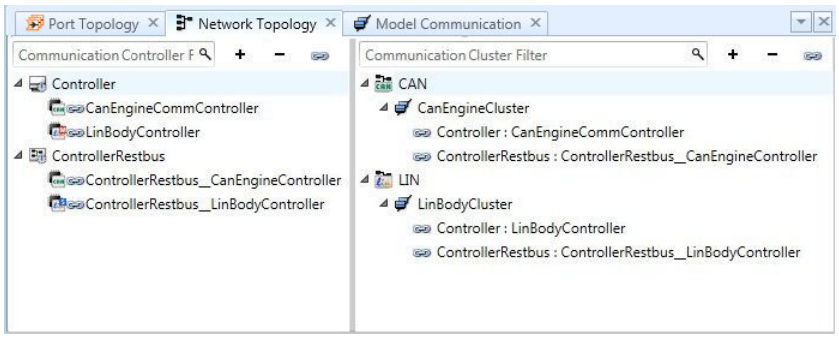
Integrating bus communication in the VEOS Player

In the VEOS Player, you can integrate bus communication for offline simulation by importing VECU and/or BSC files. During import, buses and communication clusters are created and communication controllers of VPUs are connected to communication clusters automatically, if possible. After import, bus communication elements are displayed in the **Network Topology** pane.

Integration for restbus simulation Integrating bus communication from bus simulation containers in the VEOS Player allows you to perform restbus simulation in VEOS.

As an example, the following illustration shows the **Network Topology** pane of an offline simulation application consisting of a V-ECU named **Controller** and a bus VPU named **ControllerRestbus**:

- The CAN controllers of the V-ECU and bus VPU are connected via the CAN communication cluster named **CanEngineCluster**.
- The LIN controllers of the V-ECU and bus VPU are connected via the LIN communication cluster named **LinBodyCluster**.



Simulating bus communication in VEOS

The simulation of the bus communication starts when the related offline simulation application is started.

During bus communication simulation, you can:

- Monitor bus communication via ControlDesk's Bus Monitoring devices
- Handle messages of bus VPUs via ControlDesk's Bus Instruments
- Manipulate the bus behavior via variables in the A2L file of V-ECUs

Monitoring bus communication using ControlDesk's Bus Navigator During offline simulation, you can monitor the entire bus communication by using ControlDesk's Bus Navigator. To do so, you use a bus monitoring device for the specific communication cluster type.

For instructions on configuring bus monitoring devices, refer to:

- How to Configure a CAN Bus Monitoring Device (ControlDesk Platform Management 🕮)

- How to Configure an Ethernet Bus Monitoring Device (ControlDesk Platform Management 🕮)
- How to Configure a LIN Bus Monitoring Device (ControlDesk Platform Management 🕮)

For more information on bus monitoring by using ControlDesk's Bus Navigator, refer to Monitoring and Logging Bus Communication (ControlDesk Bus Navigator 🕮).

Bus VPUs: Manipulating bus messages During offline simulation, you can handle CAN messages and LIN frames of a bus VPU to manipulate restbus communication, for example. Restbus communication is the bus communication of the bus VPU. ControlDesk's Bus Navigator provides specific Bus Instruments for this.

For more information, refer to Working with Bus Instruments (ControlDesk Bus Navigator 🕮).

V-ECUs: Manipulating bus behavior VEOS provides a mechanism to manipulate the behavior of a communication bus connecting the members of a simulation system. A2L file(s) generated for V-ECUs contain specific A2L variables for simulating bus failures. For example, you can disable specific bus controllers or the transmission of specific bus messages during bus simulation.

For details on manipulating bus behavior in an offline simulation via specific A2L variables, refer to Specific Variables for Simulation on page 189.

Related topics

Basics

Bus Communication Features for SIL Testing 196
Restbus Simulation in VEOS 199

Debugging Source Code During Offline Simulation

Where to go from here

Information in this section

Basics on Debugging Source Code in an Offline Simulation 202
Source code debugging in an offline simulation lets you locate problems in ECU code and environment model code.

How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation 205
During offline simulation, you can debug the source code of VPUs built with the MSVC compiler using Microsoft® Visual Studio®.

How to Debug Source Code Built with the GCC Compiler in an Offline Simulation.....209
During offline simulation, you can debug the source code of VPUs built with the GNU GCC C Compiler by using the GNU debugger (GDB).

Basics on Debugging Source Code in an Offline Simulation

Introduction

Source code debugging is useful, for example, to locate problems in:

- ECU software
  - Software component code
  - RTE code
  - Basic software code
- Environment model code
  - Code in S-functions of Simulink models
  - Code in FMU models

You can set breakpoints, inspect variables, and step through the code.

Note

The information in this section does not apply to adaptive V-ECUs. Refer to Basics on Debugging Adaptive V-ECUs on page 244 for information on how to debug adaptive V-ECUs.

Basics

Enabling debugging To debug the source code of a V-ECU, for example, you have to enable source code debugging when you import the related V-ECU implementation to the VEOS Player.

When source code debugging is enabled, debug symbol files are added to the generated VPU. As a result, you can debug the source code during offline simulation.

Note

- Do not enable source code debugging if you want to preserve the software's intellectual property. Disabling source code debugging prevents debug symbol files from being added to the generated VPU.
- Do not enable source code debugging in connection with the MSVC compiler if the model implementation to be imported contains third-party libraries that were built without debug information.

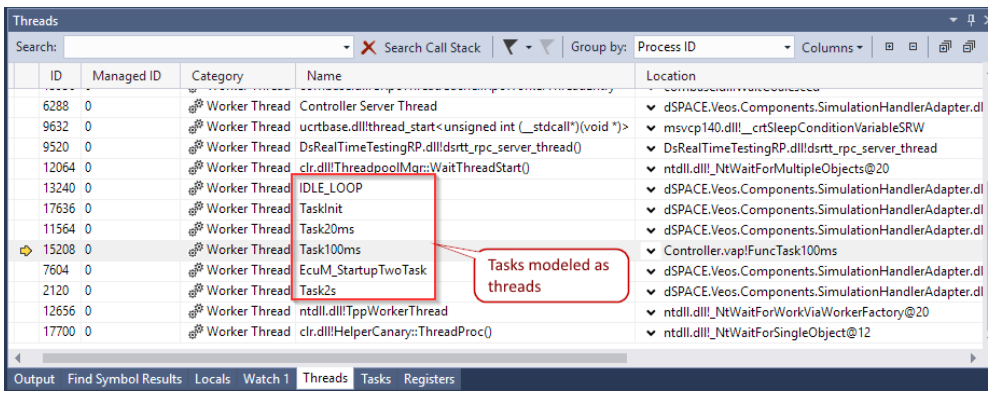
Tip

When you select a VPU in the VEOS Player, the **Properties** pane provides the **Contains debug information** property which displays whether the selected VPU contains debug information. Refer to VPU (V-ECU / Environment VPU / Controller VPU / Bus VPU) on page 131.

- To debug the source code of VPUs built with the MSVC compiler, use Microsoft® Visual Studio®.

  For instructions on debugging source code built with the MSVC compiler, refer to How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation on page 205.

Debugging V-ECUs VEOS assigns a separate thread to each task that belongs to a V-ECU as shown in the following illustration from the Microsoft® Visual Studio® Debugger.



Accessing call stack information in case of an exception When an exception of the model code of a specific VPU process (based on an FMU or SIC) occurs during offline simulation, the following call stack information is issued in an error message:

- The name of the function causing the exception
- The name and location of the source code file containing the function
- The call stack related to the exception

The call stack is provided only if source code debugging was enabled for the VPU's build process, and if the MSVC compiler was used.

Separate VPU processes

Each VPU is executed in a separate process of the PC's operating system. The processes are started when you load an offline simulation application.

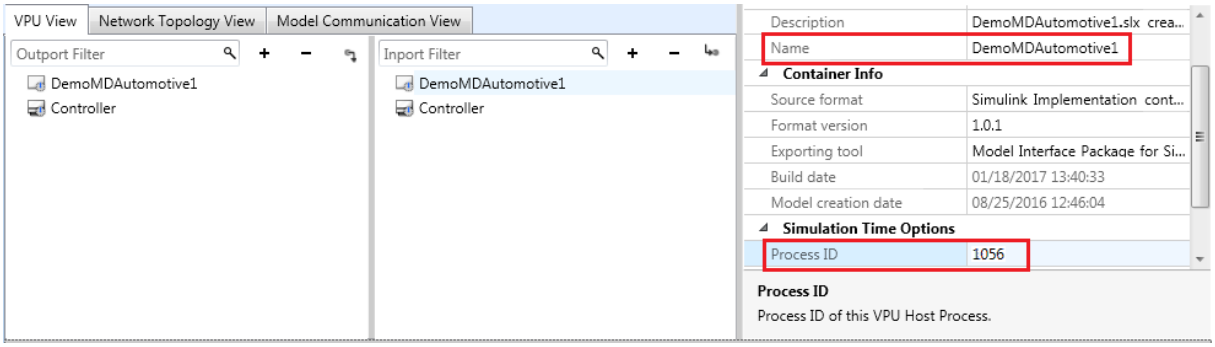
Identifying the VPU process to debug To debug a specific VPU, you have to identify its unique application identifier (ID).

You can identify the application ID in the following ways:

- Via the **Properties** pane of the VEOS Player:

  When you select a VPU in the VEOS Player and if the offline simulation application is loaded to VEOS, the **Properties** pane provides the **Application ID** property together with the **Name** of the VPU.

  The illustration below shows an example VPU process with the application ID **1056** and the VPU name **DemoMDAutomotive1**.

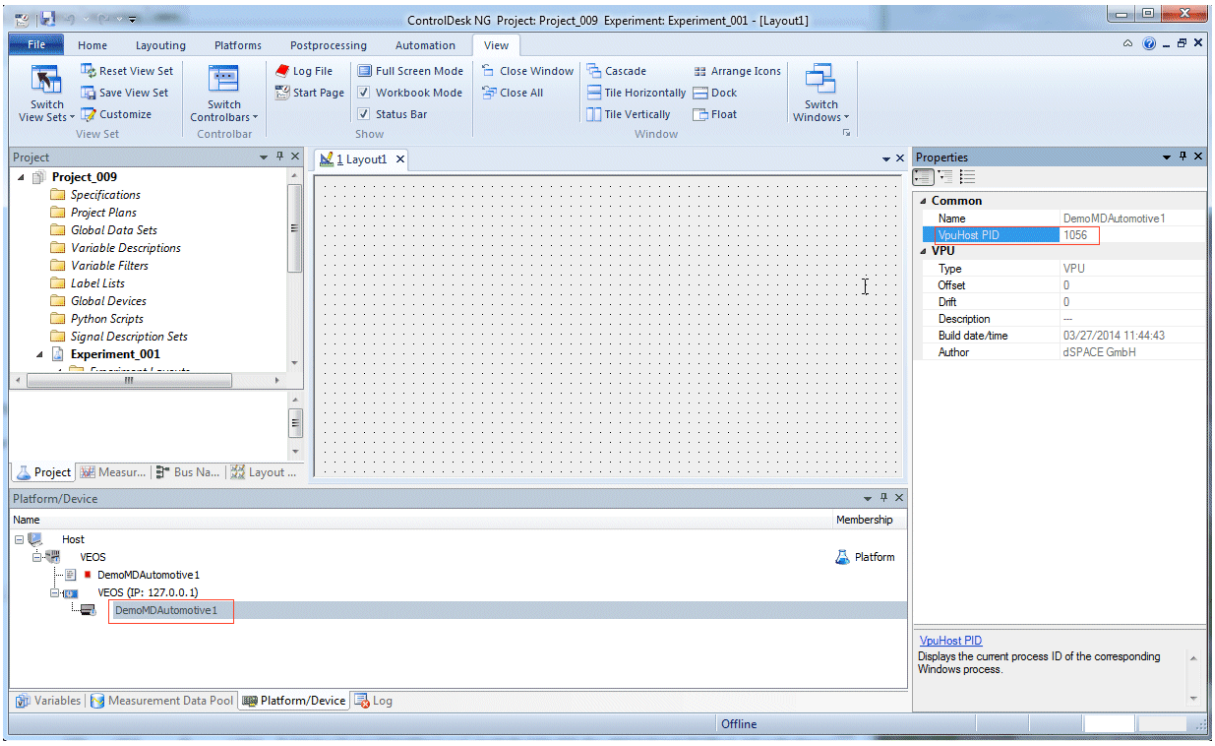


  Refer to VPU (V-ECU / Environment VPU / Controller VPU / Bus VPU) on page 131.

- Via ControlDesk:

  ControlDesk's **Properties** pane displays the application ID of a VPU process.

  The illustration below shows an example VPU process with the application ID (**VpuHost PID**) **1056** and the VPU name **DemoMDAutomotive1**.



Related topics

Basics

Basics on Debugging Adaptive V-ECUs..... 244

HowTos

How to Debug an Adaptive V-ECU..... 245
How to Debug Source Code Built with the GCC Compiler in an Offline Simulation..... 209
How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation..... 205
How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux..... 306

How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation

Objective

During offline simulation, you can debug the source code of VPUs built with the *MSVC* compiler (Microsoft Visual C/C++ compiler).

Preconditions

- The VPUs to be debugged must be built with the MSVC compiler and integrated into a simulation system.

  For information on the MSVC compiler version required for debugging, refer to Basics on Integrating the Simulation System on page 62.
- The **Enable debug** build option must have been selected for the import.

  Refer to Import on page 152.

Restrictions

There is a limitation for VPUs built with the MSVC compiler. Refer to Limitations for Integrating the Simulation System on page 178.

Method

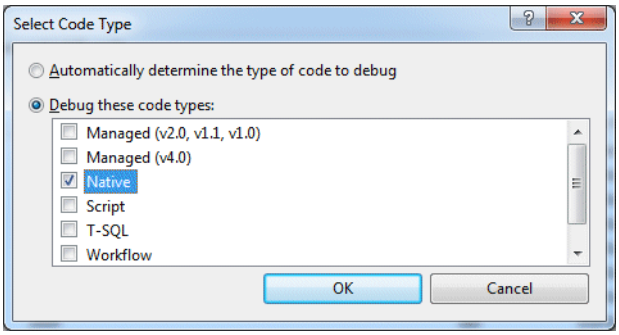
To debug source code built with the MSVC compiler in an offline simulation

1. Load the offline simulation application to VEOS. Refer to Load on page 216.
2. Start Microsoft® Visual Studio®.
3. In the **Debug** menu in Microsoft® Visual Studio®, select **Attach to Process**.

   Microsoft® Visual Studio® opens the **Attach to Process** dialog.
4. In the **Attach to Process** dialog, click **Select**.

   Microsoft® Visual Studio® opens the **Select Code Type** dialog.

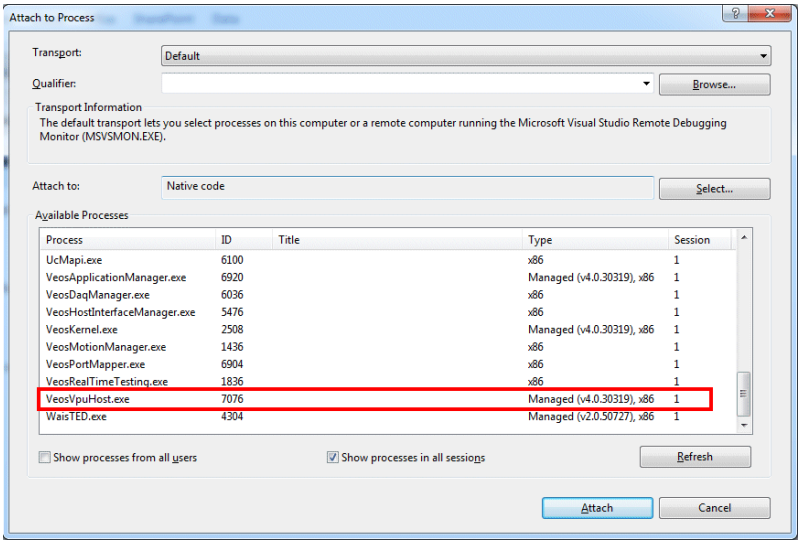
5 In the Select Code Type dialog, select Debug these code types and then Native.



6 Click OK to close the Select Code Type dialog.

7 In the Attach to Process dialog, select the VPU process to debug in the Available Processes list. VPU processes can be identified via their unique PID, see Separate VPU processes on page 203.

The illustration below shows one VPU process (with the PID 7076) as an example.



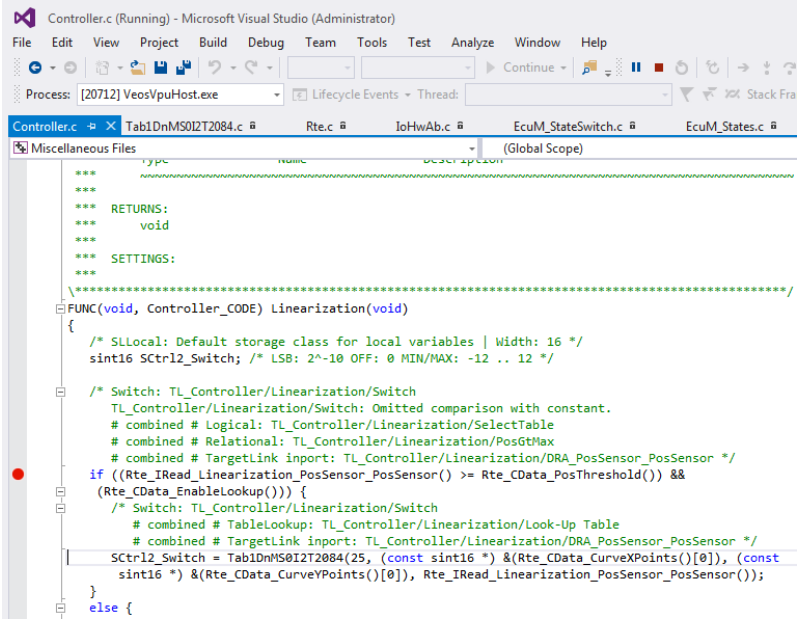
8 Click Attach.

9 In Microsoft® Visual Studio®, load all the code files to debug.

Note

You have to load the code files to be debugged from the VPU's build directory as specified on the Build Options page of the Import dialog. The code files are located in the `\ExtractedData` subdirectory of the build directory.

10 Set a breakpoint, see the example below.



11 Start the offline simulation application. Refer to Start on page 221.

12 The debugger stops at the specified breakpoint. You can now use the debugger as usual.

Tip

You can also let the debugger be stopped automatically when a serious error occurs. Refer to Stopping the debugger automatically when a serious error occurs.

Result

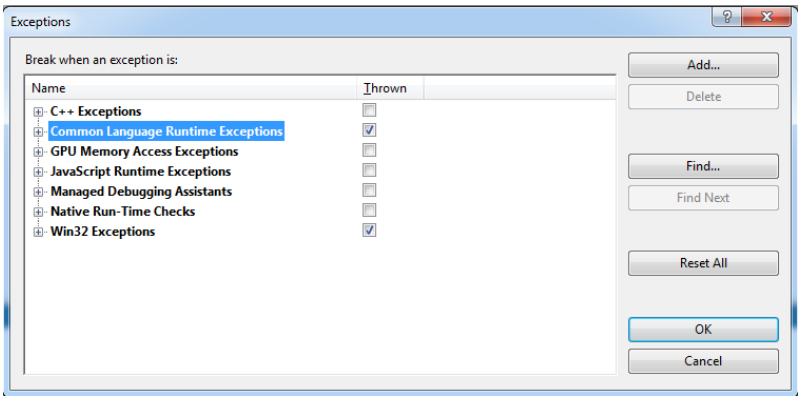
You have debugged the source code of a VPU built with the MSVC compiler using Microsoft® Visual Studio®.

Stopping the debugger automatically when a serious error occurs

To stop the debugger automatically when a serious error occurs:

1. Select Exceptions in the Debug menu of Microsoft® Visual Studio®.
   Microsoft® Visual Studio® opens the Exceptions dialog.

2. In the Exceptions dialog, select Common Language Runtime Exceptions and Win32 Exceptions.



3. Click OK to close the Exceptions dialog.

Debugging the load process

Problems during the OSA load process might occur, for example, in connection with VPUs containing handcoded S-functions. In this case, debugging the load process, i.e., debugging the loading and initializing of the binaries, is useful.

You have the following two options.

Debugging the load process via VEOS Player Perform the following steps:

1. On the VEOS Player ribbon, click File - Preferences and select Wait for Debugger on Load.
2. Load the offline simulation application via the VEOS Player.

   As a result, a message dialog opens for you to interrupt the load process and carry out the steps 2 ... 10 of the method described above.
3. To continue the load process, close the message dialog.

Debugging the load process via the VEOS command line interface Perform the following steps:

1. To preload the offline simulation application, enter the following command:

   On Windows: `veos.exe sim load "<path to osa>" --break`

   On Linux: `veos sim load "<path to osa>" --break`
2. Carry out the steps 2 ... 10 of the method described above.
3. To continue the load process, enter the following command:

   On Windows: `veos.exe sim load --continue`

   On Linux: `veos sim load --continue`

Related topics

Basics

Basics on Debugging Source Code in an Offline Simulation..... 202

References

Load..... 216
Wait for Debugger on Load..... 224

How to Debug Source Code Built with the GCC Compiler in an Offline Simulation

Objective

During offline simulation, you can debug the source code of VPUs built with the GNU GCC C Compiler by using the GNU debugger (GDB).

Preconditions

On the Build Options page of the VEOS Import dialog, the following specifications must be made for the VPU to debug:

Simulation target	HostPC64/GCC[1)]
Enable debug	✓[2)]

1) See Restrictions.
2) This enables the addition of debug symbol files to the generated VPU.

Refer to Import on page 152.

Restrictions

You can use the GDB contained in the GNU GCC C Compiler 5.2 (installed with VEOS 5.2) to debug 64-bit applications only.

Method

To debug source code built with the GCC compiler in an offline simulation

1. Load the offline simulation application to VEOS. Refer to Load on page 216.
2. In the file system of the host PC, change to the `\bin` folder of the GNU GCC C Compiler installation, and double-click `gdb.exe`. The console of the GDB opens.

   Tip

   The Simulation Target Manager displays the compiler installation path.

3. In the GDB console, type `attach <process_ID>` to attach the debugger to the VPU process to debug. VPU processes can be identified via their unique PID, see Separate VPU processes on page 203.

The following illustration shows an example:

```
C:\Program Files\Common Files\dSPACE\GCC\5.2\bin\gdb.exe
(gdb) attach 17544
Attaching to process 17544
[New Thread 17544.0xc5c]
[New Thread 17544.0x4544]
[New Thread 17544.0x1c0]
[New Thread 17544.0xf88]
[New Thread 17544.0x438c]
[New Thread 17544.0x2b50]
[New Thread 17544.0x4444]
[New Thread 17544.0x45b0]
[New Thread 17544.0x2ddc]
[New Thread 17544.0x19c8]
[New Thread 17544.0x2f94]
[New Thread 17544.0x32b0]
[New Thread 17544.0x18b0]
Reading symbols from C:\Program Files\dSPACE VEOS 4.5\Bin\VeosVpuHost.exe...(no debugging symbols found)...done.
warning: section .gnu_debuglink not found in C:\Users\egbertR\AppData\Local\Temp\dSPACE\VEOS\4.5\VeosKernel_13120\ModelC
ontroller\w0ir0mcj\Controller.dwo
0x00007ffec44b3a61 in ntdll!DbgBreakPoint () from C:\WINDOWS\SYSTEM32\ntdll.dll
(gdb)
```

4 Type `info functions <Expression>` to search for one or more code files and functions to debug.

For example, if you type `info functions Task100ms`, all the code files containing functions with `Task100ms` in their names are listed, as shown in the following illustration:

```
C:\Program Files\Common Files\dSPACE\GCC\5.2\bin\gdb.exe
(gdb) info functions Task100ms
All functions matching regular expression "Task100ms":

File .\ExtractedData\Rte\Rte.c:
void FuncTask100ms(void);
void Tsk_Task100ms(void);
(gdb)
```

5 Type `break <Function_Name>` to set a breakpoint.

For example, type `break FuncTask100ms` as shown in the following illustration:

```
C:\Program Files\Common Files\dSPACE\GCC\5.2\bin\gdb.exe
(gdb) break FuncTask100ms
Breakpoint 1 at 0x279516a8: file .\ExtractedData\Rte\Rte.c, line 124.
(gdb)
```

6 Type `continue` to resume the execution of the offline simulation.

The following illustration shows an example:

```
C:\Program Files\Common Files\dSPACE\GCC\5.2\bin\gdb.exe
(gdb) continue
Continuing.
[New Thread 17544.0x2158]
[Thread 17544.0x18b0 exited with code 0]
[New Thread 17544.0xd18]
[New Thread 17544.0x4ad4]
```

7 Start the offline simulation application. Refer to Start on page 221.

8 The debugger stops at the specified breakpoint. You can now use the debugger as usual.

The following illustration shows an example:

```
C:\Program Files\Common Files\dSPACE\GCC\5.2\bin\gdb.exe
(gdb) continue
Continuing.
[New Thread 17544.0x2158]
[Thread 17544.0x18b0 exited with code 0]
[New Thread 17544.0xd18]
[New Thread 17544.0x4ad4]
[Thread 17544.0x4544 exited with code 0]
[Thread 17544.0x1c0 exited with code 0]
[Thread 17544.0xf88 exited with code 0]
[New Thread 17544.0xb44]
[New Thread 17544.0x4e38]
[New Thread 17544.0x4b04]
[New Thread 17544.0x3c98]
[New Thread 17544.0x4998]

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 17544.0x4444]
0x00000000274dee80 in ?? ()
(gdb)
```

9 Analyze and debug the VPU source code:

GDB Command	Purpose
`bt`	To show the stack trace.
`list`	To show the source code.
`info args`	To show function arguments.
`info locals`	To show local variables.
`step`	To step through the source code.

Result

You have debugged the source code of a VPU built with the GNU GCC C Compiler using the GNU debugger (GDB).

Debugging the load process

Problems during the OSA load process might occur, for example, in connection with VPUs containing handcoded S-functions. In this case, debugging the load process, i.e., debugging the loading and initializing of the binaries, is useful.

You have the following two options.

Debugging the load process via VEOS Player Perform the following steps:

1. On the VEOS Player ribbon, click File - Preferences and select Wait for Debugger on Load.
2. Load the offline simulation application via the VEOS Player.

   As a result, a message dialog opens for you to interrupt the load process and carry out the steps 2 ... 6 of the method described above.
3. To continue the load process, close the message dialog.

Debugging the load process via the VEOS command line interface Perform the following steps:

1. To preload the offline simulation application, enter the following command:

   On Windows: `veos.exe sim load "<path to osa>" --break`

   On Linux: `veos sim load "<path to osa>" --break`
2. Carry out the steps 2 ... 6 of the method described above.
3. To continue the load process, enter the following command:

   On Windows: `veos.exe sim load --continue`

   On Linux: `veos sim load --continue`

Related topics

Basics

Basics on Debugging Source Code in an Offline Simulation..... 202

References

Load..... 216
Simulation Target Manager..... 163
Wait for Debugger on Load..... 224

Reference Information

Where to go from here

Information in this section

Break Time....213
To enter a time when the simulation pauses automatically.

Clear Simulator List....213
To clear the list of simulators.

Generate Bus Log Files....214
To log bus communication and the bus load.

Generate Simulation Log Files....215
To log specific information about the current simulation run.

Generate XCP Log Files....215
To log specific information about the XCP service.

Load....216
To load the open offline simulation application to VEOS and initialize the offline simulation.

Pause....218
To pause the simulation running on the PC.

Real-Time Acceleration Factor....218
To specify a factor to accelerate or decelerate the simulation run in relation to real time.

Select Simulator....219
To select a simulator for downloading the offline simulation application to.

Single Step Time....220
To specify the duration of one simulation step in seconds.

Start....221
To start simulation on the PC.

Step....222
To run simulations stepwise.

Stop....222
To stop the simulation running on the PC.

Stop Time....223
To enter the simulation stop time in seconds.

Unload....223
To unload the offline simulation application by cutting the connection to VEOS Simulator.

Wait for Debugger on Load....224
To enable the debugging of the load process.

Break Time

Access

You can access this command via:

Ribbon	Simulation – Control Options
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To enter a time when the simulation pauses automatically.

Description

You can resume the simulation via the (Start) button. The break time is limited to floating-point numbers smaller than the stop time. It is specified in seconds.

You can also enter **`Infinity`** or select the break time from the list to disable an automatic simulation break.

Related topics

References

Real-Time Acceleration Factor 218
Single Step Time 220
Start 221
Stop Time 223

Clear Simulator List

Access

You can access this command via:

Ribbon	Simulation - Load
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To clear the list of simulators.

Description

You can clear either the entire list by clicking the command's button or remove one of the items in the list by clicking ✕.

Result

The list of simulators is cleared.

Related topics

HowTos

How to Select a Remote Simulator 186

Generate Bus Log Files

Access

You can access this command via:

Ribbon	File – Preferences
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To log bus communication and the bus load.

Description

Lets you log bus communication and the bus load in the `<VPU>.BusTransfer.log` file.

The log options are saved in the VEOS Player preferences and are valid for every opened offline simulation application (OSA) file. When you save the offline simulation application, the options are not saved to the OSA file.

Related topics

References

Preferences 408

Generate Simulation Log Files

Access

You can access this command via:

Ribbon	File – Preferences
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To log specific information about the current simulation run.

Description

Specific information about the current simulation run is logged in the `<VPU>.Simulation.log` file for support requests.

Note

If you enable logging, the simulation is slower.

The log options are saved in the VEOS Player preferences and are valid for every opened offline simulation application (OSA) file. When you save the offline simulation application, the options are not saved to the OSA file.

Related topics

References

Preferences..... 408

Generate XCP Log Files

Access

You can access this command via:

Ribbon	File – Preferences
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To log specific information about the XCP service.

Description

Lets you log XCP communication for every XCP service in the `<VEcuName>.XCP.<XCP Ethernet port[1|2]>.log` file.

XCP communication is available only for VPUs containing an XCP service. XCP Ethernet port values can be found in the **Build time options** properties of the VPU.

The log options are saved in the VEOS Player preferences and are valid for every opened offline simulation application (OSA) file. When you save the offline simulation application, the options are not saved to the OSA file.

Related topics

References

Preferences..... 408

Load

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	`Ctrl` + `L`
Icon	

Purpose

To load the open offline simulation application to VEOS and initialize the offline simulation.

Description

VEOS Player lets you download the open offline simulation application to VEOS.

You can specify whether to download the application to the local simulator or to a remote simulator.

Using the Load button You can use the **Load** button as follows:

- If you click the arrow part of the **Load** button, VEOS Player opens the list of selected simulators. This list shows PC names, IP addresses, and VEOS versions. You can select an entry in the list to load the offline simulation application to.
- If you click the main icon part of the **Load** button, VEOS Player loads the open offline simulation application to the default simulator, which is first in the list of simulators.

If you changed the open offline simulation application, you are asked to save it before loading.

Loading when a simulation is running If an offline simulation application is already running, VEOS asks you whether to:

- Connect to the running offline simulation application
- Replace the running offline simulation application

Enumerations: Checking data type width During the download of an offline simulation application, VEOS automatically checks the data type width of enumerations and pointers. VEOS expects a data type width of 4 bytes. If it detects another width, VEOS issues an error message and stops the download of the offline simulation application.

VEOS performs this check:

- Independent of the container format
- Independent of the compiler

You can deactivate the check of the data type width and prevent VEOS from stopping the application download by specifying the following preprocessor defines:

Define	Description
`VEOS_NO_DTSIZE_CHECK`	To deactivate the check for enumerations and pointers.
`VEOS_NO_DTSIZE_CHECK_ENUM`	To deactivate the check for enumerations only.
`VEOS_NO_DTSIZE_CHECK_PTR`	To deactivate the check for pointers only.

Note

Keep in mind that deactivating the check of the data type width is useful only if enumerations and pointers are not part of the application's variable description.

Result

The offline simulation is loaded to the simulator you specified. You can now run the simulation.

Related topics

HowTos

How to Load and Run an Offline Simulation Application..... 188
How to Select a Remote Simulator..... 186

Pause

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	**F6**
Icon	

Purpose

To pause the simulation running on the PC.

Result

A running simulation is paused.

Related topics

References

Start.....221
Stop.....222

Real-Time Acceleration Factor

Access

You can access this command via:

Ribbon	Simulation – Control Options
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To specify a factor to accelerate or decelerate the simulation run in relation to real time.

Description

Specify a real-time acceleration factor:

Factor	Description
1	To run the simulation in real time[1)].
> 1	To run the simulation faster than real time[1)].

Factor	Description
< 1	To run the simulation slower than real time[1].
0	To run the simulation as fast as possible.

[1] Specifying a real-time acceleration factor that is not equal to 0 enables soft real time simulation. For more information, refer to Simulation in soft real time on page 26.

To avoid the loss of measurement and stimulation data, VEOS automatically decelerates the simulation run if more measurement data is generated than the experiment software, e.g., ControlDesk, can sample. VEOS also decelerates the simulation if stimulation data is not received in time. This is independent of the specified real-time acceleration factor, i.e., receiving complete measurement data has higher priority than achieving the specified real-time acceleration factor.

Related topics

Basics

Timing in Offline Simulation with VEOS.....24

References

Break Time.....213
Single Step Time.....220
Stop Time.....223

Select Simulator

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	`Ctrl` + `R`
Icon	

Purpose

To select a simulator for downloading the offline simulation application to.

Description

VEOS Player opens the **Select Simulator** dialog that lets you select a simulator in your local network.

Select Simulator dialog

To scan for PCs that can act as remote simulators in your local network and to select a simulator. For details on preconditions for simulators, refer to How to Select a Remote Simulator on page 186.

Filter scan results Lets you specify a filter term to reduce the number of simulators displayed.

Scan for Simulators To scan or rescan for simulators in your local network.

Computer name or IP address Lets you specify a computer name or an IP address.

Try find To find the simulator with the specified computer name or IP address.

Select Simulator To add the selected simulator to the simulator list. For loading the offline simulation application to a simulator, refer to Load on page 216.

Related topics

HowTos

How to Load and Run an Offline Simulation Application..... 188
How to Select a Remote Simulator..... 186

References

Load..... 216

Single Step Time

Access

You can access this command via:

Ribbon	Simulation – Control Options
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To specify the duration of one simulation step in seconds.

Description

The duration specification is used when you perform a single-step simulation. Refer to Step on page 222.

Related topics

References

Break Time..... 213
Real-Time Acceleration Factor..... 218
Step..... 222
Stop Time..... 223

Start

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	**F5**
Icon	

Purpose

To start simulation on the PC.

Description

You can start the simulation without having loaded the offline simulation application to VEOS. The offline simulation application is first automatically loaded to VEOS and then the simulation is started. For loading the offline simulation application, refer to Load on page 216.

Related topics

HowTos

How to Load and Run an Offline Simulation Application..... 188

References

Pause..... 218
Stop..... 222

Step

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	**F7**
Icon	

Purpose

To run simulations stepwise.

Result

The next step of a simulation is executed. You can run all the steps separately in consecutive order.

Description

You can use this command to inspect a simulation in detail. Each time you use the command, the next simulation step is executed.

You can set the single step time using the Single Step Time command.

Related topics

References

Single Step Time..... 220

Stop

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	**F8**
Icon	

Purpose

To stop the simulation running on the PC.

Result

A running simulation is stopped.

Related topics

References

Pause 218
Start 221

Stop Time

Access

You can access this command via:

Ribbon	Simulation – Control Options
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To enter the simulation stop time in seconds.

Description

You can enter **`Infinity`** to perform an endless simulation run.

Related topics

References

Break Time 213
Real-Time Acceleration Factor 218
Single Step Time 220

Unload

Access

You can access this command via:

Ribbon	Simulation – Control
Context menu of	None
Shortcut key	**`Ctrl`** + **`U`**
Icon	

Purpose

To unload the offline simulation application by cutting the connection to VEOS Simulator.

Description

Other products such as ControlDesk can then load another offline simulation application.

Result

The offline simulation is stopped and the offline simulation application is unloaded from the PC.

Related topics

References

Load..... 216

Wait for Debugger on Load

Access

You can access this command via:

Ribbon	File – Preferences
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To enable the debugging of the load process.

Description

When you select this option and load an offline simulation application via the VEOS Player, a message dialog opens for you to interrupt the load process, and attach a debugger to the offline simulation application processes to debug.

For instructions, refer to:

- How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation on page 205
- How to Debug Source Code Built with the GCC Compiler in an Offline Simulation on page 209

Related topics

HowTos

How to Debug Source Code Built with the GCC Compiler in an Offline Simulation..... 209
How to Debug Source Code Built with the MSVC Compiler in an Offline Simulation..... 205

References

Load..... 216
Preferences..... 408

Automation

Where to go from here

Information in this section

Automating the Simulation Configuration.....226
You can automate the configuration of an offline simulation.

Automating the Download and Control of an Offline Simulation Application.....227
You can automate the download and the control (start, pause and stop) of an offline simulation application.

Automating VEOS Player Event Handling with Python.....228
You can handle events such as OnSimulationStarted in automation scripts.

Automating the Simulation Configuration

Introduction

You can automate the configuration of an offline simulation.

The program listings below consist of excerpts from the Python demo scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: [...].

Configuring the simulation

The following listing shows how you can configure the simulation:

- Enable bus simulation logging
- Enable XCP simulation logging
- Enable simulation logging
- Set the duration of a simulation run to infinity
- Set the single step time to 0.0
- Disable an automatic simulation break.
- Set the real-time acceleration factor to 1

```
def ConfigureOfflineSimulationApplication(application):
    configuration = application.Configuration
    configuration.EnableBusSimulationLog = True
    configuration.EnableXcpSimulationLog = False
    configuration.EnableSimulationLog = True
    configuration.SimulationTime = 3.0E9999
    configuration.SingleStepTime = 0.0
    configuration.BreakAtTime = Infinity
    configuration.AccelerationFactor = 1.0
```

Configuring VPUs

The following listing shows how you can configure VPUs in the loaded offline simulation application:

- Set the COM port IDs
- Set the timer drift
- Set the timer offset

```
def ConfigureVpus(vpus):
    comPortCounter = 1
    for vpu in vpus.Elements:
        vpu.ComPortId = comPortCounter
        comPortCounter += 1
        vpu.Drift = 0.0
        vpu.Offset = 0.0
```

Related topics

References

ISimulationConfiguration..... 364
IVpu..... 368

Automating the Download and Control of an Offline Simulation Application

Introduction

You can automate:

- The download of an offline simulation application
- The control of an offline simulation by starting, pausing and stopping it

The program listings below consist of excerpts from the Python demo scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: [...].

Downloading an offline simulation application

Download on a local simulator The following listing shows how you can download an offline simulation application to VEOS on the local simulator.

```
def DownloadSimulationApplication(player):
    simulator = player.Simulator
    simulator.Download()
```

Download on a remote simulator The following listing shows how you can download an offline simulation application to VEOS on a remote simulator. The IP address is provided as a parameter of the `Download` method.

```
def DownloadSimulationApplication(player):
    simulator = player.Simulator
    simulator.Download("10.1.101.172")
```

Starting a simulation run

The following listing shows how you can start an offline simulation run.

```
def StartSimulation(player):
    simulator = player.Simulator
    simulator.Start()
```

Stopping a simulation run

The following listing shows how you can stop an offline simulation run after 60 seconds.

```
def StartSimulation(player):
    time.sleep(60)
    simulator = player.Simulator
    simulator.Stop()
```

Related topics

Basics

Loading and Running an Offline Simulation Application, and Debugging Source Code 183

References

ISimulator 365

Automating VEOS Player Event Handling with Python

Introduction

You can handle events such as `OnSimulationStarted` in automation scripts.

Methods for handling events

The VEOS Player provides events which you can handle in automation scripts. You can use the `OnSimulationStopped` event, for example, to load a new simulation system to VEOS directly after a simulation run.

Tip

The control of a VEOS simulation via automation is synchronous, so that using events for simulation control is not necessary.

You can define the following methods in Python scripts to handle events:

- `def OnSimulationStarted(self):`
- `def OnSimulationPaused(self):`
- `def OnSimulationContinued(self):`
- `def OnSimulationStopped(self):`
- `def OnSimulationSystemDownloaded(self):`

- `def OnSimulationSystenUnloaded(self):`
- `def OnSimulationSystemTerminated(self):`
- `def OnVpuSettingsChanged(self):`

For details on these events, refer to ISimulatorEvents on page 367.

Workflow

1. Open a COM connection to the VEOS Player version you want to automate. Refer to Automating Different VEOS Player Versions on page 329.
2. Write a Python script that loads the Python event library. See Loading the Python event library on page 229.
3. In the script, define the above methods in the context of a class. After you have instantiated the class with the automation element that provides the events, you can handle the events by the defined event handler methods. Refer to Defining event handler classes on page 229.

Loading the Python event library

The following listing shows how to load the Python event library for the VEOS Player.

```
import os, sys, time
import importlib
importlib.reload(VEOS_SimulationEventsLib)
```

The `VEOS_SimulationEventsLib.py` Python event library is imported and loaded to make its elements accessible to the VEOS Player's automation feature.

Python event library location The `VEOS_SimulationEventsLib.py` Python event library is part of the `1Ecu_Standard.zip` demo ZIP file. Refer to Demos for VEOS on page 37.

Defining event handler classes

The following listings show how you can define an event handler class.

```
class SimulatorEvents(VEOS_SimulationEventsLib.ISimulatorEvents):
   def OnSimulationContinued(self):
      print("Simulation continued.")
   def OnSimulationPaused(self):
      print("Simulation paused.")
   def OnSimulationStarted(self):
      print("Simulation started.")
   def OnSimulationStopped(self):
      print("Simulation stopped.")
   def OnSimulationSystemDownloaded(self):
      print("Simulation system downloaded.")
   def OnSimulationSystenUnloaded(self):
      print("Simulation system unloaded.")
   def OnSimulationSystemTerminated(self):
      print("Simulation system terminated.")
   def OnVpuSettingsChanged(self):
      print("VPU settings changed.")
```

The `SimulatorEvents` class is defined. The `ISimulatorEvents` element of the `VEOS_SimulationEventsLib.py` Python library is used as a class parameter. The class lets you handle simulation events.

Handling simulation events

The following listings show how you can handle simulation events. During simulation, events are handled by printing messages.

- The `events = SimulatorEvents(player.Simulator)` statement at the beginning of the `main` function connects the simulator events.
- The `events.close()` and `del events` statements at the end of the `main` function are used to close and delete the event instance.

Note

Keep in mind to close and delete the event instance at the end of your automation script.

```
from win32com.client import Dispatch
import os, sys, time
import importlib
importlib.reload(VEOS_SimulationEventsLib)
```

```
class SimulatorEvents(VEOS_SimulationEventsLib.ISimulatorEvents):
   def OnSimulationContinued(self):
      print("Simulation continued.")
   def OnSimulationPaused(self):
      print("Simulation paused.")
   def OnSimulationStarted(self):
      print("Simulation started.")
   def OnSimulationStopped(self):
      print("Simulation stopped.")
   def OnSimulationSystemDownloaded(self):
      print("Simulation system downloaded.")
   def OnSimulationSystenUnloaded(self):
      print("Simulation system unloaded.")
   def OnSimulationSystemTerminated(self):
      print("Simulation system terminated.")
   def OnVpuSettingsChanged(self):
      print("VPU settings changed.")
```

```
def CreatePlayer():
    player = Dispatch("VeosPlayer.Application")
    return player
```

```
def SimulateApplication(player):
    print ("Simulation ...")
    simulator = player.Simulator
    simulator.Download()
    print (" ... downloaded")
    simulator.Start()
    print (" ... started")
    time.sleep(5)
    simulator.Pause()
    print (" ... paused")
    simulator.Step()
    print (" ... stepped")
    simulator.Stop()
    print (" ... stopped")
    simulator.Unload()
    print (" ... unloaded")
def QuitPlayer(player):
    player.Quit()
def main():
    player = CreatePlayer()
    events = SimulatorEvents(player.Simulator)
    application = OpenOfflineSimulationApplication(player)
    SimulateApplication(player)
    events.close()
    del events
    QuitPlayer(player)
```

Related topics

References

ISimulatorEvents..... 367

Limitations

Limitations for Loading and Running an Offline Simulation Application, and Debugging Source Code

Limitations for simulating on a remote simulator

There are the following limitations when using a remote simulator for simulation:

- Source code debugging is not possible on the remote simulator.
- Unlike for SCALEXIO, there is no Web interface for configuring VEOS.
- Adaptive V-ECUs cannot be simulated on a remote simulator.

Limitations for stimulating V-ECU variables

Real-Time Testing (RTT) is used for the stimulation of V-ECU variables.

Stimulation in the V-ECU's fastest basic task only The RTT service is executed in the V-ECU's fastest basic task. As a result, V-ECU variables can be stimulated only when the fastest basic task is executed, independent of the priorities of the tasks of the V-ECU.

Example:

A V-ECU has the following two tasks:

Task	Period
T_1	2 ms
T_2	3 ms

Since T_1 is the fastest basic task, the RTT service is executed every 2 ms. As a consequence, task T_2 can be stimulated only every second time, independent of the priorities of T_1 and T_2.

No V-ECU stimulation in some cases Stimulating V-ECU variables is not possible as long as the V-ECU is not initialized completely, for example, if the start of a V-ECU is delayed. RTT is not accessible during V-ECU initialization.

Limitations for measuring variables during simulation

Measurement rasters You have to select a measurement raster for measuring a variable during simulation. Each raster corresponds to the sample time of an OS task specified for the ECU.

However, you can select only rasters derived from *basic tasks* of the ECU. Rasters derived from the ECU's *extended tasks* are not available for measuring variables during simulation.

Maximum number of variables that can be measured The number of variables that can be measured is limited.

Data Type	Maximum Number of Variables That Can Be Measured ...	
	... A2L Variables	... TRC Variables[1)]
Float, int, short, or byte	Up to 64,260	Up to 1,048,576
Double	Up to 44,100	Up to 131,072

[1)] The maximum numbers apply to measurements with only a single measurement raster. If you use multiple measurement rasters, higher maximum numbers apply. However, these numbers depend on the specific measurement scenario.

Adaptive V-ECUs: Measurement and calibration not possible Measurement and calibration is not possible on an adaptive V-ECU.

Limitation for simulating a system containing one or more bus VPUs

When you stop an offline simulation application containing one or more bus VPUs and then restart it, this may lead to faulty simulation results. To avoid this, you have to unload the offline simulation application before you restart it.

Refer to Unload on page 223.

Limitation for rapid prototyping access to V-ECU internal variables and functions

Rapid prototyping access to V-ECU internal variables and functions, i.e., virtual bypassing, is not supported for 64-bit V-ECUs.

Limitations for LIN bus simulation

There are some limitations for LIN bus simulation of V-ECUs:

- Only the following frame types are supported:
  - Event-triggered frames
  - Sporadic frames
  - Unconditional frames
- As soon as the offline simulation application is loaded to VEOS, you can switch the LIN schedule table *before V-ECU start* via specific A2L variables. This requires dSPACE-specific *network management stub* code. For details, refer to LIN Interface (LinIf) (SystemDesk Manual).

  However, switching schedule tables manually *after V-ECU start* is not supported. Automatic switching schedule tables in connection with frame collisions, however, is supported.
- LIN module configurations with LIN node configuration are not supported.

Limitation for Ethernet communication simulation

Ethernet communication simulation is supported only for V-ECUs.

Limitation for simulating multicore V-ECUs

On VEOS, the tasks of a multicore V-ECU cannot be calculated in parallel on several cores. Instead, the task processing in an offline simulation is based on the simulated AUTOSAR OS scheduler, which executes all tasks on one physical core of the simulator. The sequence of task execution depends on the scheduling policy. The VEOS implementation of the AUTOSAR OS scheduler respects the configured task priorities in each OS application, but other core-dependent effects cannot be simulated in VEOS.

However, because the deterministic behavior of the offline simulation is ensured, offline simulation of multicore V-ECUs is suitable for detecting errors in the V-ECU code.

The following multicore OS functions are limited due to the serial processing of OS tasks:

- `StartCore()`: VEOS does not start a core or a parallel thread.
- `StartOs()` and `ShutdownOS()`: VEOS does not run the functions in parallel on several cores.
- `GetSpinlock()`: VEOS ignores the spinlock to avoid a deadlock and outputs a related warning. However, due to the serial processing of OS tasks on VEOS, this warning will rarely be issued.
- `ShutdownOS()`: VEOS does not shut down the core on which the function is called. However, all the alarms, schedule tables, and tasks belonging to the core are stopped.

Related topics

Basics

Simulating Multicore V-ECUs.....27

Working with Adaptive V-ECUs in VEOS

Where to go from here

Information in this section

Adaptive V-ECUs in VEOS.....236
Working with the VEOS Adaptive Platform Demos.....248
Working with the AUTOSAR Adaptive Platform Demonstrator Samples.....266
Working with the GPU Passthrough Demo.....271

Adaptive V-ECUs in VEOS

Where to go from here

Information in this section

Basics on Simulating Adaptive V-ECUs.....236
Describes simulation aspects specific to adaptive V-ECUs.

Hypervisor Requirements for Adaptive V-ECU Simulation.....239
Because VEOS simulates adaptive V-ECUs on a virtual Linux machine, you require a hypervisor on the VEOS host PC.

Workflow for Creating and Simulating Adaptive V-ECUs.....241
The dSPACE tool chain lets you create and simulate adaptive V-ECUs.

Basics on Adaptive V-ECU Implementation Layers.....243
There are several ways of creating implementation layers for adaptive V-ECUs. This topic provides an overview with examples of the available methods.

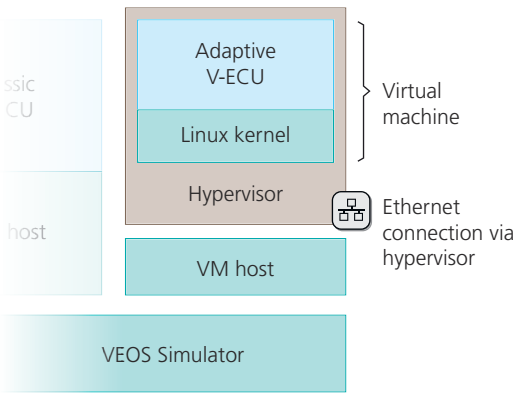
Basics on Debugging Adaptive V-ECUs.....244
The simulation architecture of adaptive V-ECUs supports remote debugging.

How to Debug an Adaptive V-ECU.....245
To debug an adaptive V-ECU during offline simulation.

Basics on Simulating Adaptive V-ECUs

Introduction

The following illustration shows the architecture for simulating adaptive V-ECUs in VEOS.



The adaptive V-ECU contains a set of adaptive applications, which are specific instances of Linux applications.

Adaptive V-ECU

In VEOS, an adaptive V-ECU is based on the adaptive V-ECU implementation container (VECU) file created with SystemDesk and integrated with the VEOS Player or with the VEOS command line interface.

An adaptive V-ECU consists of the following components:

- Layers of middleware and adaptive applications that are copied to the Linux virtual machine which simulates the adaptive V-ECU.
- Target machine configuration (Tmc) module: A configuration module for specifying the virtual hardware that is used to simulate the adaptive V-ECU.

Depending on the host operating system, the layers can consist of archive files or a QEMU image. Refer to the following table.

Host Operating System	Layer File Type
Windows	Archive files
Linux	▪ Archive files ▪ QEMU image

Linux kernel

dSPACE provides a specific Linux kernel for running Linux-based applications, such as adaptive V-ECUs. During offline simulation on VEOS, the VEOS Simulator and the dSPACE VEOS Linux kernel are synchronized. Refer to Time synchronization on page 238.

Virtual machine

The Linux kernel and the adaptive V-ECU are executed on a virtual machine.

Hypervisor

The hypervisor provides the environment for running the virtual machine. The hypervisor communicates with the VEOS Simulator via the *VM host*.

For more information on using hypervisors, refer to the following pages:

- For VEOS on Windows, refer to Hypervisor Requirements for Adaptive V-ECU Simulation on page 239.
- For VEOS on Linux, refer to Hypervisor Configuration for Adaptive V-ECU Simulation on page 296.

Ethernet connection

The VEOS simulation architecture enables the adaptive V-ECU to communicate via Ethernet with other V-ECUs or with applications outside of the simulation. This lets you, for example, connect a remote debugger.

You can specify the connection of the adaptive V-ECU to the Ethernet network of the host PC on the **Hypervisor Options** page of the **Import** on page 152 dialog.

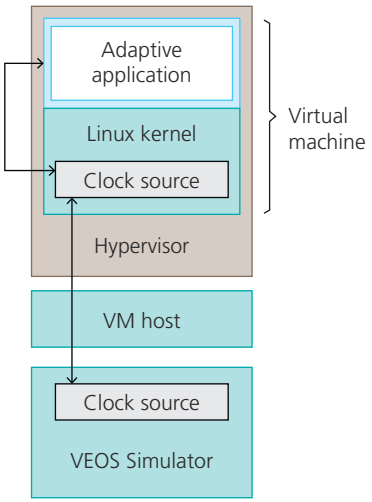
Tip

The VEOS Adaptive Platform demos provide a remote-debugging scenario. Refer to Remote-debugging an adaptive V-ECU (GDB) on page 252.

Limitation for external access to adaptive V-ECUs Only one adaptive V-ECU in a simulation can have an Ethernet network connection that allows access from outside the simulation, i.e., **HostOnly**, **Bridged**, or **NAT**. For any other adaptive V-ECUs in the simulation, the connection must be set to **Disconnected**.

Time synchronization

In the simulation of an adaptive V-ECU, VEOS synchronizes the Linux clock source with the global simulation time. Whenever an application in an adaptive V-ECU requests time information, it uses the Linux clock source. Refer to the following illustration:



The Linux real-time clock devices (/dev/rtc, /dev/rtc0, /dev/rtc1, etc.) are not synchronized with the simulation time in VEOS.

Related topics

Basics

Basics on Adaptive V-ECU Implementation Layers..... 243
Basics on the VEOS Command Line Interface..... 315
Hypervisor Requirements for Adaptive V-ECU Simulation..... 239
Simulation Architecture of VEOS..... 23
Workflow for Creating and Simulating Adaptive V-ECUs..... 241

Working with the VEOS Adaptive Platform Demos..... 248

HowTos

How to Load and Run an Offline Simulation Application..... 188

References

Import..... 152

Hypervisor Requirements for Adaptive V-ECU Simulation

Introduction

Because VEOS simulates adaptive V-ECUs on a virtual Linux machine, you require a hypervisor on the VEOS host PC.

Hardware support for virtualization (Intel VT-X/AMD-V) in the BIOS or UEFI of the host PC must be activated.

On Windows, VEOS supports the following hypervisor software:

- VMWare Workstation 15 Player or Pro
- Microsoft Hyper-V on all Windows 10 operating systems supported by dSPACE. Refer to Operating System (Installing dSPACE Software 🕮).

Note

If Hyper-V is active, it is used for virtualization regardless of whether VMware Workstation Player is installed on the same host PC.

Hypervisor connection

When you import an adaptive V-ECU to an offline simulation application in VEOS, the Import dialog lets you select the connection type of the hypervisor. The connection type determines if and how the adaptive V-ECU can communicate with network participants outside the simulation.

Virtual network adapters The hypervisor software creates virtual network adapters that provide access to the Ethernet network of the host. Depending on the required connection type, the access can be restricted to the host, or it can extend to other machines in the network.

The IP addresses of the virtual network adapter and the adaptive V-ECU that is hosted by the hypervisor must be in the same subnet.

For more information on the virtual network adapters provided by the supported hypervisor software on Windows, refer to Hypervisor Connection on page 156.

Hypervisor MAC range The Import dialog also lets you specify a range of MAC addresses (first four bytes) that is used for the virtual machine hosted by the hypervisor.

When you select a connection type other than **Disconnected**, the specified range is used to generate MAC addresses for the virtual network adapter in the local network of the host PC.

Note

The specified MAC range is *not* used for the Ethernet adapter of the adaptive V-ECU. The Ethernet adapter of the adaptive V-ECU must be specified in the Target Machine Configuration (`Tmc`) module configuration of the adaptive V-ECU implementation. This is a *SystemDesk* task. For more information, refer to Tmc Module (SystemDesk Manual 🕮).

Settings specific to Hyper-V

To work with Microsoft Hyper-V, you have to run the `ConfigureHyperV.bat` script, which is located under `\Tools\Hypervisor\HypervConfiguration` in the VEOS installation folder.

The script asks you for confirmation to perform the following tasks:

- Enabling Hyper-V
- Registering required Hyper-V sockets
- Adding you to the Hyper-V Administrator group

Confirm each request with `Y`. After the script has finished, restart the PC to complete the activation of Hyper-V.

After you activated Hyper-V, you can access the **Hyper-V Manager** via the Windows Start menu to further configure Hyper-V.

You can reverse the changes made by the `ConfigureHyperV.bat` script by running the `UnconfigureHyperV.bat` script and restarting the PC.

Enabling bridging To enable bridging, you have to manually change some settings in the **Hyper-V Manager**: Under **Actions**, select **Virtual Switch Manager**. In the **Virtual Switch Manager**, select the `dSPACE_VEOS_5.2_Switch_Internal` switch and choose **External network** under **Connection type**. Leave the **Allow management operating system to share this network adapter** checkbox unchanged.

VEOS on a virtual machine

If you run VEOS on a virtual machine that is hosted by the ESXi hypervisor, you have to take the following compatibility requirements and settings into account.

Hypervisor Used for Adaptive V-ECU Simulation	Required ESXi Versions and VM Configuration Settings in ESXi Client
VMware Workstation 15 Player or Pro	▪ ESXi version 5.5 or higher ▪ VM configuration in ESXi clients: ▪ From **Summary** – **VM hardware**, select **Compatibility**: Set to hardware version 10 or higher.

Hypervisor Used for Adaptive V-ECU Simulation	Required ESXi Versions and VM Configuration Settings in ESXi Client
	▪ From Summary – VM hardware – CPU, select Hardware virtualization: Set to Enabled.
Microsoft Hyper-V on Windows 10	▪ ESXi version 6.5 or higher ▪ VM configuration in ESXi clients: ▪ From Summary – VM hardware, select Compatibility: Set to hardware version 13 or higher. ▪ From Summary – VM hardware – CPU, select Hardware virtualization: Set to Enabled. ▪ From Summary – VM hardware, select Edit Settings .... On the Edit Settings page, select VM Options – Advanced – Configuration Parameters. Open EDIT CONFIGURATION and add the following name and value: – Name: hypervisor.cpuid.v0 – Value: FALSE **Note** Setups combining hypervisors of different software providers can be unstable and are therefore not recommended.

Related topics

Basics

Basics on Simulating Adaptive V-ECUs 236
Hypervisor Configuration for Adaptive V-ECU Simulation 296
Workflow for Creating and Simulating Adaptive V-ECUs 241

References

Import 152

Workflow for Creating and Simulating Adaptive V-ECUs

Introduction

The dSPACE tool chain lets you create and simulate adaptive V-ECUs.

Workflow for creating and simulating an adaptive V-ECU

The following steps show the workflow for creating an adaptive V-ECU from an empty template in SystemDesk and simulating it on VEOS:

1. Create layers of adaptive applications and middleware that you want to run on the adaptive V-ECU. Layers can be tarballs, for example.

Refer to Basics on Adaptive V-ECU Implementation Layers on page 243 for more information on adaptive V-ECU layers.

2. In SystemDesk:
   1. Create an empty adaptive V-ECU.
   2. Add the layers containing the adaptive applications and middleware to the adaptive V-ECU.

      When you simulate the adaptive V-ECU in VEOS, the layers are loaded in the order specified in the V-ECU implementation.
   3. Specify the Ethernet adapter of the adaptive V-ECU by configuring the target machine configuration (`Tmc`) module.
   4. Export the V-ECU implementation of the adaptive V-ECU.

   For more information, refer to How to Create an Adaptive V-ECU (Empty) (SystemDesk Manual 🕮).

   For instructions on how to create an adaptive V-ECU based on an AUTOSAR model, refer to How to Create an Adaptive V-ECU (Based on AUTOSAR Model) (SystemDesk Manual 🕮).
3. In VEOS:
   1. Import the implementation container of the adaptive V-ECU. When you import the V-ECU implementation of an adaptive V-ECU, you have to specify the simulator network settings that can be used to access the adaptive V-ECU. For more information, refer to Hypervisor Options page (Import dialog) on page 155.
   2. Run the offline simulation. During simulation, information on the virtual machine that is used to simulate the adaptive V-ECU can be displayed in a separate hypervisor-specific window.

      Note

      In VEOS on Windows, the kernel boot of the virtual machine that hosts the adaptive V-ECU is part of the load process and is therefore not counted as simulation time. The simulation starts with the Linux init process, i.e., when user-space processes are started.

      In VEOS on Linux, however, the simulation starts with the Linux kernel boot of the virtual machine. Therefore, the simulation time is greater than zero when the init process starts.

   For more information, refer to Importing V-ECU Implementations on page 65 and Loading and Running an Offline Simulation Application on page 184.

Related topics

Basics

Basics on Adaptive V-ECU Implementation Layers..... 243
Basics on Simulating Adaptive V-ECUs..... 236
Hypervisor Requirements for Adaptive V-ECU Simulation..... 239
Working with the VEOS Adaptive Platform Demos..... 248

Basics on Adaptive V-ECU Implementation Layers

Introduction

There are several ways of creating implementation layers for adaptive V-ECUs:

- Exporting adaptive applications and middleware as archive files, e.g., tarballs
- Creating a QEMU image with the adaptive applications and middleware (only for VEOS on Linux)
- Creating tarballs from Docker images

Init script You always have to include a Linux init script that serves as the entrypoint of the simulation in one of the layers.

When you have created all the layers you require, you can follow the workflow described in Workflow for Creating and Simulating Adaptive V-ECUs on page 241 to create and simulate an adaptive V-ECU.

Creating archive files

You can use any archiving program to export the adaptive applications and middleware as archive files, for example, tarballs.

On Linux, the following command creates an exemplary `layer.tar` tarball from an `adaptive_application` binary and an `init.sh` init script:

```
tar -cf layer.tar adaptive_application init.sh
```

Creating a QEMU image

When working with VEOS on Linux, you can use a QEMU image as an implementation layer for an adaptive V-ECU.

You can create a QEMU image directly via a standard tool for creating disk images, such as `mkfs`, or you can create an image from a base image and archive files.

The following program listing shows an example of manually creating the `image.raw` QEMU image by using the standard DSKVM image as a base image and adding the `layer1.tar`, `layer2.tar`, and `layer3.tar` tarballs to it:

```
DSKVM_IMG=/opt/dspace/common/dskvm/5.2/bin/dskvm-img
mkfs.ext4 -G4096 -m0 image.raw 4g
mkdir mnt
${DSKVM_IMG} mount-loop image.raw mnt
${DSKVM_IMG} untar-root layer1.tar mnt
${DSKVM_IMG} untar-root layer2.tar.gz mnt
${DSKVM_IMG} untar-root layer3.tgz mnt
${DSKVM_IMG} umount-loop mnt
```

Note

If you use both a QEMU image and archive files as implementation layers for an adaptive V-ECU, the QEMU image must be the first layer.

Preparing Docker containers

The following example shows you how to create a tarball from a Docker image so that you can use it as an implementation layer for an adaptive V-ECU.

The example uses the Ubuntu 18.04 Docker image and adds a standard `/sbin/init` script as the simulation entrypoint.

1. In a temporary folder, create a Dockerfile with the following content:

```
FROM amd64/ubuntu:18.04
RUN echo "#!/bin/bash \n\
export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin\n\
export HOME=/root\n\
/bin/bash --login" >> /sbin/init
RUN chmod 755 /sbin/init
```

2. In the same folder, open a PowerShell and run the following commands:

```
docker image build -t ubuntu .
docker create --name ubuntu ubuntu
docker container export -o ../ubuntu.tar ubuntu
```

   This creates the `ubuntu` Docker image on the basis of the Dockerfile you created in the previous step. From the Docker image, a Docker container with the same name is created. Finally, the third line exports the Docker container as a tarball for import into SystemDesk.

Related topics

Basics

Basics on Simulating Adaptive V-ECUs 236
Workflow for Creating and Simulating Adaptive V-ECUs 241

HowTos

How to Create an Adaptive V-ECU (Empty) (SystemDesk Manual)

Basics on Debugging Adaptive V-ECUs

Introduction

The simulation architecture of adaptive V-ECUs supports remote debugging.

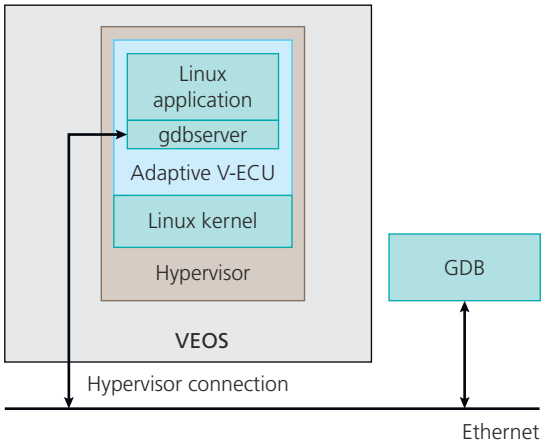
Simulation aspects specific to adaptive V-ECUs

In VEOS, an adaptive V-ECU is simulated on a virtual Linux machine. For more information, refer to Basics on Simulating Adaptive V-ECUs on page 236.

The connection to other V-ECUs or external applications is established via the hypervisor that hosts the virtual machine.

Remote debugging

To be able to debug an application on an adaptive V-ECU, you have to start the application via the gdbserver program. This allows for a remote debugger, i.e., GDB, to be attached. Refer to the following illustration.



The remote debugger can run on the operating system of the host PC, on another virtual machine, or on another PC.

The hypervisor must connect the Ethernet controller of the adaptive V-ECU to the Ethernet network of the GNU Debugger (GDB) host.

Related topics

HowTos

How to Debug an Adaptive V-ECU..... 245

How to Debug an Adaptive V-ECU

Objective

To debug an adaptive V-ECU during offline simulation.

Preconditions

- You must have included a script on the adaptive V-ECU that starts the application to be debugged via gdbserver.
- You must have the GNU Debugger (GDB) in one of the following configurations:
  - As a cross-compiler on the host PC.
  - On a virtual machine that runs on the host PC.
  - On an external PC.

  You can also use an IDE that supports remote debugging with GDB, such as Eclipse.
- You must have a copy of the application to be debugged on the GDB host machine.

- When you import the adaptive V-ECU to the OSA, you must specify the hypervisor connection according to your GDB setup:

GDB Setup	Hypervisor Connection
GDB on host PC	HostOnly
GDB on virtual machine on host PC	HostOnly
GDB on external PC	Bridged **Note** Consult your IT administrator before setting the hypervisor connection to Bridged.

Method

To debug an adaptive V-ECU

1. Load and start the OSA that contains the adaptive V-ECU.

   VEOS starts the virtual machine that hosts the adaptive V-ECU.

   The simulation pauses almost immediately after it starts and waits for a remote debugger to attach.

2. Start GDB on the GDB host machine via a command-line interface. The start command must include as the first parameter the path to the local copy of the application to be debugged. The name of the GDB executable depends on your GDB host system:
   - Linux: `gdb <path to application>`
   - Windows: `x86_64-linux-gnu-gdb.exe <path to application>`

3. In GDB, execute the following commands:

   ```
   set sysroot <path>
   target extended-remote tcp:<IP address of adaptive V-ECU>:<port>
   ```

   The `set sysroot` command sets the path of the libraries on the GDB host machine. If no path is specified, GDB looks for the libraries in the local working directory.

   The `target extended-remote` command establishes a TCP connection to the adaptive V-ECU in extended-remote mode. This means that GDB remains connected to the target even when the program to be debugged exits or GDB detaches from it.

4. Once the connection is established, you can start the debugging session.

   When the debugger encounters a breakpoint, the simulation time is paused. It resumes when you prompt the debugger to continue.

Result

You started a remote debugging session for an adaptive V-ECU.

Next steps

You can use GDB in the usual way, for example, to search for functions to debug and to set breakpoints.

When you start the simulation, GDB will stop it at the specified breakpoints, i.e., the simulation time will be paused. To resume, use the `continue` command in GDB.

Related topics

Basics

Basics on Debugging Adaptive V-ECUs..... 244

Working with the VEOS Adaptive Platform Demos

Where to go from here

Information in this section

Introduction to the VEOS Adaptive Platform Demos.....248
The VEOS adaptive platform demos illustrate the handling of adaptive V-ECUs in VEOS and demonstrate the corresponding workflows in SystemDesk and VEOS.

Overview of the Demo Scenarios.....249
There are four demo scenarios that are based on the same configuration: A time server application runs on an adaptive V-ECU, which communicates with a time client application via Ethernet. The scenarios differ in the host of the time client application. The fourth scenario also illustrates remote debugging.

How to Make Preparations for the Demo Simulations.....253
Before you can experiment with the different demo scenarios, you have to make some preparations on the Windows host PC and build the Linux applications on an Ubuntu system.

How to Create, Configure, and Export the Adaptive V-ECUs.....255
To create, configure and export the required adaptive V-ECUs.

How to Build a Simulation System and Run the Demo Simulations.....258
After you have exported the V-ECU implementation containers, you must integrate the V-ECUs into a simulation system in VEOS Player.

Introduction to the VEOS Adaptive Platform Demos

Purpose

The VEOS adaptive platform demos illustrate the handling of adaptive V-ECUs in VEOS and demonstrate the corresponding workflows in SystemDesk and VEOS.

Demonstrated features

The VEOS adaptive platform demos demonstrate the following features of VEOS that are specific to adaptive V-ECUs:

- Simulation of adaptive V-ECUs and communication between them
- Communication between classic and adaptive V-ECUs
- Communication between adaptive V-ECUs and external applications
- Remote debugging an adaptive V-ECU

In addition, the included demo scripts demonstrate automation via the automation interfaces of VEOS Player and SystemDesk.

Demo scripts

The demos contain Python scripts that use the automation interfaces of SystemDesk and VEOS Player to create and configure the required adaptive V-ECUs, integrate them into a simulation system, and run a simulation on Windows.

Technical requirements

To work with the demos, you must have the following:

- Hypervisor software as specified in Hypervisor Requirements for Adaptive V-ECU Simulation on page 239.
- Access to an Ubuntu operating system that includes the following packages:

Package	Description
build-essential	Basic packages required to build C/C++ applications
gdb	GNU Debugger
net-tools	System administration utilities for network configuration

- dSPACE SystemDesk 5.5
- dSPACE VEOS 5.2
- dSPACE licenses for
  - SYD_GEN
  - VEOS_BASE
  - VEOS_ECU
  - VEOS_ETH
  - VEOS_AAP
- GNU Debugger

Related topics

Basics

Basics on Adaptive V-ECUs (SystemDesk Manual 🕮)
Basics on Simulating Adaptive V-ECUs 236
Overview of the Demo Scenarios 249

HowTos

How to Make Preparations for the Demo Simulations 253

Overview of the Demo Scenarios

Introduction

There are four demo scenarios that are based on the same configuration: A time server application runs on an adaptive V-ECU, which communicates with a time client application via Ethernet. The time server application listens for incoming

request messages. Each time the client sends such a request, the server application answers with a time info message.

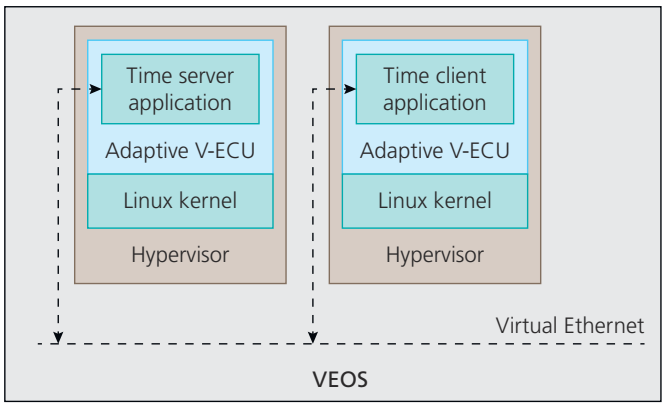
The scenarios differ in the host of the time client application. The fourth scenario also illustrates remote debugging.

To keep it simple and to focus on the handling of adaptive V-ECUs in VEOS, the adaptive V-ECUs of these four demo scenarios are not modeled according to AUTOSAR specifications.

Adaptive V-ECU to adaptive V-ECU (A2A) scenario

The A2A scenario consists of two adaptive V-ECUs, which communicate in VEOS via a virtual Ethernet bus. The virtual Ethernet bus is not connected to the real Ethernet network of the host PC. Therefore, the V-ECUs run in a self-contained simulation system, where VEOS does all the time synchronization. This allows you to run simulations faster or slower than real-time or to pause them.

In this scenario, one adaptive V-ECU acts as a time server, the other as a time client, as shown in the illustration below.



Real Ethernet

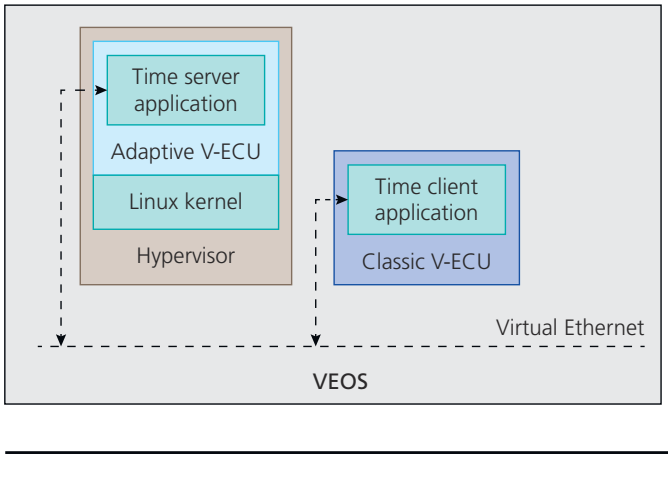
When you run this demo scenario, VEOS boots two virtual machines which simulate the adaptive V-ECUs.

The V-ECUs have the following static IP configurations:

V-ECU	IP Address	Subnet Mask
Time server	192.168.174.10	255.255.255.0
Time client	192.168.174.20	255.255.255.0

Adaptive V-ECU to classic V-ECU (A2C) scenario

In the A2C scenario, the time server application runs on an adaptive V-ECU, while the time client is implemented on a classic V-ECU, as shown in the following illustration.

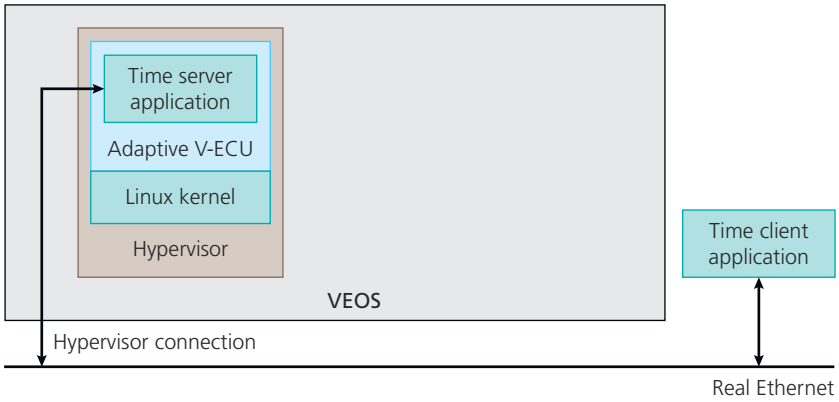


The time client application directly uses the MCAL Ethernet driver of the classic V-ECU. For reasons of simplicity, the classic V-ECU is not compliant with the AUTOSAR Classic Platform specifications. It consists only of a small piece of code on top of the Ethernet driver.

In this example, the adaptive V-ECU binds raw Ethernet sockets, since no TCP/IP Ethernet stack is configured in the classic V-ECU.

Adaptive V-ECU to Windows host PC (A2H) scenario

In the A2H scenario, the time server is implemented on an adaptive V-ECU in VEOS, while the time client application runs directly on the host PC. Therefore, in order to enable communication between the time server application and the time client application, the adaptive V-ECU has to be connected to the host PC via an Ethernet network connection, which is provided by the hypervisor, as shown in the following illustration.



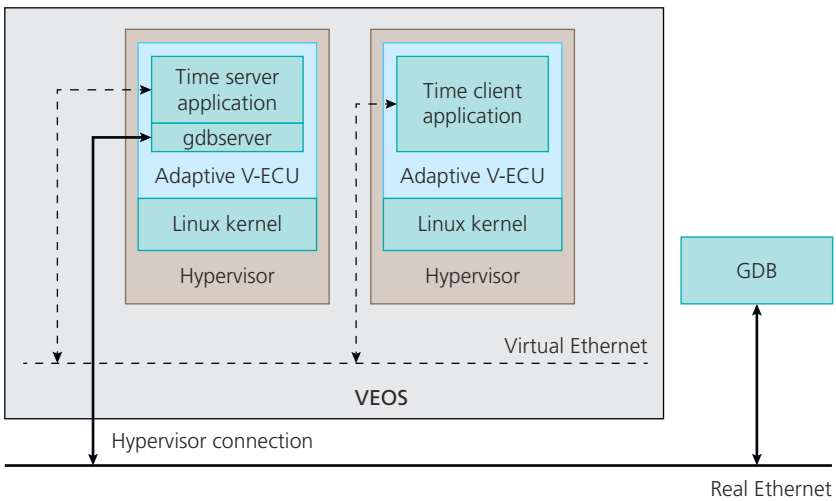
Note

- VEOS only provides time synchronization for the components of the simulation system. Because the time client application is not a part of the simulation system in this case, there is no time synchronization for the Ethernet packages.
- The adaptive V-ECU and the host PC also exchange Ethernet messages, such as ARP and DNS, that are independent of the simulation.

Remote-debugging an adaptive V-ECU (GDB)

This scenario is similar to the A2A scenario in the sense that both the time server application and the time client application run on adaptive V-ECUs in VEOS. However, in contrast to the A2A scenario, the time server application is started by the gdbserver application that enables remote debugging. For remote debugging, the hypervisor must connect the Ethernet controller of the adaptive V-ECU to the Ethernet network of the GNU Debugger (GDB) host.

The following illustration shows a graphical representation of the scenario.



Possible debugging setups are:

- Running a cross GDB on the Windows host PC
- Running GDB on a local Ubuntu VM
- Running GDB on an Ubuntu PC that is connected to the Windows host PC via Ethernet

Related topics

Basics

Simulation Architecture of VEOS 23

HowTos

How to Build a Simulation System and Run the Demo Simulations 258

How to Create, Configure, and Export the Adaptive V-ECUs..... 255
How to Make Preparations for the Demo Simulations..... 253

How to Make Preparations for the Demo Simulations

Objective

Before you can experiment with the different demo scenarios, you have to make some preparations on the Windows host PC and build the Linux applications on an Ubuntu system.

Basics

In VEOS, an adaptive V-ECU consists of one or more applications that run on the dSPACE Linux Kernel. To focus on the workflow, the applications for the demo scenarios are kept deliberately simple. You can modify or replace them to create your own adaptive V-ECUs.

The following workflow shows you how to prepare your PC for the simulation of adaptive V-ECUs and how to build the Linux applications on an Ubuntu system.

Preconditions

To carry out the following instructions, you must have VEOS 5.2 and access to an Ubuntu system as listed in Technical requirements on page 249.

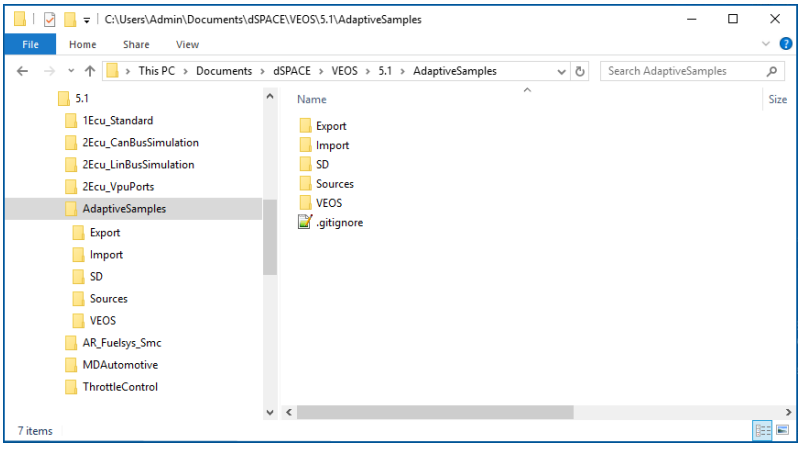
Method

To make preparations for the demo simulations

1 Start VEOS Player and click File - Copy Demo Files.

VEOS copies all demo archives to the Documents folder of your VEOS installation and extracts them.

After extraction, the VEOS adaptive platform demo files are located in the `\AdaptiveSamples` demo folder, as shown in the following illustration:



Tip

If you work with a Git repository, you can use the `.gitignore` file to prevent temporary build artifacts from being tracked.

2 Download the Ubuntu cloud image that corresponds to the version of the Ubuntu system you are using to the `<demo folder>\Import` folder on the host PC.

The following instructions assume that you are using Ubuntu 18.04. You can download the required tarball here:

https://partner-images.canonical.com/core/bionic/current/ubuntu-bionic-core-cloudimg-amd64-root.tar.gz.

It provides the dependencies required for the Linux applications and is used as the first layer in the implementation of the adaptive V-ECUs, as described in Basics on adaptive applications on page 256.

Note

If you use the image file of another Ubuntu version and you want to use the demo scripts, you must change the file name in the `<demo folder>/SD/Scripts/SD_TimeServer_xxx.py` scripts. The file name is stored in the `BionicRootFile` variable.

3 Start the Ubuntu system on which you want to build the applications.

4 Copy the subfolder `\Sources` from the demo folder on the Windows host PC to the Ubuntu system.

5 On the Ubuntu system, open a terminal, change to the `Linux` subfolder of the `Sources` folder you copied in the last step and call `make`. This calls GCC, which compiles the Linux applications and creates five tarballs.

The following table lists the tarballs and their content:

Tarball	File	Function

`timeclient_tcp.tar.gz`
`timeserver_tcp.tar.gz`
`timeclient_raw.tar.gz`[1)]
`timeserver_raw.tar.gz`
	`/bin/timexxxxxx_yyy`[2)]	Executable of the Linux application.

	`/sbin/init`	The initial file that is executed when VEOS starts a simulation. In the demo scenarios, it is a Linux bash script that performs the following steps: ▪ It mounts the `/proc` folder. ▪ It configures a static IP address. ▪ It calls the executable of the Linux application.
	`/sbin/ifconfig`	Linux utility that is required to configure the network interfaces. It has to be copied from the Ubuntu system, because it is not included in the Ubuntu cloud image file.

`timeserver_tcp_gdb.tar.gz`	`/bin/timeserver_tcp`	Linux bash script that starts the renamed time server application `timeserver_tcp_gdb` via gdbserver:
		`#!/bin/sh`
		`/usr/bin/gdbserver :2159 /bin/timeserver_tcp_gdb`

Tarball	File	Function
		It is added in a third layer in the GDB scenario and overrides the time server executable `timeserver_tcp` in the previous layer.
	`/bin/timeserver_tcp_gdb`	Renamed time server application.
	`/bin/gdbserver`	Program that enables remote debugging with the GNU debugger.

[1] This file is not used in the demo scenarios. You can use it to create an adaptive V-ECU that binds raw Ethernet sockets and build a simulation system analogous to the A2C scenario.
[2] The file name corresponds to the name of the tarball.

Note

If you modify or exchange any of the script files contained in the tarballs, ensure that they have Unix line endings (LF). Windows line endings (CR LF) can lead to unexpected results.

The `make` command also creates the `rootfs` subfolder, which contains the contents of the tarballs in correspondingly named subfolders, for example, `rootfs/timeserver_tcp`.

6 Copy all the created tarballs to the `<demo folder>\Import` folder on your host PC.

Result

You have made the necessary preparations for running the VEOS adaptive platform demos.

Next steps

You can now proceed with creating the adaptive V-ECUs. Refer to How to Create, Configure, and Export the Adaptive V-ECUs on page 255.

Related topics

Basics

Basics on Importing V-ECU Implementations..... 65

HowTos

How to Build a Simulation System and Run the Demo Simulations..... 258
How to Create, Configure, and Export the Adaptive V-ECUs..... 255

How to Create, Configure, and Export the Adaptive V-ECUs

Objective

To create, configure, and export the required adaptive V-ECUs.

Basics on adaptive applications

In SystemDesk, you can add tarballs with middleware and adaptive applications to the implementation of an adaptive V-ECU.

VEOS extracts the tarballs in the order in which they are layered in the V-ECU implementation. If you configured multiple layers that include files with identical names and paths, each of these layers overwrites the previous one.

Precondition

You must have made the preparations described in How to Make Preparations for the Demo Simulations on page 253.

Possible methods

There are two ways to create, configure, and export the adaptive V-ECUs required for the demos:

- You can perform the necessary steps manually. Refer to Method 1 on page 256.
- You can use the Python demo scripts for automation. Refer to Method 2 on page 258.

Method 1

To create, configure, and export the adaptive V-ECUs manually

1. Start SystemDesk.
2. Create a new project TimeServer_xxx.sdp in the `\AdaptiveSamples\SD` folder, where xxx stands for the demo scenario shortcut, such as A2A.
3. On the V-ECU ribbon, select V-ECU – New.

   In the Create V-ECU dialog, select Empty adaptive V-ECU and click Finish to confirm.

   The following table lists the adaptive V-ECUs you have to create for each scenario:

Scenario	V-ECUs
Adaptive V-ECU to adaptive V-ECU (A2A) scenario	timeserver_tcp timeclient_tcp
Adaptive V-ECU to classic V-ECU (A2C) scenario	timeserver_raw[1)]
Adaptive V-ECU to Windows host PC (A2H) scenario	timeserver_tcp
Remote-debugging an adaptive V-ECU (GDB)	timeserver_gdb timeclient_tcp

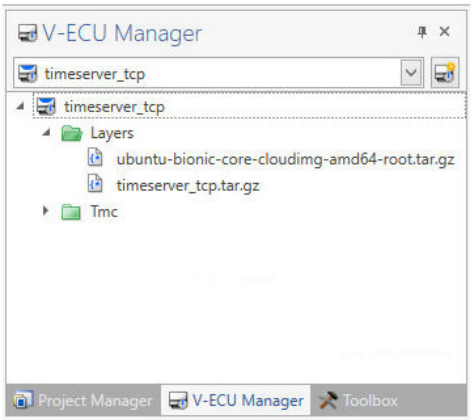
   [1)] The classic V-ECU for this scenario, timeclient_raw, is already included in the `<demo folder>\Sources\ClassicVEcu` folder.

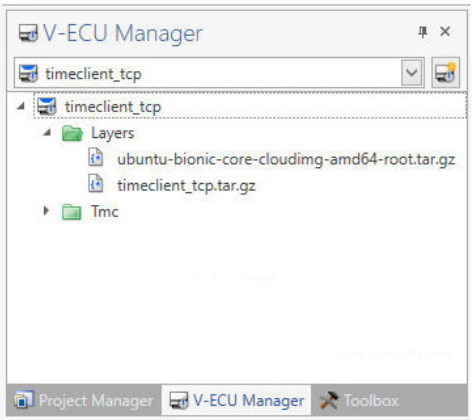
4. Add the Ubuntu cloud image and the tarballs you created in step 5 of How to Make Preparations for the Demo Simulations on page 253 to the Layers folder of the adaptive V-ECUs in the order listed in the following table:

V-ECU	Tarballs	
timeserver_tcp	First layer	`Import\<Ubuntu cloud image>`
	Second layer	`Import\timeserver_tcp.tar.gz`
timeserver_raw	First layer	`Import\<Ubuntu cloud image>`
	Second layer	`Import\timeserver_raw.tar.gz`
timeserver_gdb	First layer	`Import\<Ubuntu cloud image>`
	Second layer	`Import\timeserver_tcp.tar.gz`
	Third layer	`Import\timeserver_tcp_gdb.tar.gz`[1]
timeclient_tcp	First layer	`Import\<Ubuntu cloud image>`
	Second layer	`Import\timeclient_tcp.tar.gz`

[1] Adding this tarball as the third layer causes the time server application of the second layer to be overwritten by a debuggable version that is started by gdbserver.

The following illustration shows the adaptive V-ECUs of the A2A scenario in the V-ECU Manager:





5 Configure the Tmc module as described in the following table:

Property	Configuration
TmcGeneral	TmcCoreCount: 1 TmcMemorySize: 1024 TmcTickDuration: 0.01
TmcNetworking - TmcEthernetController[1]	TmcEthControllerIdx: 0 TmcEthMacLayerType: TMC_ETH_MAC_LAYER_TYPE_XGMII TmcEthPhysicalAddress: ▪ timeserver_tcp/raw/gdb: 00:11:22:33:44:55 ▪ timeclient_tcp: 00:11:22:33:44:66

[1] You have to add a new instance of TmcEthernetController first by selecting New from the context menu of the TmcNetworking node.

6 Export each adaptive V-ECU by selecting it in the V-ECU Manager and clicking V-ECU - Export on the V-ECU ribbon.

In the V-ECU Implementation Container Export dialog, navigate to the `\AdaptiveSamples\Export` folder and click Save.

SystemDesk exports the adaptive V-ECU as a V-ECU implementation container, i.e., VECU file, to the `\AdaptiveSamples\Export` folder.

Method 2

To create, configure and export the adaptive V-ECUs via automation

1. Start PythonWin or another Python interpreter of your choice.
2. In the Python interpreter, run the `SD_TimeServer_xxx.py` scripts in the `\AdaptiveSamples\SD\Scripts` folder to create the adaptive V-ECUs for the different demo scenarios as listed in step 3 on page 256.

   Alternatively, run the `SD_TimeServer.py` script, which calls all the other scripts.

Result

You have created and configured the adaptive V-ECUs required by the demo scenarios and exported them to the `\AdaptiveSamples\Export` folder on your host PC.

Next steps

You can now integrate the V-ECUs into a simulation system in VEOS Player. Refer to How to Build a Simulation System and Run the Demo Simulations on page 258.

Related topics

Basics

Automating SystemDesk (SystemDesk Manual 🕮)
Creating Adaptive V-ECUs (SystemDesk Manual 🕮)
Exporting V-ECU Implementations (SystemDesk Manual 🕮)
Overview of the Demo Scenarios..... 249

References

New V-ECU (SystemDesk Manual 🕮)

How to Build a Simulation System and Run the Demo Simulations

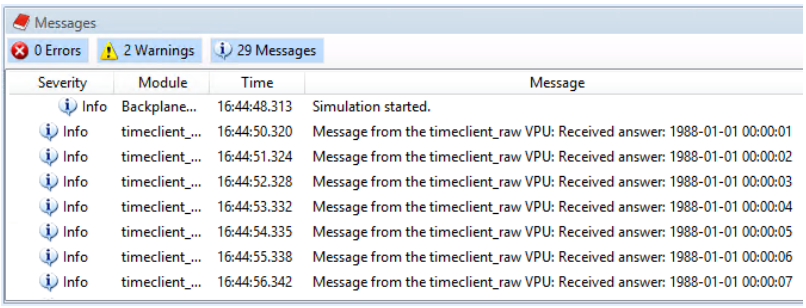
Objective

After you have exported the V-ECU implementation containers, you must integrate the V-ECUs into a simulation system in VEOS Player.

Basics on V-ECU messages

The demo applications are configured to generate messages for the user. Where these messages are displayed depends on the type of the V-ECU:

- Messages from the classic V-ECU are displayed directly in the VEOS Player **Messages** pane, as shown in the following illustration.



- The Linux applications running on the virtual machines produce standard output (`stdout`) and standard error (`stderr`) messages. These messages are redirected to external OUT files that are linked in the VEOS Player **Messages** pane.

  The following illustration shows an excerpt from a time server OUT file.

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::211:22ff:fe33:4455  prefixlen 64  scopeid 0x20<link>
        ether 00:11:22:33:44:55  txqueuelen 20000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

Wait for requests
Send answer
Send answer
Send answer
Send answer
Send answer
Send answer
Send answer
Send answer
Send answer
Send answer
```

Preconditions

You must have created, configured, and exported the required adaptive V-ECUs. Refer to How to Create, Configure, and Export the Adaptive V-ECUs on page 255.

Possible methods

There are two ways to build the simulation systems and run the demo simulations:

- You can perform the required steps manually. Refer to Method 1 on page 260.
- You can use the Python demo scripts for automation. Refer to Method 2 on page 264.

Note

The instructions and demo scripts are only valid for VEOS on Windows.

Method 1

To build a simulation system in VEOS Player and run the demo simulations manually

1 Start VEOS Player.

2 On the Home ribbon, click New to create a new offline simulation application. Save it as TimeServer_xxx.osa, where xxx stands for one of the four scenario shortcuts.

3 Use drag & drop to add V-ECU implementation containers to the simulation system.

VEOS opens the Import dialog. On the Hypervisor Options page, set the Hypervisor connection according to the following table. Leave the other options at their predefined values.

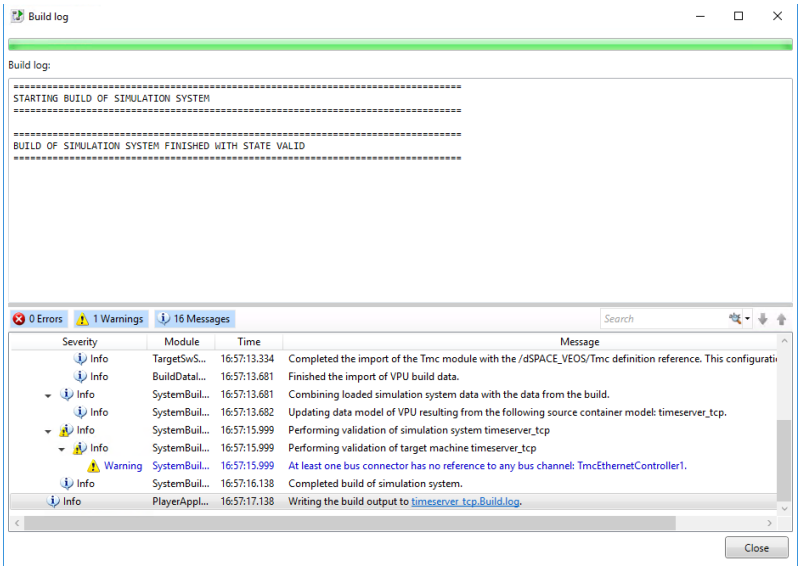
Simulation System	Adaptive V-ECU Implementation Containers	Hypervisor Connection
TimeServer_A2A	`Export\timeserver_tcp.vecu` `Export\timeclient_tcp.vecu`	Disconnected Disconnected
TimeServer_A2C	`Export\timeserver_raw.vecu` `Sources\ClassicVEcu\timeclient_raw.vecu`	Disconnected not applicable
TimeServer_A2H	`Export\timeserver_tcp.vecu`	HostOnly[1)]
TimeServer_GDB	`Export\timeserver_gdb.vecu` `Export\timeclient_tcp.vecu`	HostOnly[2)] or Bridged[3)] Disconnected **Note** Consult your IT administrator before setting Hypervisor connection to Bridged.

1) The HostOnly connection is established via the virtual network adapter provided by the hypervisor. As long as the hypervisor connection is set to HostOnly, a virtual machine is connected to the host PC via a private Ethernet network and cannot access the physical Ethernet of the host PC.

2) If you run a cross GDB on your Windows host PC or if you run GDB on a local Ubuntu VM.

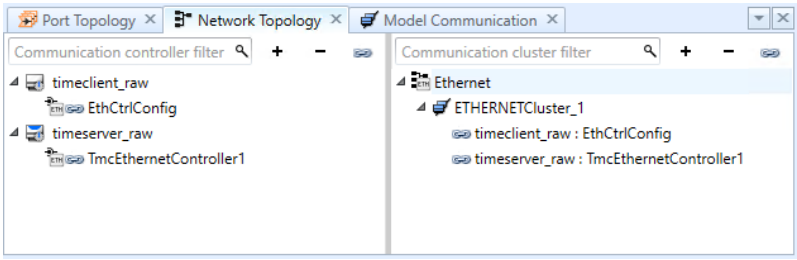
3) If you run GDB on an Ubuntu PC that is connected to the Windows host PC via a physical Ethernet connection. If you use the Hyper-V hypervisor, you have to adjust some settings in the Hyper-V Manager first. Otherwise GDB is not able to connect to the gdbserver application. Refer to Hypervisor Connection on page 156.

Each time you add a V-ECU implementation container, VEOS builds the corresponding V-ECU implementation. After the build has completed, you are shown the following message:



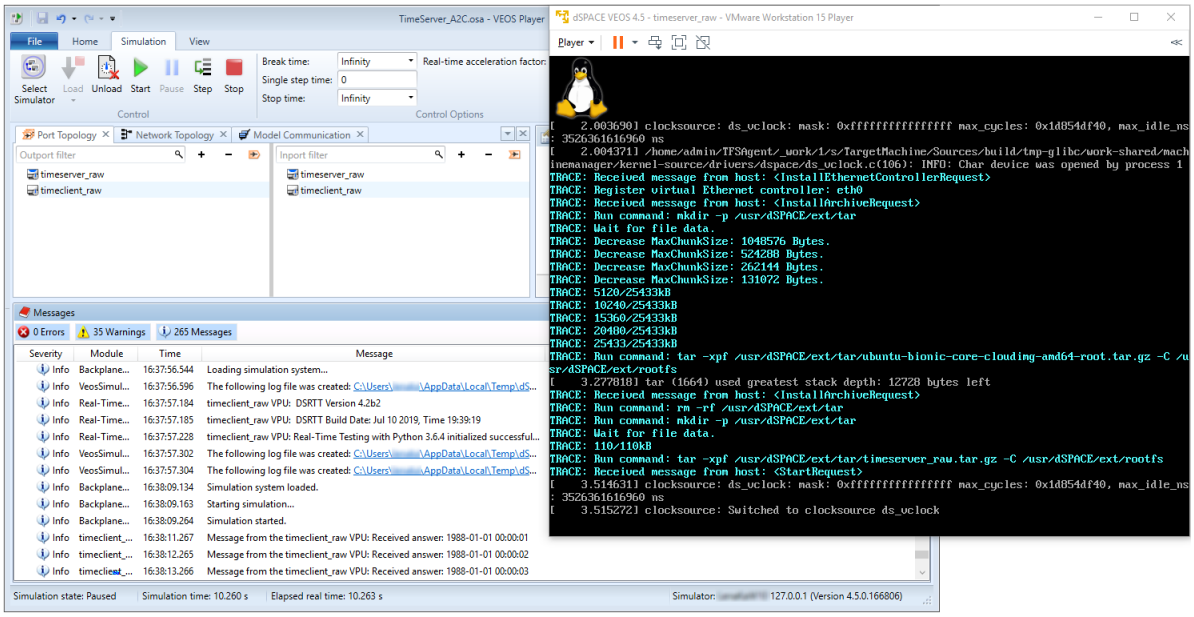
4 After you added all required V-ECUs, click **Home - Autoconnect** to create an Ethernet communication cluster and connect the Ethernet controllers of the V-ECUs to it.

VEOS displays the bus simulation configuration in the **Network Topology** pane, as shown in the following illustration for the A2C scenario:



5 Click **Simulation - Start** to start the simulation.

VEOS boots a Linux virtual machine for each adaptive V-ECU. The following illustration shows the simulation of the A2C scenario with the VMware Workstation hypervisor.



Messages from the classic V-ECU are displayed directly in the VEOS Player **Messages** pane, while messages from the adaptive V-ECU are redirected to an OUT file, as described in Basics on V-ECU messages on page 259.

If you are working with the Hyper-V hypervisor, a window for each virtual machine is not automatically visible. You can use the Hyper-V Manager to display the virtual machines that simulate the adaptive V-ECUs.

Note

In the GDB scenario, the simulation pauses almost immediately after it starts and waits for a remote debugger to attach. The simulation resumes after the debugger has attached. Depending on the settings of the debugger, you might have to prompt the simulation to continue.

6 The A2H and the GDB scenario require additional steps:

- 1. Set the static IPv4 address of the corresponding virtual network adapter to 192.168.174.100 and the subnet mask to 255.255.255.0.

  If you work with the Hyper-V hypervisor:

  Use the vEthernet (dSPACE_VEOS_5.2_Switch_Internal) virtual network adapter. For the GDB scenario with GDB running on an external PC (connection type Bridged), refer to Hypervisor Connection on page 156 for information on how to properly configure the virtual network adapter.

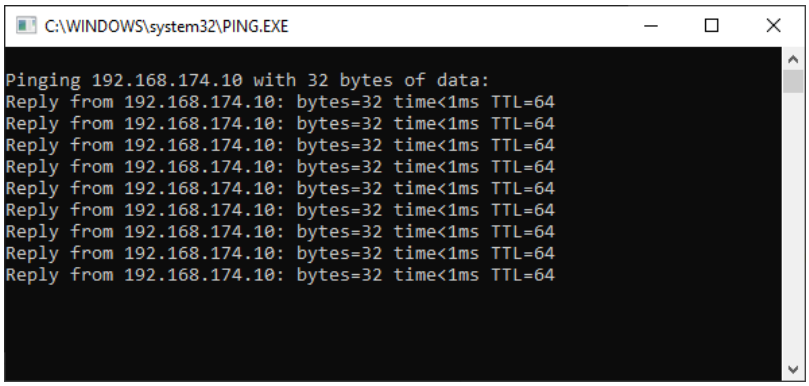
  If you work with the VMware Workstation hypervisor:

  Use the VMnet1 virtual network adapter for the A2H scenario and for the GDB scenario with GDB running on the host PC or on a local VM. For the GDB scenario with GDB running on an external PC, use VMnet0.

  2. To confirm that a connection to the adaptive V-ECU can be established, open a Command Prompt window on the Windows host PC and execute the following command:

```
start ping -t 192.168.174.10
```

A successful connection generates the following output:



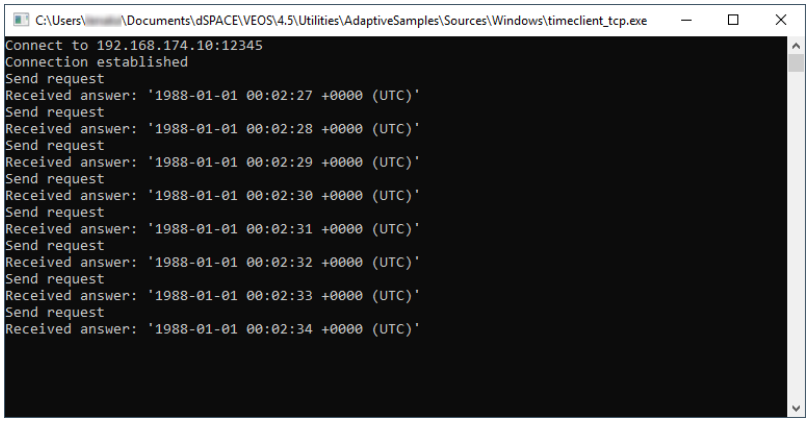
- A2H scenario:

  In another Command Prompt window on the Windows host PC, execute the following command:

  ```
  start <demo folder>/Sources/Windows/timeclient_tcp.exe
  192.168.174.10
  ```

  This command starts the Windows time client application. The time client application establishes an Ethernet connection to the time server application running in the adaptive V-ECU and requests time information.

  The following illustration shows the expected output of the time client application.



- GDB scenario:

  Extract the `timeserver_tcp_gdb` executable from the `timeserver_tcp_gdb.tar.gz` archive in the `<demo folder>\Import` folder and copy it to the GDB host system.

  Start GDB. The start command must include as the first parameter the path to the local copy of the application to be debugged. The name of the GDB executable depends on your GDB host system:

  - Cross GDB on a Windows machine:

    In a Command Prompt window, type

```
x86_64-linux-gnu-gdb.exe <path to timeserver_tcp_gdb
executable>
```

- GDB on a Linux machine:

  In a terminal, type

  ```
  gdb <path to timeserver_tcp_gdb executable>
  ```

In GDB, execute the following commands:

```
set sysroot
target extended-remote tcp:192.168.174.10:2159
```

The `set sysroot` command sets the path of the libraries on the GDB host machine, in this case, the local working directory.

The `target extended-remote tcp:192.168.174.10:2159` command establishes a TCP connection to the target, i.e., the adaptive V-ECU, in extended-remote mode. This means that GDB remains connected to the target even when the program to be debugged exits or GDB detaches from it.

Tip

You can also use an IDE that supports remote debugging with GDB, such as Eclipse.

After the connection is established, you can start the debugging session. When the debugger encounters a breakpoint, the simulation time is paused. It resumes when you prompt the debugger to continue.

7 To end the simulation, click Simulation - Stop. This restarts the virtual machines that host the adaptive V-ECUs and then keeps them powered on. They are only shut down when you unload the simulation system by clicking Simulation - Unload.

Method 2

To build a simulation system in VEOS Player and run the demo simulations via automation

1 Start PythonWin or another Python interpreter of your choice.

2 In the Python interpreter, run the `VEOS_TimeServer_xxx.py` scripts in the `\AdaptiveSamples\VEOS\Scripts` folder to create and configure simulation systems for the different demo scenarios.

The scripts use the VEOS Player automation interface to perform the steps detailed in Method 1 on page 260.

3 For the GDB scenario, refer to step 6 on page 262 after the simulation has started.

The scripts for the A2H and GDB include the continuous `ping` command described in 2 on page 262.

Result

You set up and ran the demo simulations in VEOS.

Next steps

You can use ControlDesk's Bus Navigator to monitor Ethernet communication between the time server and time client applications. To do this, you have to register VEOS as a platform in ControlDesk, create a new project and experiment, and add an Ethernet Bus Monitoring device to the experiment. For details on bus monitoring, refer to How to Monitor Bus Communication (ControlDesk Bus Navigator 🕮).

Related topics

Basics

Automating the VEOS Player..... 325
Overview of the Demo Scenarios..... 249

HowTos

How to Import V-ECU Implementations..... 67
How to Load and Run an Offline Simulation Application..... 188

References

Autoconnect..... 138
Import..... 152

Working with the AUTOSAR Adaptive Platform Demonstrator Samples

Where to go from here

Information in this section

Basics on the AUTOSAR Adaptive Platform Demonstrator.....266
AUTOSAR provides its members with software repositories for building sample adaptive applications that simulate typical adaptive platform functionalities such as radar fusion.

How to Build AUTOSAR Adaptive V-ECUs and Integrate them into an Offline Simulation Application.....267
To build adaptive V-ECUs in SystemDesk using the AUTOSAR Adaptive Demonstrator radar and fusion sample images and to integrate the adaptive V-ECUs into an offline simulation application in VEOS.

Basics on the AUTOSAR Adaptive Platform Demonstrator

Introduction

AUTOSAR provides its members with software repositories for building sample adaptive applications that simulate typical adaptive platform functionalities such as radar fusion.

Organization of the AUTOSAR Adaptive Platform Demonstrator sources

AUTOSAR uses the Yocto Project's layered architecture for building customized Linux kernels and adaptive applications.

You can access the AUTOSAR Adaptive Platform Demonstrator sources and instructions for building from them under https://svn.autosar.org/repos/work/26_Standards/02_Releases/30_AP/R19-11/APDemonstrator/AUTOSAR_SRC_AdaptivePlatformDemonstrator.zip and https://svn.autosar.org/repos/work/26_Standards/02_Releases/30_AP/R19-11/APDemonstrator/AUTOSAR_SRC_AdaptivePlatformDemonstratorTestData.zip.

Sample applications

The AUTOSAR Adaptive Platform Demonstrator contains sample applications and test data for simulating a simple radar fusion scenario where radar is provided as a service.

The demo implementation of the radar fusion scenario consists of two adaptive V-ECUs which communicate via Ethernet, similarly to the A2A scenario described in Working with the VEOS Adaptive Platform Demos on page 248.

Each adaptive V-ECU consists of the following layers:

- An AUTOSAR image containing the root file system and adaptive applications.
- A `netinit` script for the configuration of the network interface.

Supported AUTOSAR releases

Refer to AUTOSAR Adaptive Platform compatibility on page 43.

Related topics

Basics

Working with the VEOS Adaptive Platform Demos..... 248

HowTos

How to Build AUTOSAR Adaptive V-ECUs and Integrate them into an Offline Simulation Application..... 267

How to Build AUTOSAR Adaptive V-ECUs and Integrate them into an Offline Simulation Application

Objective

To build adaptive V-ECUs in SystemDesk using the AUTOSAR Adaptive Demonstrator radar fusion sample images and to integrate the adaptive V-ECUs into an offline simulation application in VEOS.

Preconditions

- You must have built the AUTOSAR Adaptive Platform Demonstrator samples according to the instructions provided by AUTOSAR. Refer to Basics on the AUTOSAR Adaptive Platform Demonstrator on page 266.
- You must meet the software requirements specified in Introduction to the VEOS Adaptive Platform Demos on page 248, except for the Ubuntu system requirements.

Part 1

To build AUTOSAR adaptive V-ECUs in SystemDesk

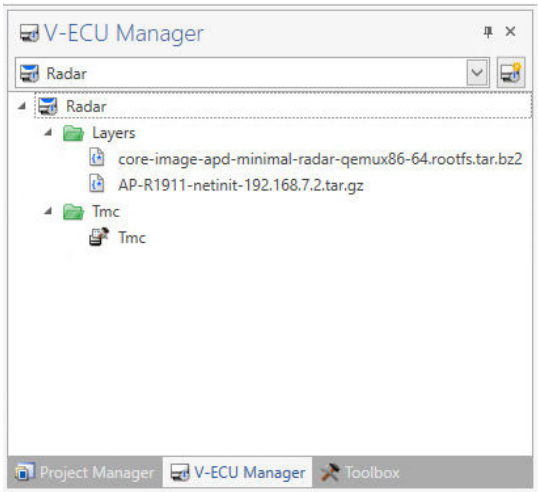
1. Create a new SystemDesk project.
2. In the new project, create two adaptive V-ECUs as listed in the following table:

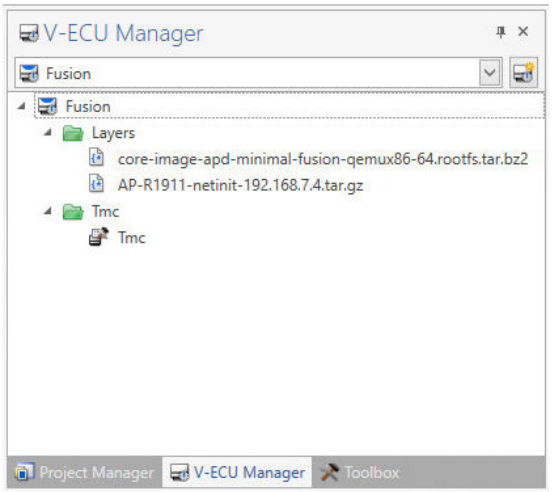
V-ECU	Tarballs	
Radar	First layer	`core-image-apd-minimal-radar-qemux86-64.rootfs.tar.bz2`
	Second layer	`AP-Rxxxx-netinit-192.168.7.2.tar.gz`[1)]

V-ECU	Tarballs	
Fusion	First layer	`core-image-apd-minimal-fusion-qemux86-64.rootfs.tar.bz2`
	Second layer	`AP-Rxxxx-netinit-192.168.7.4.tar.gz`[1]

[1] Replace `xxxx` by the AUTOSAR release you are using.

The following illustration shows the resulting configurations in the V-ECU Manager:





3 Configure the Tmc modules of the adaptive V-ECU as follows:

Property	Configuration
TmcGeneral	TmcCoreCount: 1 TmcMemorySize: 1024 TmcTickDuration: 0.01
TmcNetworking - TmcEthernetController[1]	TmcEthControllerIdx: 0 TmcEthMacLayerType: TMC_ETH_MAC_LAYER_TYPE_XGMII TmcEthPhysicalAddress: ▪ Radar: 00:11:22:33:44:55 ▪ Fusion: 00:11:22:33:44:66

[1] You have to add a new instance of TmcEthernetController first by selecting New from the context menu of the TmcNetworking node.

4 Export the adaptive V-ECUs to the `\AdaptiveSamples\Export` folder, for example.

SystemDesk exports the adaptive V-ECUs as VECU container files.

Interim result

You have built the AUTOSAR adaptive V-ECUs of the radar fusion example in SystemDesk.

Part 2

To integrate the adaptive V-ECUs into an offline simulation application in VEOS

1 In VEOS, create a new offline simulation application.

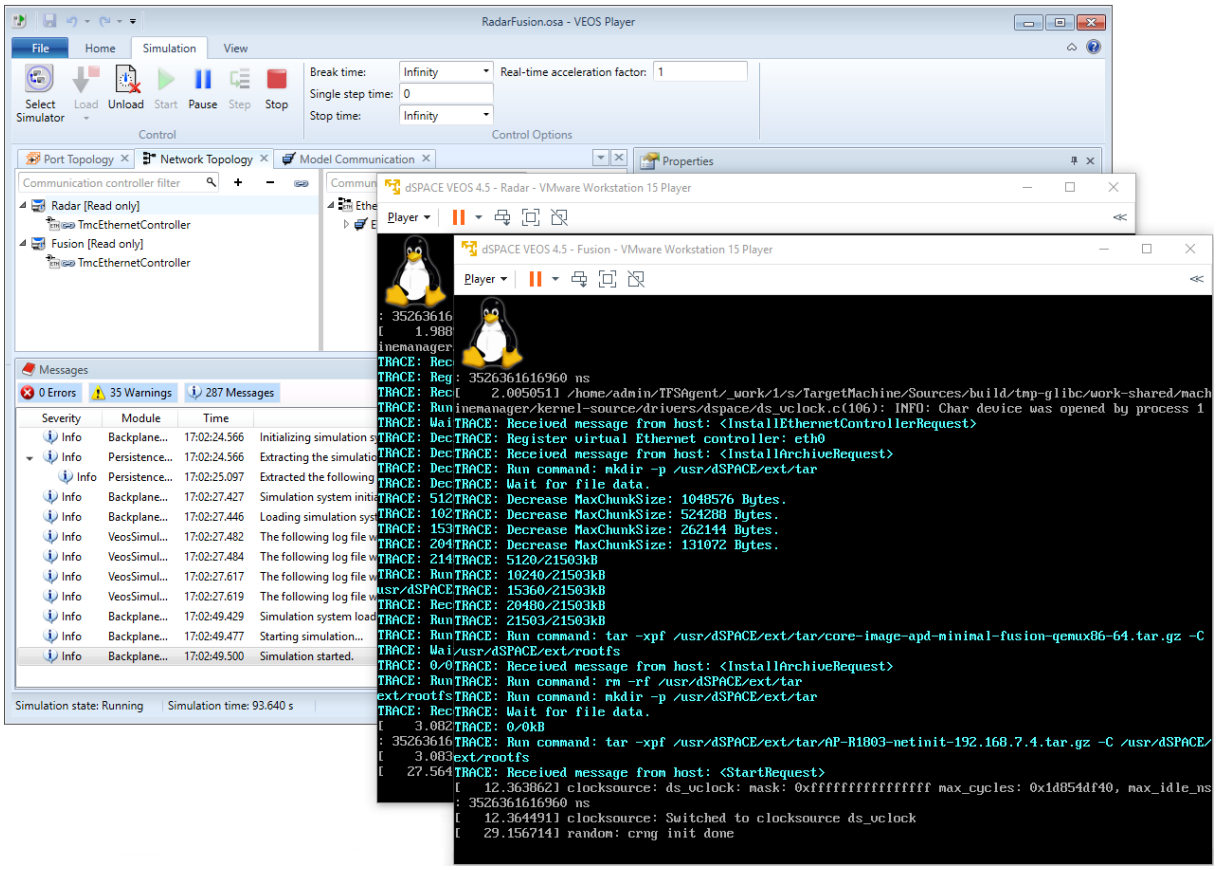
2 Add the V-ECU implementation containers you exported in part 1 to the simulation system.

On the Hypervisor Options page of the Import dialog, set Hypervisor connection to Disconnected.

3 When you have imported the V-ECU implementation containers, click Home - Autoconnect to create an Ethernet communication cluster and connect the Ethernet controllers of the V-ECUs to it.

4 Start the simulation.

VEOS boots a Linux virtual machine for each adaptive V-ECU, as shown in the following illustration.



Note

If you are using the Hyper-V hypervisor, the virtual machines are not displayed.

Result

You have built AUTOSAR adaptive V-ECUs for the radar fusion example in SystemDesk and integrated them into an offline simulation application in VEOS.

Next steps

You can use ControlDesk's Bus Navigator to monitor Ethernet communication between the radar and fusion applications. To do this, you have to register VEOS as a platform in ControlDesk, create a new project and experiment, and add an Ethernet Bus Monitoring device to the experiment. For details on bus monitoring, refer to How to Monitor Bus Communication (ControlDesk Bus Navigator 🕮).

Related topics

Basics

Basics on the AUTOSAR Adaptive Platform Demonstrator..... 266
Creating Adaptive V-ECUs (SystemDesk Manual 🕮)
Exporting V-ECU Implementations (SystemDesk Manual 🕮)

HowTos

How to Import V-ECU Implementations..... 67
How to Load and Run an Offline Simulation Application..... 188

References

Autoconnect..... 138
Import..... 152
New V-ECU (SystemDesk Manual 🕮)

Working with the GPU Passthrough Demo

Where to go from here

Information in this section

Introduction to the GPU Passthrough Demo.....271
VEOS on Linux lets you use the graphical processing unit (GPU) of the host PC in the simulation of adaptive V-ECUs. This requires passing the GPU through to virtual machines.
The GPU passthrough demo shows you how to enable GPU passthrough and provides an example based on the NVIDIA CUDA fluid simulation.

Overview of the GPU Passthrough Demo.....272
The demo comprises three different implementations of the NVIDIA CUDA fluid simulation: Simple text output on the console, graphical output via GTK3, and separation of simulation and graphical output into a server and a client application.

How to Prepare the Host PC for Running the Fluid Simulation.....276
To prepare the host PC for running the fluid simulation.

How to Run the Fluid Simulation on the Host PC.....278
To run the fluid simulation on the host PC.

How to Enable GPU Passthrough.....281
To enable GPU passthrough so that a virtual machine can access the GPU of the host PC.

How to Start a Virtual Machine with GPU Passthrough.....283
To start a virtual machine with GPU passthrough using the DSKVM hypervisor.

How to Run the Fluid Simulation with VEOS.....285
To run the fluid simulation with VEOS.

Introduction to the GPU Passthrough Demo

Purpose

VEOS on Linux lets you use the graphical processing unit (GPU) of the host PC in the simulation of adaptive V-ECUs. This requires passing the GPU through to virtual machines.

The GPU passthrough demo shows you how to enable GPU passthrough and provides an example based on the NVIDIA CUDA fluid simulation.

Required knowledge

To follow the instructions below you require the following knowledge:

- Advanced Linux knowledge, in particular concerning drivers, devices, and virtualization

- Basic familiarity with GPUs
- Familiarity with the simulation architecture of adaptive V-ECUs in VEOS.

  You are recommended to work through the VEOS Adaptive Platform Demos (refer to Working with the VEOS Adaptive Platform Demos on page 248) before starting with the GPU passthrough demo.

Technical requirements

To work with the GPU passthrough demo, you must have:

- A Linux host PC that is equipped with the following items:
  - An Intel CPU and an NVIDIA GPU with support for CUDA 10.2 and virtualization
  - Access to the Internet or a repository that provides the following software:
    - The NVIDIA graphics driver installer

      `NVIDIA-Linux-x86_64-450.80.02.run`
    - The CUDA 10.12 platform installer

      `cuda_10.2.89_440.33.01_linux.run`
  - VEOS 5.2 with the AUTOSAR Adaptive Platform
  - The demo applications
- A Windows host PC with VEOS 5.2 for building an offline simulation application

Obtaining the demo applications

To obtain the demo applications, contact dSPACE Support.

Related topics

Basics

Overview of the GPU Passthrough Demo..... 272
Working with the VEOS Adaptive Platform Demos..... 248

HowTos

How to Prepare the Host PC for Running the Fluid Simulation..... 276

Overview of the GPU Passthrough Demo

Introduction

The demo comprises three different implementations of the NVIDIA CUDA fluid simulation:

- Simple text output on the console
- Graphical output via GTK3
- Separation of simulation and visualization into a server and a client application

The following instructions use these different implementations to explain GPU passthrough in three steps:

1. To familiarize yourself with the three implementations of the simulation, you will execute them on the host PC without GPU passthrough.

   Refer to Running the fluid simulation on the host PC on page 274.

2. You will then implement a simple GPU passthrough scenario using the DSKVM hypervisor. This illustrates the processes that are involved when the GPU is passed through to an adaptive V-ECU in a VEOS simulation.

   Refer to Starting a virtual machine with GPU passthrough on page 274.

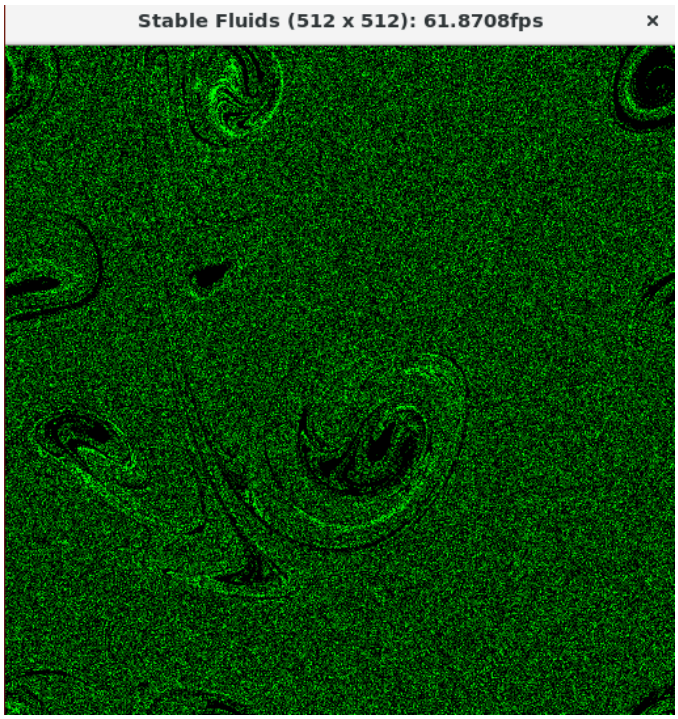
3. Finally, you will run the fluid simulation in VEOS using adaptive V-ECUs.

   Refer to Running the fluid simulation in VEOS on page 275.

Basics on the NVIDIA CUDA fluid simulation

The NVIDIA CUDA fluid simulation, which is delivered with the CUDA 10.2 platform, uses the GPU to solve the Navier-Stokes differential equations for simulating particle motions in a fluid. If it is implemented with graphical output, the user can interact with the simulation by pushing particles with the cursor.

Refer to the following illustration.



Modifications The demo implementation for VEOS contains the following modifications:

- The `fluidsGtk3` application for graphical output uses GTK3 instead of OpenGL.
- The `fluidsConsole` application lets you run the fluid simulation in a console application without graphical output.

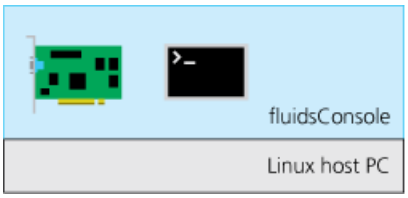
- The demo also includes an implementation in two separate applications for simulation and graphical output, i.e., `fluidsServer` and `fluidsClientGtk3`, which communicate over TCP.

For more information on the NVIDIA CUDA fluid simulation, refer to http://developer.download.nvidia.com/compute/DevZone/C/html/C/src/fluidsGL/doc/fluidsGL.pdf.

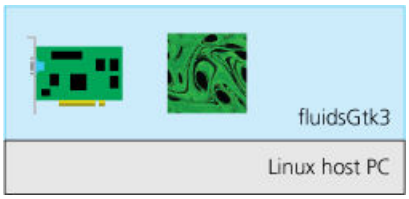
Running the fluid simulation on the host PC

You can run the fluid simulation on the host PC in three different ways:

- In the console without graphical output:



- With graphical output via GTK3:



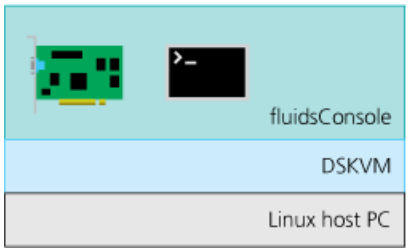
- In a server application, which communicates with a client application over TCP. The client application provides the graphical output.

Tip

For testing purposes, there are also dummy implementations which do not require a GPU: `dummyConsole`, `dummyGtk3`, and `dummyServer`. These dummy implementations run a simplified simulation on the CPU.

Starting a virtual machine with GPU passthrough

This example consists of a virtual machine hosted by the DSKVM hypervisor, where GPU passthrough is enabled. To keep the virtual machine small, the disk image includes only the GPU driver, not the fluid simulation.



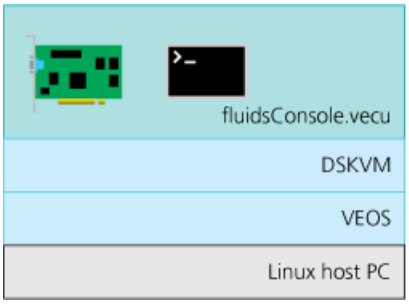
Running the fluid simulation in VEOS

In VEOS, the fluid simulation is performed by an adaptive V-ECU. Because an adaptive V-ECU runs on a virtual machine, it requires GPU passthrough.

You can run the fluid simulation in VEOS in two ways:

- Without graphical output, using the `fluidsConsole` adaptive V-ECU.

  Refer to the following illustration.



- With graphical output redirected to the `fluidsClientGtk3` application on the host PC.

  In this variant, the actual simulation is performed by the `fluidsServer` adaptive V-ECU, which communicates via TCP with the `fluidsClientGtk3` application on the host PC.

  Refer to the following illustration.

Related topics

Basics

Hypervisor Configuration for Adaptive V-ECU Simulation..... 296

Introduction to the GPU Passthrough Demo..... 271

HowTos

How to Enable GPU Passthrough..... 281
How to Prepare the Host PC for Running the Fluid Simulation..... 276
How to Run the Fluid Simulation on the Host PC..... 278
How to Run the Fluid Simulation with VEOS..... 285
How to Start a Virtual Machine with GPU Passthrough..... 283

How to Prepare the Host PC for Running the Fluid Simulation

Objective

To prepare the host PC for running the fluid simulation.

The preparation consists of the following parts:

- Installing the required NVIDIA software. Refer to Part 1 on page 276.
- Rebuilding and updating the VEOS Linux kernel. Refer to Part 2 on page 277.
- Determining the PCI bus address and device code of the GPU and assigning the values to environment variables. Refer to Part 3 on page 277.
- Building the demo applications. Refer to Part 4 on page 277.

Preconditions

You must have a Linux host PC with the following:

- VEOS 5.2 with the AUTOSAR Adaptive Platform
- Access to the NVIDIA software listed in Introduction to the GPU Passthrough Demo on page 271

Part 1

To install the NVIDIA software

1. Create a folder for the NVIDIA sources in your home directory and change to it:

```
mkdir -p ~/veos-demos/nvidia
cd ~/veos-demos/nvidia
```

2. Download the installers of the NVIDIA GPU driver and the CUDA platform to the new folder:

```
wget https://de.download.nvidia.com/XFree86/Linux-x86_64/450.80.02/\
NVIDIA-Linux-x86_64-450.80.02.run

wget http://developer.download.nvidia.com/compute/cuda/10.2/Prod/\
local_installers/cuda_10.2.89_440.33.01_linux.run
```

3. Run the installer shell scripts:

```
sudo sh NVIDIA-Linux-x86_64-450.80.02.run
sudo sh cuda_10.2.89_440.33.01_linux.run
```

Interim result

You installed the required NVIDIA GPU driver and the CUDA platform.

Part 2

To rebuild and update the VEOS Linux kernel

1 Create a temporary folder for extracting the dSPACE VEOS Linux kernel from the AUTOSAR Adaptive Platform archive:

```
mkdir -p ~/veos-demos/veos-linux
```

2 Extract the AUTOSAR Adaptive Platform sources to the new folder:

```
tar -xf /opt/dspace/veosautosaradaptiveplatform5.2/Sources/\
AutosarAdaptivePlatform-sources.tar.gz -C ~/veos-demos/veos-linux
```

3 Rebuild the VEOS Linux kernel with the following command:

```
make -C ~/veos-demos/veos-linux/meta-veos/recipes-kernel/linux/files
```

4 Update the VEOS Linux kernel with the following command:

```
sudo make install -C ~/veos-demos/veos-linux/meta-veos/\
recipes-kernel/linux/files
```

Interim result

You rebuilt and updated the VEOS Linux kernel.

Part 3

To determine the PCI bus address and device code of the GPU

1 Use the following command to list all the VGA-compatible controllers on the PCI bus:

```
lspci -nn | grep -i vga
```

Refer to the following illustration, where the relevant information is framed in blue:

```
00:02.0 VGA compatible controller [0300]: Intel Corporation Device [8086:3e92]
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK107GL [Quadro K600]
[10de:0ffa] (rev a1)
```

2 Assign the PCI bus address and and the device code of the GPU to environment variables for later use:

```
export PCI_HOST_BUS_ADDRESS="01:00.0"
export PCI_DEVICE_CODE="10de:0ffa"
```

Interim result

You determined the PCI bus address and device code of the GPU and assigned the values to environment variables.

The environment variables are required by the makefile for building the demo applications in the next part and later on for enabling GPU passthrough.

Part 4

To build the demo applications

1 Extract the demo applications to the `~/veos-demos` folder you created previously:

```
tar -xf /opt/dspace/veosautosaradaptiveplatform5.2/Demos/\
fluidsDemos.tar.gz -C ~/veos-demos
```

2 Build the demo applications:

```
make -C ~/veos-demos/fluidsDemos
```

Result

You prepared the host PC for running the fluid simulation by installing the required NVIDIA software, rebuilding and updating the VEOS Linux kernel, and building the demo applications.

Related topics

Basics

Introduction to the GPU Passthrough Demo..... 271

HowTos

How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS..... 292
How to Run the Fluid Simulation on the Host PC..... 278
How to Run the Fluid Simulation with VEOS..... 285
How to Start a Virtual Machine with GPU Passthrough..... 283

How to Run the Fluid Simulation on the Host PC

Objective

To run the fluid simulation on the host PC.

Precondition

You must have made the preparations described in How to Prepare the Host PC for Running the Fluid Simulation on page 276.

Possible methods

You can run the fluid simulation on the host PC in three different ways:

- Via the command line without graphical output. Refer to Method 1 on page 278.
- With graphical output via GTK3. Refer to Method 2 on page 279.
- Via a client and a server application. Refer to Method 3 on page 280.

Method 1

To run the fluid simulation in the console

1 Change to the folder in which you installed the demo applications:

```
cd ~/veos-demos/fluidsDemos
```

2 Call the `fluidsConsole` application with the following command:

```
./src/fluidsConsole
```

The fluidsConsole application produces output similar to the following.

```
[redacted]:~/veos-demos/fluidsDemos$ ./src/fluidsConsole
NOTE: The CUDA Samples are not meant for performance measurements. Results may v
ary when GPU Boost is enabled.

Initialize CudaSimulatorBackend...
Initialize CUDA...
GPU Device 0: "Kepler" with compute capability 3.0

Use CUDA device [Quadro K600] with 1 Multi-Processors
--limit=2147483647
Initialize ConsoleSimulatorFrontend...
Start FPS counter...

------------------------------------
Stable Fluids (512 x 512): 62.2919fps

min: 0
max: 8
mean: 0.333333
std: 0.745733

------------------------------------
Stable Fluids (512 x 512): 63.3876fps
```

Result

You ran the fluids simulation in the console by starting the `fluidsConsole` application.

Method 2

To run the fluid simulation with graphical output

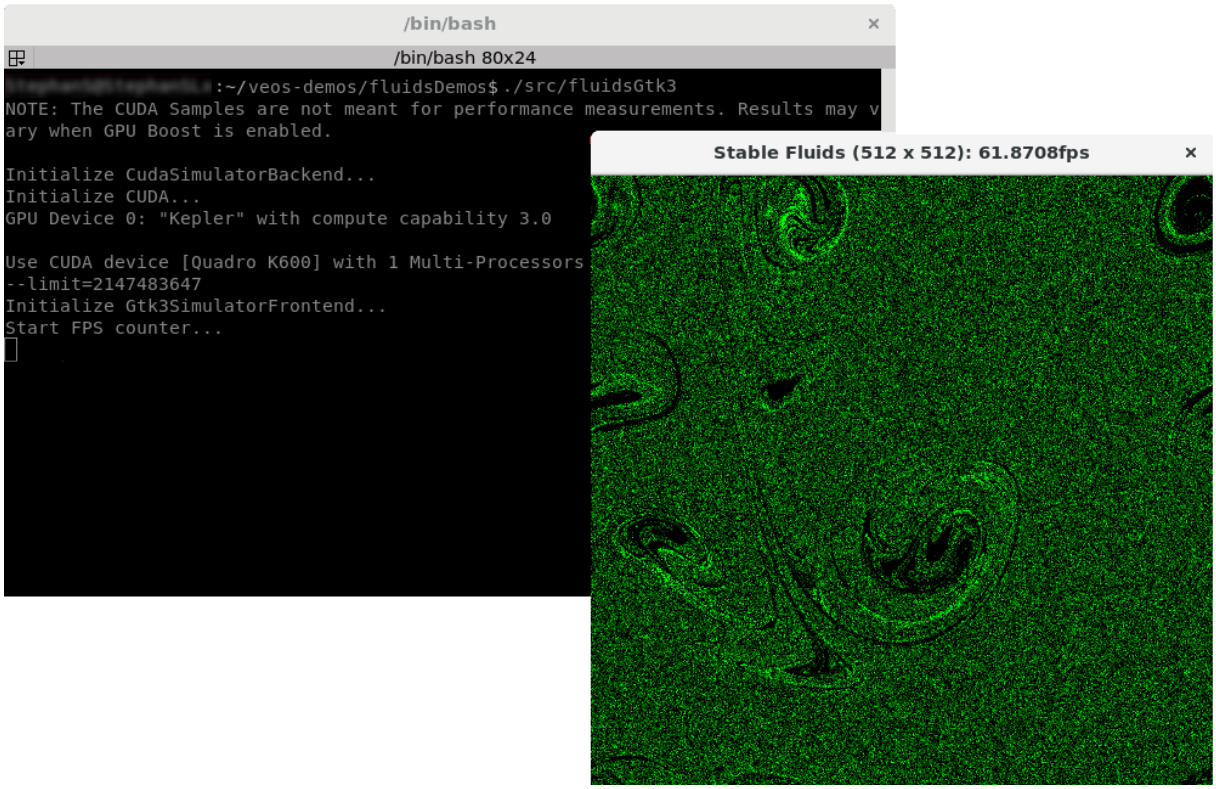
1. Change to the folder in which you installed the demo applications:

```
cd ~/veos-demos/fluidsDemos
```

2. Call the `fluidsGtk3` application with the following command:

```
./src/fluidsGtk3
```

A GTK3 window opens and displays a visualization of the fluid simulation. Refer to the following illustration.



Result

You ran the fluids simulation with graphical output by starting the `fluidsGtk3` application.

Method 3

To run the fluid simulation via a client and a server application

1. Change to the folder in which you installed the demo applications:

   ```
   cd ~/veos-demos/fluidsDemos
   ```

2. Start the server application with the following command:

   ```
   ./src/fluidsServer --port=12345
   ```

   The `--port` option specifies the port through which the server application communicates with the client application.

3. In another terminal, start the client application with the following command:

   ```
   ~/veos-demos/fluidsDemos/src/fluidsClientGtk3 --ip=127.0.0.1 --port=12345
   ```

   The `--ip` option specifies the IP address of the computer that runs the server application, in this case, the host PC.

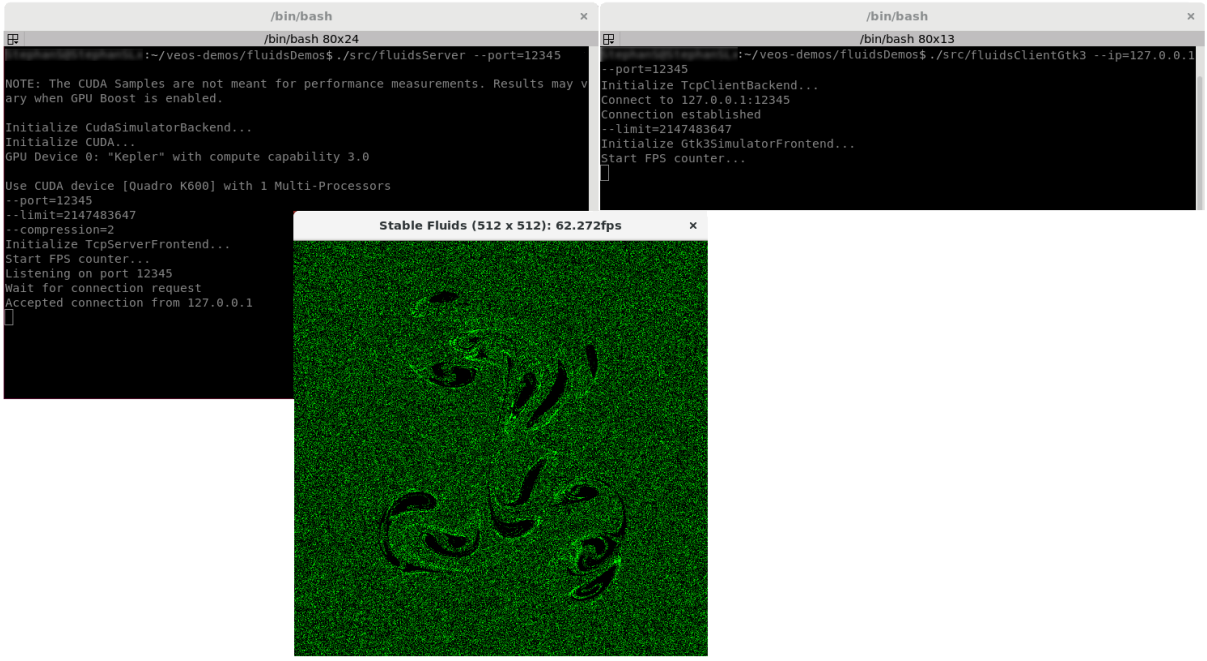
   The `--port` option specifies the port of the server application that you defined in the previous step.

Result

You ran the fluid simulation via a client and a server application which communicate over TCP through a port you specified.

The `fluidsServer` application runs the simulation and sends the resulting data to the `fluidsClientGtk3` application for visualization.

Refer to the following illustration.



Related topics

Basics

Overview of the GPU Passthrough Demo 272

HowTos

How to Prepare the Host PC for Running the Fluid Simulation 276
How to Run the Fluid Simulation with VEOS 285
How to Start a Virtual Machine with GPU Passthrough 283

How to Enable GPU Passthrough

Objective

To enable GPU passthrough so that a virtual machine can access the GPU of the host PC.

Precondition

You must have made the preparations described in How to Prepare the Host PC for Running the Fluid Simulation on page 276.

Method

To enable GPU passthrough

1. Enabling a virtual machine to use the GPU of the host requires activation of the input-output memory management unit (IOMMU). The IOMMU maps virtual device addresses to physical addresses.

   To activate the IOMMU, edit the `GRUB_CMDLINE_LINUX_DEFAULT` line in the `/etc/default/grub` bootloader configuration file as follows:

   ```
   GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"
   ```

2. Update the GRUB bootloader configuration and restart the host PC with the following commands:

   ```
   sudo update-grub
   sudo reboot
   ```

3. Verify that the IOMMU is active by executing the following command:

   `dmesg | grep -i iommu`

   This produces output similar to the following:

   ```
   [    0.000000] DMAR: IOMMU enabled
   [    0.004000] DMAR-IR: IOAPIC id 2 under DRHD base  0xfed91000 IOMMU 1
   [    0.692666] iommu: Adding device 0000:00:00.0 to group 0
   [    0.692677] iommu: Adding device 0000:00:01.0 to group 1
   [    0.692684] iommu: Adding device 0000:00:02.0 to group 2
   ```

4. Before you can make the GPU available to a virtual machine, you have to prevent the host PC from using the GPU. To do this, add corresponding `blacklist` commands to the `/etc/modprobe.d/blacklist-nvidia.conf` file using the following commands:

   ```
   echo "blacklist nvidia" | sudo tee -a /etc/modprobe.d/blacklist-nvidia.conf
   echo "blacklist nvidia_drm" | sudo tee -a /etc/modprobe.d/blacklist-nvidia.conf
   echo "blacklist nvidia_uvm" | sudo tee -a /etc/modprobe.d/blacklist-nvidia.conf
   echo "blacklist nvidiafb" | sudo tee -a /etc/modprobe.d/blacklist-nvidia.conf
   echo "blacklist nouveau" | sudo tee -a /etc/modprobe.d/blacklist-nvidia.conf
   echo "options nouveau modeset=0" | sudo tee -a /etc/modprobe.d/blacklist-\
   nvidia.conf
   ```

   Note

   After this step, the GPU can no longer be used by the host PC. Therefore, if you want to run the fluid simulation on the host PC again, you first have to reverse the changes you made in this step.

5. Enable automatic loading of the vfio-pci driver during startup and bind the GPU to the vfio-pci driver with the following command:

   ```
   echo "vfio-pci" | sudo tee -a /etc/modules-load.d/vfio-pci.conf
   echo "options vfio-pci ids=${PCI_DEVICE_CODE}" | sudo tee -a /etc/modprobe.d/\
   vfio.conf
   ```

6. Update the initramfs image and restart the computer with the following commands:

   ```
   sudo update-initramfs -u
   ```

```
sudo reboot
```

7 After restart, use the following command to verify that the kernel has loaded the vfio-pci driver:

```
dmesg | grep -i vfio
```

The output is similar to this:

```
[    2.578414] VFIO - User Level meta-driver version: 0.3
[    2.606123] vfio_pci: add [10de:0ffa[ffffffff:ffffffff]] class 0x000000/00000000
```

Result

You enabled GPU passthrough on the host PC.

Related topics

Basics

Overview of the GPU Passthrough Demo 272

HowTos

How to Prepare the Host PC for Running the Fluid Simulation 276
How to Run the Fluid Simulation with VEOS 285
How to Start a Virtual Machine with GPU Passthrough 283

How to Start a Virtual Machine with GPU Passthrough

Objective

To start a virtual machine with GPU passthrough using the DSKVM hypervisor.

To keep the virtual machine small, its disk image includes only the GPU driver, not the fluid simulation.

Preconditions

You must have made the preparations described in How to Prepare the Host PC for Running the Fluid Simulation on page 276 and enabled GPU passthrough as explained in How to Enable GPU Passthrough on page 281.

Method

To start a virtual machine with GPU passthrough

1 Specify the location of the DSKVM hypervisor in an environment variable, i.e.,

```
export DSKVM=/opt/dspace/common/dskvm/5.2/bin/dskvm-x86_64
```

2 Use DSKVM to start a virtual machine that includes GPU passthrough:

```
${DSKVM} -enable-kvm -cpu host,kvm=off -m 4096 -nodefaults \
-L /usr/share/seabios -L /usr/lib/ipxe/qemu \
-drive file=~/veos-demos/fluidsDemos/img/gpu-driver.raw,if=virtio,format=raw \
-serial stdio -nographic \
-device virtio-vga -device vfio-pci,host=${PCI_HOST_BUS_ADDRESS} \
-kernel /opt/dspace/veosautosaradaptiveplatform5.2/Images/\
AutosarAdaptivePlatform.bzImage \
```

```
-append "root=/dev/vda console=ttyS0 rw"
```

The options have the following effects:

- `-enable-kvm` enables full virtualization support.
- `-cpu host,kvm=off` instructs DSKVM to emulate the CPU of the host PC.
- `-m 4096` allocates 4,096 MB of RAM to the virtual machine
- `-nodefaults` instructs DSKVM not to emulate default devices, e.g., serial ports, parallel ports, etc.
- `-L /usr/share/seabios` specifies the location of SeaBIOS, which is the BIOS of the virtual machine.

  `-L /usr/lib/ipxe/qemu` specifies the location of the iPXE ROM images, which enable starting the virtual machine from the network.
- `-drive file=/tmp/fluidsDemos/img/gpu-driver.raw, if=virtio,format=raw` specifies the image, driver, and image format to be used for the virtual hard disk.
- `-serial stdio -nographic` redirects the virtual serial port to standard I/O and completely disables graphical output.
- `-device virtio-vga -device vfio-pci, host=${PCI_HOST_BUS_ADDRESS}` adds the device drivers for the virtual I/O display device and for the virtual function I/O PCI device, to which the GPU is bound.
- `-kernel /opt/dspace/veosautosaradaptiveplatform5.2/Images/AutosarAdaptivePlatform.bzImage` specifies the kernel image file.
- `-append "root=/dev/vda console=ttyS0 rw"` assigns the root file system to the first virtualized disk, i.e., `/dev/vda`, specifies the first serial port, i.e., `ttyS0`, to be used for the console, and assigns read and write access to it.

Result

You used the DSKVM hypervisor to start a virtual machine with GPU passthrough.

The virtual machine provides information on the GPU driver in the console. Refer to the following illustration.

Note

The virtual machine console might display `Kernel panic` messages. This is not critical. It is the result of the `init` process terminating.

```
[    1.297667] nvidia: module license 'NVIDIA' taints kernel.
[    1.298360] Disabling lock debugging due to kernel taint
[    1.311316] nvidia-nvlink: Nvlink Core is being initialized, major device number 247
[    1.321903] PCI Interrupt Link [LNKC] enabled at IRQ 11
[    1.322826] nvidia 0000:00:03.0: vgaarb: changed VGA decodes: olddecodes=io+mem,decodes=none:owns=io+mem
[    1.408448] tsc: Refined TSC clocksource calibration: 3192.030 MHz
[    1.409354] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x2e02e10e5ee, max_idle_ns: 440795339652 ns
[    1.410643] clocksource: Switched to clocksource tsc
[    3.924521] NVRM: loading NVIDIA UNIX x86_64 Kernel Module  450.80.02  Wed Sep 23 01:13:39 UTC 2020
[    3.925941] modprobe (1015) used greatest stack depth: 13288 bytes left
Fri Mar  5 13:42:10 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.80.02    Driver Version: 450.80.02    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Quadro K600         Off  | 00000000:00:03.0 Off |                  N/A |
| 46%   48C    P0    N/A /  N/A |      0MiB /   982MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Related topics

Basics

Overview of the GPU Passthrough Demo..... 272

HowTos

How to Enable GPU Passthrough..... 281
How to Prepare the Host PC for Running the Fluid Simulation..... 276
How to Run the Fluid Simulation with VEOS..... 285

How to Run the Fluid Simulation with VEOS

Objective

To run the fluid simulation with VEOS.

Preconditions

- You must have made the preparations described in How to Prepare the Host PC for Running the Fluid Simulation on page 276 and enabled GPU passthrough as explained in How to Enable GPU Passthrough on page 281.
- Additionally, it is useful to define the VEOS installation directory in an environment variable. You can do this via the following command:

```
export VEOS_DIR=/opt/dspace/veos5.2/bin
```

Possible methods

You can run the fluid simulation in VEOS in two different ways:

- Using the `fluidsConsole` adaptive V-ECU. This produces only text output. Refer to Method 1 on page 286.

- Using the `fluidsServer` adaptive V-ECU, which communicates with the `fluidsClientGtk3` application on the host PC via TCP. The `fluidsClientGtk3` application provides graphical output.

  Refer to Method 2 on page 287.

Method 1

To run the fluid simulation with VEOS without graphical output

1. Copy the `fluidsConsole.vecu` adaptive V-ECU implementation container from the `~/veos-demos/fluidsDemos/vecu` folder on the Linux host PC to a temporary location on the Windows host PC, e.g., `C:\Temp\`.
2. On the Windows host PC, open a console, change to the VEOS installation folder, and execute the following command to build the `fluidsConsole.vecu` adaptive V-ECU into the `fluidsConsole.osa` offline simulation application:

   ```
   Bin\veos.exe build adaptive-vecu C:\Temp\fluidsConsole.vecu -o C:\Temp\
   fluidsConsole.osa
   ```

   Note

   Make sure to copy the command to a single line in the console.

3. Copy the `fluidsConsole.osa` offline simulation application to the `~/veos-demos` directory on the Linux host PC.
4. On the Linux host PC, specify the GPU for the DSKVM hypervisor via the `DSKVM_ADD_ARGS` environment variable and start VEOS:

   ```
   export DSKVM_ADD_ARGS="-device vfio-pci,host=${PCI_HOST_BUS_ADDRESS}"
   ${VEOS_DIR}/veoskernel &
   ```

5. Run the fluids simulation in VEOS by loading and starting the `fluidsConsole.osa` offline simulation application:

   ```
   ${VEOS_DIR}/veos-sim load -s ~/veos-demos/fluidsConsole.osa
   ```

6. You can view standard output mesages of the adaptive V-ECU during in the console during run time.

   To do this, use the following command:

   ```
   tail -n 100 -f $(ls -t /tmp/veos-*/ftp/logs/*.out | head -1)
   ```

7. To stop the simulation and shut down VEOS, execute the following commands:

   ```
   ${VEOS_DIR}/veos-sim unload
   ${VEOS_DIR}/veos-sim down
   ```

Result

You ran the fluid simulation with VEOS without graphical output and displayed standard output messages of the `fluidsConsole` adaptive V-ECU in the terminal.

Method 2

To run the fluid simulation with VEOS with graphical output

1. Copy the `fluidsServer.vecu` adaptive V-ECU implementation container from the `~/veos-demos/fluidsDemos/vecu` folder on the Linux host PC to a temporary location on the Windows host PC, e.g., `C:\Temp\`.

2. On the Windows host PC, open a console, change to the VEOS installation folder, and execute the following command to build the `fluidsServer.vecu` adaptive V-ECU implementation and integrate it into the `fluidsServer.osa` offline simulation application:

```
Bin\veos.exe build adaptive-vecu C:\Temp\fluidsServer.vecu -o
C:\Temp\fluidsServer.osa --hypervisor-connection-state HostOnly
```

   The `--hypervisor-connection-state HostOnly` option enables the adaptive V-ECU to communicate with applications on the VEOS host PC.

   Note

   Make sure to copy the command to a single line in the console.

3. Copy the `fluidsServer.osa` offline simulation application to the `~/veos-demos` directory on the Linux host PC.

4. On the Linux host PC, specify the GPU for the DSKVM hypervisor via the `DSKVM_ADD_ARGS` environment variable and start VEOS:

```
export DSKVM_ADD_ARGS="-device vfio-pci,host=${PCI_HOST_BUS_ADDRESS}"
${VEOS_DIR}/veoskernel &
```

5. VEOS requires a virtual network adapter to establish a network connection between the the host PC and the virtual machine that runs the adaptive V-ECU. On Linux, you have to create a TAP device for this.

   Use the following commands to create a TAP device, assign an IP address to it, and activate it:

```
sudo ip tuntap add ds_veos_tap mode tap
sudo ip addr add 192.168.174.250/24 dev ds_veos_tap
sudo ip link set dev ds_veos_tap up
```

6. Run the fluids simulation in VEOS by loading and starting the `fluidsServer.osa` offline simulation application:

```
${VEOS_DIR}/veos-sim load -s ~/veos-demos/fluidsServer.osa
```

7. Confirm that a connection to the `fluidsServer` adaptive V-ECU can be established by sending a `ping` request to it:

```
ping 192.168.174.10
```

   This produces the following output:

```
[redacted]:~$ ping 192.168.174.10
PING 192.168.174.10 (192.168.174.10) 56(84) bytes of data.
64 bytes from 192.168.174.10: icmp_seq=1 ttl=64 time=2.91 ms
64 bytes from 192.168.174.10: icmp_seq=2 ttl=64 time=12.0 ms
64 bytes from 192.168.174.10: icmp_seq=3 ttl=64 time=1.36 ms
64 bytes from 192.168.174.10: icmp_seq=4 ttl=64 time=9.57 ms
64 bytes from 192.168.174.10: icmp_seq=5 ttl=64 time=18.1 ms
^C
--- 192.168.174.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 1.361/8.807/18.185/6.148 ms
```

8 Observe the standard output mesages of the `fluidsServer` adaptive V-ECU via the `tail -n 100 -f $(ls -t /tmp/veos-*/ftp/logs/*.out | head -1)` command.

When you see the `Wait for connection request` message, you can start the `fluidsClientGtk3` application to visualize the simulation:

```
~/veos-demos/fluidsDemos/src/fluidsClientGtk3 --ip=192.168.174.10 --port=12345
```

9 To stop the simulation and close VEOS, execute the following commands:

```
${VEOS_DIR}/veos-sim unload
${VEOS_DIR}/veos-sim down
```

Result

You ran the fluid simulation with VEOS with graphical output on the host PC.

Note

GPU passthrough can be performed for only one adaptive V-ECU at a time. If a simulation involves more than one adaptive V-ECU, you have to specify which of them can access the GPU of the host PC. To do this, use the `DSKVM_ADD_ARGS_<V-ECU NAME>` environment variable instead of `DSKVM_ADD_ARGS` and replace `<V-ECU NAME>` with the name of the adaptive V-ECU.

Related topics

Basics

Overview of the GPU Passthrough Demo 272

HowTos

How to Enable GPU Passthrough 281
How to Prepare the Host PC for Running the Fluid Simulation 276
How to Run the Fluid Simulation on the Host PC 278
How to Start a Virtual Machine with GPU Passthrough 283

References

VEOS Build Console Command Reference 318
VEOS Simulator Console Command Reference 320

Working with VEOS on Linux

Where to go from here

Information in this section

Getting Started with VEOS on Linux.....290

Loading and Running an Offline Simulation Application, and Debugging Source Code.....301

Running VEOS 5.2 in a Linux Docker Container.....308

Getting Started with VEOS on Linux

Where to go from here

Information in this section

Introduction to VEOS on Linux.....290
VEOS lets you build offline simulation applications that you can run with VEOS on Linux.

How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS.....292
To install VEOS 5.2 on Ubuntu Linux 18.04 LTS.

Hypervisor Configuration for Adaptive V-ECU Simulation.....296
On Linux, VEOS uses the dSPACE-specific DSKVM hypervisor for adaptive V-ECU simulation.

How to Build an Offline Simulation Application for Linux with the VEOS Player.....298
To use the VEOS Player for building an offline simulation application (OSA) that you can run with VEOS on Linux.

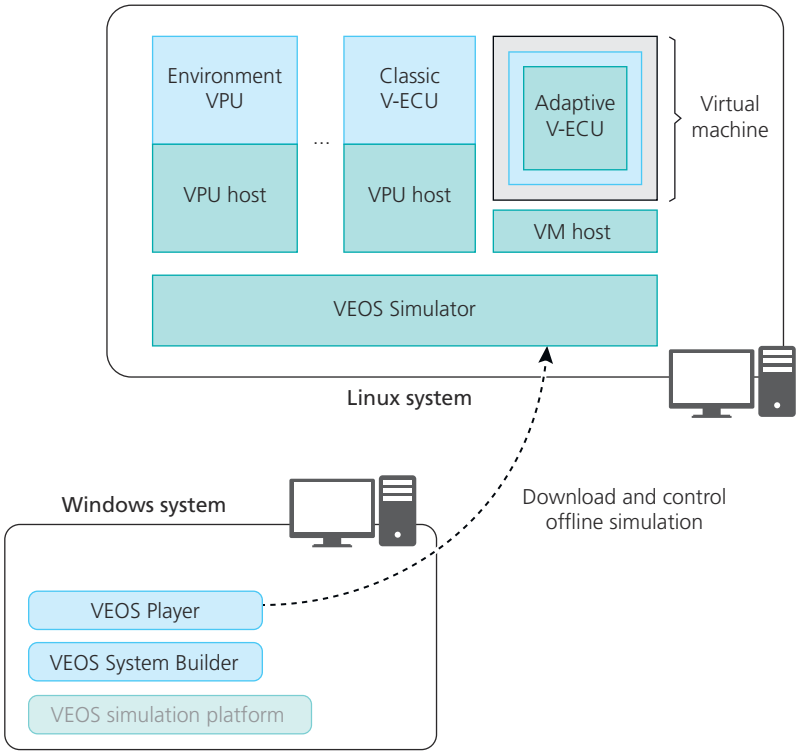
Introduction to VEOS on Linux

Overview

VEOS lets you build offline simulation applications that you can run with VEOS on Linux.

Simulation control

You can control an offline simulation on a Linux system either locally via command line interface or remotely via VEOS Player or command line interface on a Windows system. Refer to the following illustration.



You can also use AutomationDesk, ModelDesk, or ControlDesk for remote-controlling an offline simulation that runs on a Linux system.

Compatibility

Model support VEOS on Linux supports the same container and archive file formats that are supported by the Windows version of VEOS. Refer to Compatibility of VEOS 5.2 on page 40.

Simulation of bus communication VEOS on Linux supports the same features of bus communication simulation as the Windows version. Refer to Bus Communication Features for SIL Testing on page 196.

Linux distribution dSPACE recommends using Ubuntu Linux 18.04 LTS 64 bit. If you want to use a different Linux distribution, contact dSPACE Support.

Limitations

The following limitations apply:

- There is no graphical user interface, i.e., VEOS Player, for Linux.
- You must use VEOS on Windows to build an offline simulation application. Making changes to an OSA file that are not related to importing container files requires the VEOS Player.
- Virtual bypassing is not supported.

Related topics

Basics

Basics on the VEOS Command Line Interface..... 315
Compatibility of VEOS 5.2..... 40
Hypervisor Configuration for Adaptive V-ECU Simulation..... 296

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player..... 298
How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS..... 292
How to Load an Offline Simulation Application..... 301

How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS

Objective

To install VEOS 5.2 on Ubuntu Linux 18.04 LTS.

Preconditions

To install dSPACE VEOS 5.2 on Ubuntu Linux 18.04 LTS, you must meet the following preconditions:

- You must have Ubuntu Linux 18.04 LTS (Bionic Beaver).
- Microsoft .NET Core 3.1 must be installed on the Linux system. If this is not the case, you have two options:
  - If you have Internet access and clearance from your local system administrator, you can simply execute the `configure-ms-package-server.sh` script to install Microsoft .NET Core 3.1.
  - Otherwise, you must have access, for example, to a local package mirror that provides the Microsoft .NET Core 3.1 packages. You can then follow the instructions on the official Microsoft .NET Core installation website https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu.
- You must have administrative privileges on the Linux system, i.e., you must be able to run commands as root using the `sudo` command.
- The simulation of adaptive V-ECUs requires a processor that supports hardware virtualization. In addition, hardware support for virtualization must be enabled in the computer's BIOS or UEFI.

Possible methods

There are two methods for installing VEOS 5.2 on Ubuntu Linux 18.04 LTS:

- For a manual installation, refer to Method 1 on page 293.
- For using the installation script, refer to Method 2 on page 294.

Method 1

To install VEOS 5.2 on Ubuntu Linux 18.04 LTS manually

1. On the Linux system, create a temporary folder in your home directory and extract the delivered archive to this folder.
2. In a shell, change to the folder you created in the previous step and execute the following commands:
   - `chmod +x *.sh`

     to specify execute permission for the shell scripts for all users.
   - `chmod +r packages.tar`

     to specify read permission for the `packages.tar` archive for all users.
3. In the shell, use the following command to create a temporary directory `tmp_dir` and extract the `packages.tar` archive to it:

```
mkdir tmp_dir && tar -xf packages.tar -C tmp_dir --exclude 'dspace-portmapper*'
```

Note

- The VEOS Simulator requires an active port mapper. By default, it uses the RPC bind portmap service (rpcbind). If you cannot use rpcbind, leave the `--exclude 'dspace-portmapper*'` option out of the command above to enable the installation of the dSPACE port mapper.
- If you do not need adaptive platform support, you can exclude it from the installation by adding the following options to the extraction command:

```
--exclude 'dspace-veos-adaptive-platform*' --exclude 'dspace-dskvm*' \
--exclude 'dspace-veos-*all*'
```

4. Make sure that all the files in the `tmp_dir` directory can be accessed by the apt-get installer by executing

```
chmod -R a+w tmp_dir
```

You can now install the extracted DEB packages:

```
sudo apt-get install ./tmp_dir/*.deb
```

Tip

You can perform the installation in non-interactive mode. To do this, you first have to specify that you accept the EULA using the following command:

```
echo dspace-eula-2021a dspace/all boolean true | debconf-set-selections
```

You can then use the following modified installation command:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install -y \
--no-install-recommends ./tmp_dir/*.deb
```

Once the installation is complete, you can remove the temporary directory and its contents by executing the following command:

```
rm -rf tmp_dir
```

Method 2

To install VEOS 5.2 on Ubuntu Linux 18.04 LTS using the installation script

1. On the Linux system, create a temporary folder in your home directory and extract the delivered archive to this folder.
2. In a shell, change to the folder you created in the previous step and execute the following commands:
   - `chmod +x *.sh`

     to specify execute permission for the shell scripts for all users.
   - `chmod +r packages.tar`

     to specify read permission for the `packages.tar` archive for all users.
3. Modify the `install-dspace-veos-5.2.sh` installation script according to your requirements:
   - The VEOS Simulator requires an active port mapper. By default, it uses the RPC bind portmap service (rpcbind). If you cannot use rpcbind, install the dSPACE portmapper by setting `INSTALL_DSPACE_PORTMAPPER=1`.
   - If you do not need adaptive platform support, set `INSTALL_VEOS_ADAPTIVE_PLATFORM=0`.
   - For non-interactive installation, set `INSTALL_NON_INTERACTIVE=1`.
   - To activate remote access to the VEOS Simulator, set `ENABLE_VEOS_REMOTE_ACCESS=1`.

     You can also activate and deactivate remote access manually later on before you start the VEOS Simulator.
   - To enable floating network license servers, remove the `#` character in the line `enable_license_server 127.0.0.1` and replace `127.0.0.1` with the IP or FQDN of the license server. Add corresponding lines for each additional license server you want to use.
4. Execute the `install-dspace-veos-5.2.sh` script.

   Note

   Do not execute the script as superuser because this might block other users from starting the VEOS Simulator. If required, the installation script asks for superuser privileges.

Result

You installed VEOS 5.2 on Ubuntu Linux 18.04 LTS.

The installation is located under `/opt/dspace/`.

The following table provides an overview of the relevant folder content.

Subfolder	Content
`/common/dskvm/5.2`	`/bin`: Contains the DSKVM hypervisor, which is required for the simulation of adaptive V–ECUs.
	`/util`: Contains shell scripts for configuring permissions required by DSKVM.[1]
`/share`	`/<GUID>/log`: Folder for saving simulation log files.
	`/veos/config`: Contains the `5.2.config` XML file for configuring the VEOS installation.
`/veos5.2`	`/bin`: Contains all the VEOS executables and related library files.
	`/tools/shell`: Contains shell scripts for activating the VEOS environment.

[1] Refer to Hypervisor Configuration for Adaptive V-ECU Simulation on page 296.

Next steps

Before you start working with VEOS, perform the following checks:

- Verify that the CodeMeter server process is running by executing the following command:

  ```
  systemctl status codemeter.service
  ```

  If the process is running, you will see the following output:

  ```
  ● codemeter.service - CodeMeter RunTime Server
     Loaded: loaded (/lib/systemd/system/codemeter.service; enabled; vendor preset
     Active: active (running) since Thu 2020-06-04 07:23:32 PDT; 38min ago
   Main PID: 740 (CodeMeterLin)
      Tasks: 11 (limit: 2266)
     CGroup: /system.slice/codemeter.service
             └─740 /usr/sbin/CodeMeterLin -f

  Jun 04 07:23:32 ubuntu systemd[1]: Started CodeMeter RunTime Server.
  ```

  If the process is not running, it will be displayed as inactive in the output in white letters. In this case, start the process as superuser with the following command:

  ```
  sudo systemctl start codemeter.service
  ```

  By default, CodeMeter Runtime uses a hardware token. This token must be connected if you use this method to get licenses.

  If you want to use floating network licenses, you can use the following command in the shell to add a license server to the server list:

  ```
  cmu --add-server <server IP or FQDN>
  ```

- Ensure that a port mapper service is active:
  - If you use the rpcbind port mapper service, execute the following command:

    ```
    systemctl status rpcbind.service
    ```

If the service is running, you will see the following output:

```
● rpcbind.service - RPC bind portmap service
   Loaded: loaded (/lib/systemd/system/rpcbind.service; enabled; vendor preset:
   Active: active (running) since Thu 2020-06-04 07:23:31 PDT; 1min 20s ago
     Docs: man:rpcbind(8)
 Main PID: 568 (rpcbind)
    Tasks: 1 (limit: 2266)
   CGroup: /system.slice/rpcbind.service
           └─568 /sbin/rpcbind -f -w

Jun 04 07:23:31 ubuntu systemd[1]: Starting RPC bind portmap service...
Jun 04 07:23:31 ubuntu systemd[1]: Started RPC bind portmap service.
```

If the service is not running, it will be displayed as inactive in the output in white letters. In this case, activate the service as superuser with the following command:

```
sudo systemctl start rpcbind.service
```

- If you use the dSPACE port mapper, activate it via the following command:

```
/opt/dspace/common/portmapper/bin/portmapper -d
```

- If you want to simulate adaptive V-ECUs as a non-root user, you have to change several privileges. The VEOS installation provides shell scripts for this purpose. Refer to Hypervisor Configuration for Adaptive V-ECU Simulation on page 296.

Related topics

Basics

Basics on the VEOS Command Line Interface..... 315

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player..... 298
How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux..... 306
How to Load an Offline Simulation Application..... 301

Hypervisor Configuration for Adaptive V-ECU Simulation

Introduction

On Linux, VEOS uses the dSPACE-specific DSKVM hypervisor for adaptive V-ECU simulation.

To simulate adaptive V-ECUs as a non-root user, you have to change several privileges. The VEOS installation provides shell scripts for this. Refer to Configuring DSKVM on page 297.

If you require a network connection to an adaptive V-ECU from outside the simulation, you have to set up a virtual network adapter for the hypervisor. Refer to Setting up virtual network adapters on page 297.

Configuring DSKVM

By default, non-root users do not have the required privileges to access DSKVM for adaptive V-ECU simulation. You can change this by means of several configuration scripts, which are located under `/opt/dspace/common/dskvm/5.2/util`.

For each script, there is a corresponding `unconfigure<xyz>.sh` script which lets you reverse the configuration.

Tip

Instead of calling each script separately, you can run the `configure.sh` script, which calls all other scripts.
Similarly, `unconfigure.sh` calls all the other `unconfigure<xyz>.sh` scripts.

The following scripts execute mandatory tasks for enabling a non-root user to simulate adaptive V-ECUs:

- `configureKvm.sh <user>`: Gives the specified user read and write access to DSKVM by creating a new `kvm` system group for the `/dev/kvm` device and adding the user to the group.
- `configureLoop.sh <user>`: Gives the specified user read and write access to loop devices. Loop devices are used for mounting disk images.
- `configureExt4.sh`: Enables user namespace mounts for the ext4 file system.

Enabling GPU passthrough If you want to simulate adaptive V-ECUs with GPU passthrough, you have to remove the limit on the amount of RAM the user can lock. The VEOS installation provides the `configureLimits.sh` script for this. You have to specify the user when you call the script, i.e., `configureLimits.sh <user>`.

Note

You have to log out and log in again for the changes made by the configuration scripts to take effect.

Setting up virtual network adapters

If you want to establish a network connection to an adaptive V-ECU from outside the simulation, you have to set up a virtual network adapter, or TAP device, for the hypervisor. The syntax for this is as follows:

```
sudo ip tuntap add <name> mode tap
sudo ip addr add <IP> dev <name>
sudo ip link set dev <name> up
```

The first line creates the TAP device, where `<name>` specifies its name.

The second line assigns an IP address to the TAP device. The IP address must be in the same subnet as that of the adaptive V-ECU.

The third line activates the TAP device.

Related topics

Basics

Basics on Simulating Adaptive V-ECUs 236
Introduction to the GPU Passthrough Demo 271

How to Build an Offline Simulation Application for Linux with the VEOS Player

Objective

To use the VEOS Player for building an offline simulation application (OSA) that you can run with VEOS on Linux.

Preconditions

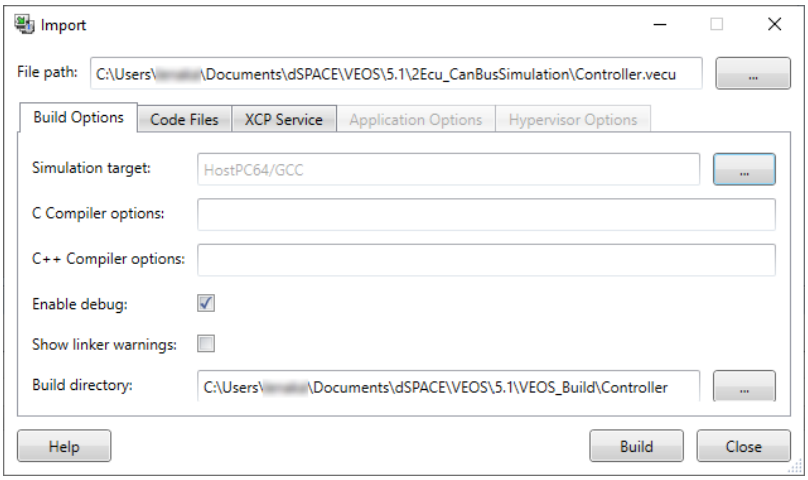
You require VEOS Player 5.2 on a Windows system.

When you import SIC and BSC files, make sure that they were generated for Linux. Refer to Compatibility of VEOS 5.2 on page 40.

Method

To build an offline simulation application for Linux with the VEOS Player

1. In the VEOS Player, click File – New.
   VEOS creates an empty OSA.
2. On the Home ribbon, click Import.
   VEOS opens the Select File for Import dialog.
3. Select the file you want to integrate into the simulation system and click Open.
   VEOS opens the Import dialog.



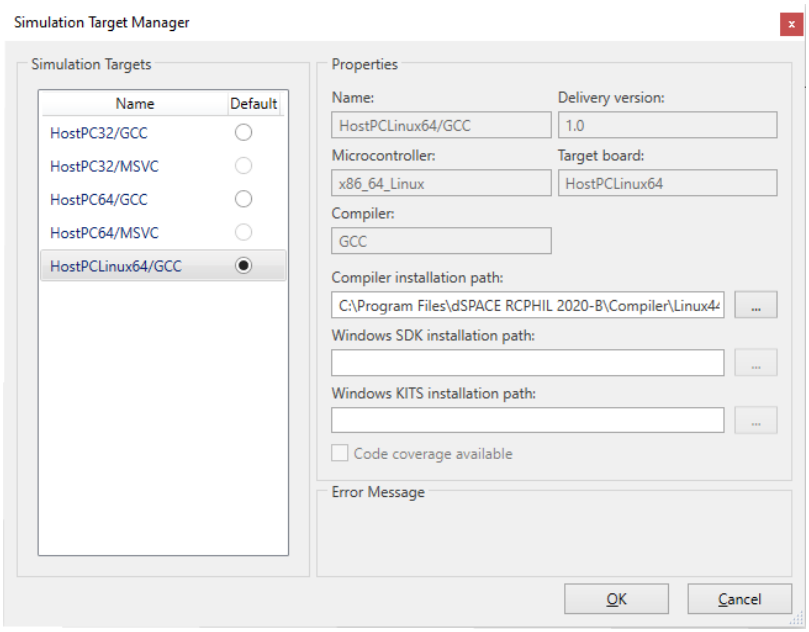
Tip

For classic VPUs, you can select the **Enable debug** checkbox on the **Build Options** page of the **Import** dialog if you want to debug the source code later during offline simulation. However, the checkbox has no effect for adaptive V-ECUs.

4 On the **Build Options** page, click the Browse button beside the **Simulation target** field.

VEOS opens the **Simulation Target Manager**.

5 In the **Simulation Target Manager**, select **HostPCLinux64/GCC** and click **OK**.



6 In the **Import** dialog, click **Build**.

VEOS starts the build process and displays progress messages in the **Build log** dialog.

When the build process is finished, click **Close**.

7 Add more files as required and, if applicable, connect the VPUs of the simulation system.

8 Save the OSA file.

Result

You used the VEOS Player to build an offline simulation application that you can run on Linux.

Tip

You can also use the VEOS command line interface on Windows to build an offline simulation application for Linux. However, there are some restrictions. Refer to VEOS Build Console Command Reference on page 318.

Next steps

You can load the OSA to a remote simulator on a Linux machine. Refer to How to Load an Offline Simulation Application on page 301.

Related topics

Basics

Basics on the VEOS Command Line Interface..... 315

HowTos

How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux..... 306
How to Load an Offline Simulation Application..... 301

References

Simulation Target Manager..... 163

Loading and Running an Offline Simulation Application, and Debugging Source Code

Where to go from here

Information in this section

How to Load an Offline Simulation Application.....301
You can load an offline simulation application via the VEOS command line interface or via VEOS Player on a Windows PC. You can also use AutomationDesk, ModelDesk, or ControlDesk on a Windows PC.

How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux.....306
To debug the source code of classic VPUs in an offline simulation on Linux using the GNU debugger (GDB).

How to Load an Offline Simulation Application

Objective

To load an offline simulation application to the VEOS Simulator on Linux.

Preconditions

You must meet the following preconditions:

- Either the rpcbind or the dSPACE port mapper service must be active. Refer to step 7 of How to Install VEOS 5.2 on Ubuntu Linux 18.04 LTS on page 292.
- The VEOS Simulator on the Linux system must be started via the `/opt/dspace/veos5.2/bin/veos sim up` command.
- For remote control, you must enable remote access via the `/opt/dspace/veos5.2/bin/veoskernel config --enable-remote-access` command *before* you start the VEOS Simulator.

  You can use the `--disable-remote-access` option to deactivate remote access after you stopped the VEOS Simulator.

Possible methods

There are three possible methods to load an offline simulation application to the VEOS Simulator on Linux:

- Locally or remotely via command line interface on Linux or Windows. Refer to Method 1 on page 302.
- Remotely via the VEOS Player on Windows. Refer to Method 2 on page 302.
- Remotely via AutomationDesk, ModelDesk, or ControlDesk on Windows.

  Refer to Method 3 on page 303 for instructions using ControlDesk as an example.

Method 1

To load an offline simulation application via command line interface

1 Depending on whether you use the VEOS command line interface on the local Linux host or on a remote host, use the following commands to load an offline simulation application to the VEOS Simulator:

- On the local Linux host: Use `./veos sim load <OSA file>` in the `/opt/dspace/veos5.2/bin` folder or simply `veos sim load <OSA file>` in any folder if the VEOS environment is active. You can activate it via `. /opt/dspace/veos5.2/tools/shell/veos-env.sh` or, in Bash, via `source /opt/dspace/veos5.2/tools/shell/veos-env.sh`.
- On a remote Linux host: Add the `-h <IP address>` option to the command above to specify the IP address of the VEOS Simulator host.
- On a remote Windows host, use `veos.exe sim load <OSA file> -h <IP address>` in the `\Bin` subfolder of the VEOS installation folder.

Tip

- If you do not know the IP address of your Linux machine, execute `hostname -I` in a terminal on the Linux machine.
- If you run Linux on a virtual machine, make sure that the network connection of the virtual machine is set to NAT.

Result

You loaded an offline simulation application to the VEOS Simulator using the VEOS command line interface on the local Linux host or on a remote Linux or Windows host.

For examples of controlling an offline simulation application via command line interface, refer to Basics on the VEOS Command Line Interface on page 315.

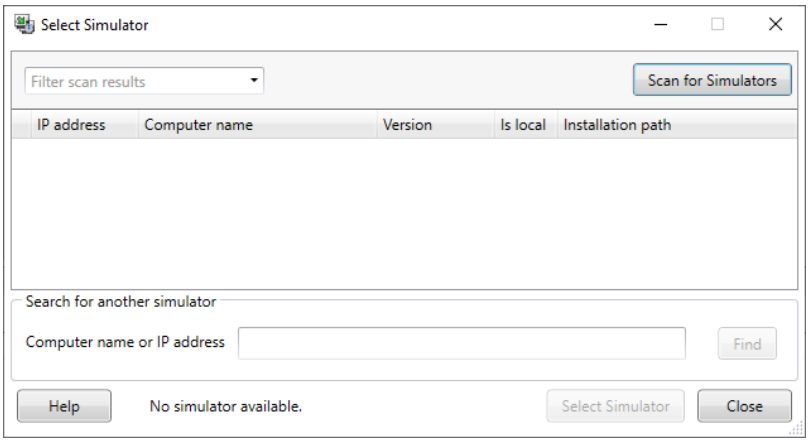
Method 2

To load an offline simulation application via VEOS Player on Windows

1 In VEOS Player on your Windows system, build or open an OSA for VEOS on Linux. Refer to How to Build an Offline Simulation Application for Linux with the VEOS Player on page 298.

2 On the **Simulation** ribbon, click **Select Simulator**.

The **Select Simulator** dialog opens.



3 Enter the IP address of the Linux machine in the **Computer name or IP address** field and click **Find**. As soon as the remote simulator is found, it is added to the list.

Tip

- If you do not know the IP address of your Linux machine, execute `hostname -I` in a terminal on the Linux machine.
- If you run Linux on a virtual machine, make sure that the network connection of the virtual machine is set to NAT.

4 Select the Linux remote simulator from the list and click **Select Simulator** to save the selection and close the dialog.

5 On the **Simulation** ribbon, click the arrow on the **Load** split button and select the remote Linux simulator from the list.

The offline simulation application is loaded to the remote Linux simulator.

Result

You selected the Linux remote simulator and loaded an offline simulation application onto it. You can now control the simulation via VEOS Player on Windows.

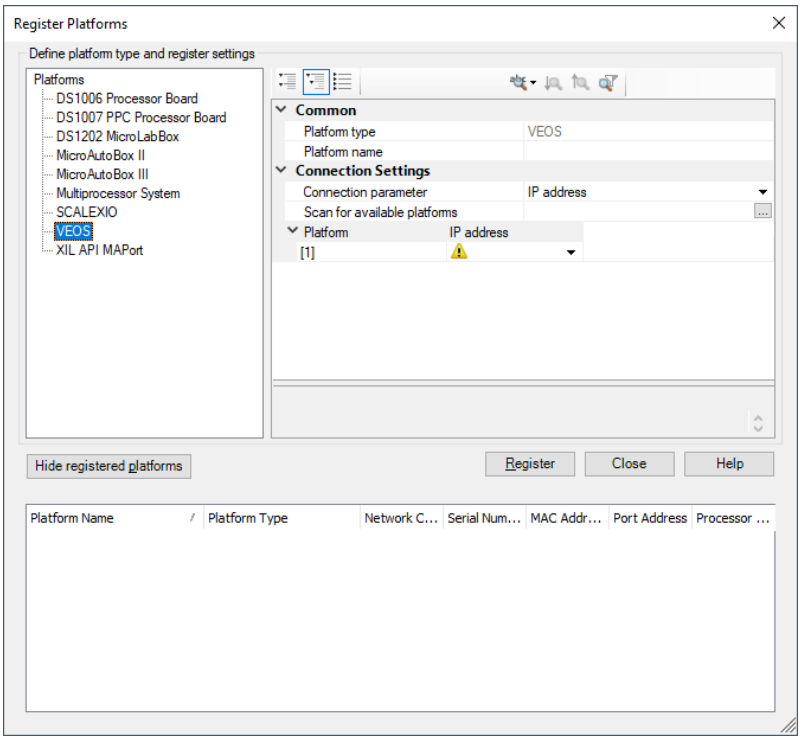
Method 3

To load an offline simulation application via ControlDesk

1 Open or create a project and experiment in ControlDesk.

2 On the **Platforms** ribbon, select **Register Platforms**.

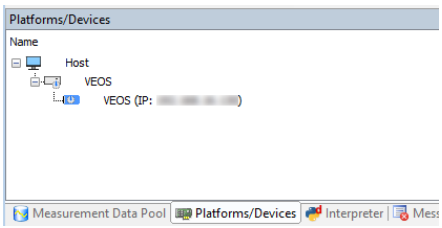
ControlDesk opens the **Register Platforms** dialog.



3 In the dialog, select **VEOS** from the Platforms list and enter the IP address of your Linux machine in the **IP address** field.

4 Click **Register** to complete the registration and **Close** to close the dialog.

ControlDesk displays the VEOS platform in the **Platforms/Devices** controlbar.

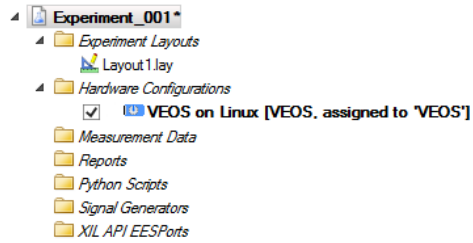


5 Assign the VEOS platform to the active experiment by selecting **Add to active experiment** from the context menu of the VEOS platform.

ControlDesk opens the **Platform Name** dialog. The name you enter must be unique within the experiment.

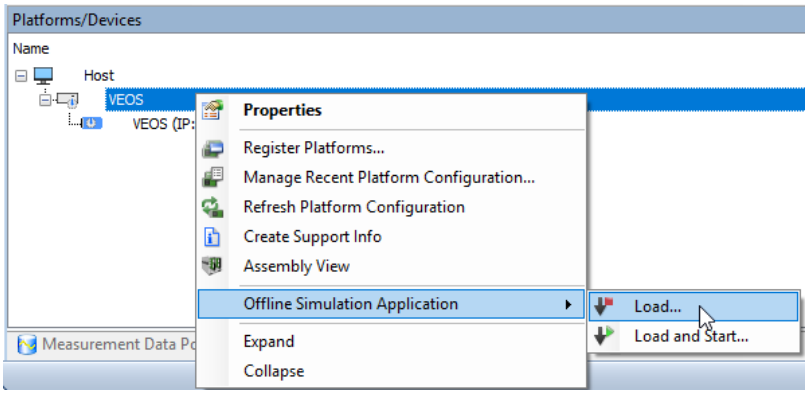
Click **OK** to confirm and close the dialog.

ControlDesk adds the VEOS platform to the active experiment.



6 Use the context menu of the VEOS platform in the **Platforms/Devices** controlbar or under **Hardware Configurations** in the **Project** controlbar to load an offline simulation application to the platform.

Refer to the following exemplary illustration.



Result

You registered the VEOS platform in ControlDesk and assigned it to an experiment. You then loaded an offline simulation application to the Linux remote simulator via the registered platform.

You can use the context menu of the VEOS platform to control the simulation.

Related topics

Basics

Basics on the VEOS Command Line Interface..... 315
Basics on the VEOS Player..... 30
Loading and Running an Offline Simulation Application..... 184

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player..... 298
How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux..... 306

References

VEOS Simulator Console Command Reference..... 320

How to Debug Source Code of Classic VPUs in an Offline Simulation on Linux

Objective

To debug the source code of classic VPUs in an offline simulation on Linux using the GNU debugger (GDB).

For instructions on how to debug adaptive V-ECUs, refer to How to Debug an Adaptive V-ECU on page 245.

Preconditions

You must have selected the **Enable debug** option on the **Build Options** page of the **Import** dialog during import of the VPU to the OSA.

Method

To prepare the debugging of source code of classic VPUs

1. Load the OSA to VEOS. Refer to How to Load an Offline Simulation Application on page 301.
2. Identify the process IDs of the VPUs you want to debug by executing the following command:

   ```
   ps ax | grep VeosVpuHost
   ```

   The output will be similar to the following:

   ```
   3459 pts/0    Sl+    0:01 /opt/dspace/veos5.0/Bin/VeosVpuHost Controller
   3461 pts/0    Sl+    0:01 /opt/dspace/veos5.0/Bin/VeosVpuHost Plant
   3679 pts/2    S+     0:00 grep --color=auto VeosVpuHost
   ```

   The numbers in the first column are the process IDs. In this case, there are two VPUs in the simulation, Controller and Plant, and their process IDs are 3459 and 3461, respectively.
3. Start GDB with the `gdb` command.

   The shell sets up the GDB environment. When it is finished, it displays the `(gdb)` prompt.
4. At the `(gdb)` prompt, type

   ```
   attach <process ID>
   ```

   GDB attaches to the VPU process. Refer to the following illustration.

   ```
   (gdb) attach 3459
   Attaching to process 3459
   [New LWP 3463]
   [New LWP 3464]
   [New LWP 3465]
   [New LWP 3466]
   [Thread debugging using libthread_db enabled]
   Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
   ```

Result

You prepared a debugging session for source code of classic VPUs in an offline simulation application on Linux.

Next steps

You can use GDB in the usual way, for example, to search for functions to debug and to set breakpoints.

When you start the simulation, GDB will stop it at the specified breakpoints, i.e., the simulation time will be paused. To resume, use the `continue` command in GDB.

Related topics

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player..... 298
How to Debug an Adaptive V-ECU..... 245
How to Load an Offline Simulation Application..... 301

References

VEOS Simulator Console Command Reference..... 320

Running VEOS 5.2 in a Linux Docker Container

Where to go from here

Information in this section

Basics on Running VEOS 5.2 in a Linux Docker Container.....308
You can run VEOS 5.2 in a Linux Docker container.

How to Prepare a Linux Docker Container for Running VEOS 5.2.....310
To prepare a Linux Docker container for running VEOS 5.2 on the basis of the delivered Dockerfile.

How to Load and Start a VEOS Simulation in a Linux Docker Container.....312
To load and start a VEOS simulation in a Linux Docker Container.

Basics on Running VEOS 5.2 in a Linux Docker Container

Introduction

You can run VEOS 5.2 in a Linux Docker container. The following paragraphs provide an overview of the prerequisites and required preparations.

Running VEOS 5.2 in a Linux Docker container

To run VEOS 5.2 in a Linux Docker container, you require a Dockerfile that specifies the following:

- A base image
- Instructions for the installation of VEOS 5.2 and the software it requires
- Exposure of the network port required by the VEOS Kernel
- A process manager as the entry point

The VEOS on Linux installation includes a ready-to-use Dockerfile that you can adapt to your requirements.

From the Dockerfile, you can then build a Docker image that lets you create a container in which you can run VEOS 5.2. Refer to How to Prepare a Linux Docker Container for Running VEOS 5.2 on page 310.

For more information on working with Docker containers, refer to the official Docker documentation at https://docs.docker.com/.

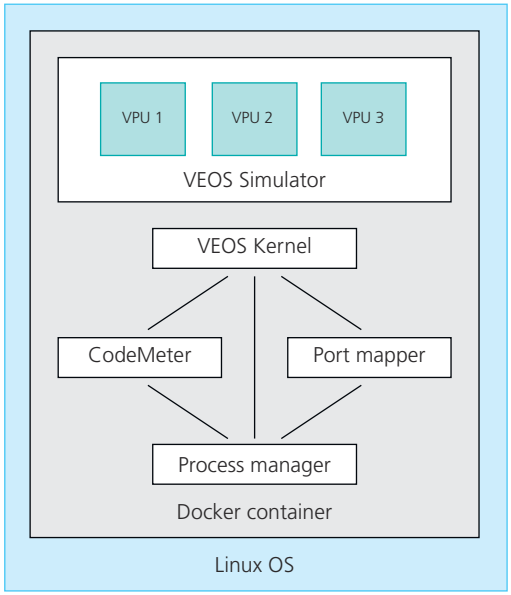
Managing processes In contrast to the usual practice of running only one process per container, the VEOS Simulator requires several processes to be started in the same container in the following sequence:

1. The CodeMeter server process
2. A port mapper, i.e., rpcbind or dspace-portmapper
3. The VEOS Kernel

For starting processes with limited privileges, the VEOS installation creates the `dspace` user.

dSPACE recommends using a process manager, such as supervisor or s6, for managing the processes.

The following illustration shows how VEOS runs in a Linux Docker container with a process manager.



Using a process manager requires that you specify the start sequence and the `dspace` user in the process manager configuration files.

The VEOS on Linux installation is delivered with a Dockerfile that uses the supervisord process manager. The installation also includes several configuration files for supervisord.

Requirements

Software dSPACE recommends using the Ubuntu 18.04 base image for running VEOS 5.2 in a Docker container.

VEOS 5.2 requires the following additional software to run in a Linux Docker container:

- A process manager
- Microsoft .NET Core 3.1
- A port mapper. dSPACE provides the dspace-portmapper for rootless environments, e.g., OpenShift.
- Auxiliary utilities: wget, gnupg2, software-properties-common, apt-utils, curl

Limitations

For general limitations regarding VEOS 5.2 on Linux, refer to Introduction to VEOS on Linux on page 290.

Related topics

Basics

Introduction to VEOS on Linux..... 290

HowTos

How to Prepare a Linux Docker Container for Running VEOS 5.2..... 310

How to Prepare a Linux Docker Container for Running VEOS 5.2

Objective

To prepare a Linux Docker container for running VEOS 5.2 on the basis of the delivered Dockerfile.

Preconditions

In addition to the `dspace-veos-5.2.tar.gz` archive, which contains VEOS 5.2 for Linux, you must have the following:

- A Docker installation on a Linux operating system.
- Access to the Internet or to a local repository that provides the following:
  - Auxiliary packages: wget, gnupg2, software-properties-common, apt-utils, curl, python3-pip, debconf, python3, python3-setuptools
  - supervisord
  - Microsoft .NET Core 3.1
  - The Ubuntu 18.04 Docker base image

The delivered Dockerfile is written for a host with an Internet connection. If you want to use a local repository, you have to add a line to the Dockerfile for adding the repository to the `/etc/apt/sources.list` file.

Possible methods

The method for preparing a Linux Docker container for running VEOS depends on whether you want to simulate adaptive V-ECUs:

- To prepare a Docker container without adaptive V-ECU support, refer to Method 1 on page 310.
- To prepare a Docker container with adaptive V-ECU support, refer to Method 2 on page 311.

Method 1

To prepare a Linux Docker container for running VEOS 5.2 without adaptive V-ECU support

1. Extract the Dockerfile to the same directory to which you saved the `dspace-veos-5.2.tar.gz` archive.
2. Modify the Dockerfile according to your requirements, for example, by adding additional container labels or adding a local package repository.

3 Build a Docker image from the Dockerfile using the following command:

```
docker build --build-arg DS_ACCEPT_EULA=yes --tag dspace/veos/simulator:5.2 \
--target=veos .
```

4 Start a Docker container with the following command:

```
docker run --rm -p 7216:7216/tcp --env DS_CMU_SERVER=<license-server-\
address> -it --name dspace-veos dspace/veos/simulator:5.2
```

- The `--name` option substitutes `dspace-veos` for the container ID.
- `--rm` automatically removes the Docker container when it exits.
- `-p 7216:7216/tcp` maps the TCP port 7216 in the Docker container to port 7216 of the host.
- `--env DS_CMU_SERVER=<license-server-address>` sets the IP address of the CodeMeter license server.

Interim result

You prepared a Linux Docker container for running VEOS 5.2 without adaptive V-ECU support.

Method 2

To prepare a Linux Docker container for running VEOS 5.2 with adaptive V-ECU support

1 Extract the Dockerfile to the same directory to which you saved the `dspace-veos-5.2.tar.gz` archive.

2 Modify the Dockerfile according to your requirements, for example, by adding additional container labels or adding a local package repository.

3 Build a Docker image from the Dockerfile using the following command:

```
docker build --build-arg DS_ACCEPT_EULA=yes --build-arg \
KVM_GID=<KVM group ID> --tag dspace/veos/simulator:5.2-adaptive \
--target=veos_adaptive .
```

Replace `<KVM group ID>` with the group ID (GID) of the `/dev/kvm` device. The GID is specified in the `kvm` entry of the `/etc/group` file. For example, if the entry reads `kvm:x:128:root`, the GID is 128.

4 Start a Docker container with the following command:

```
docker run -u root --privileged --rm -p 7216:7216/tcp \
--env DS_CMU_SERVER=<license-server-address> -it --name \
dspace-veos dspace/veos/simulator:5.2-adaptive
```

- The `-u root --privileged` options start a *privileged* container.

  If you want to start an *unprivileged* container, use the following options instead:

  `-u root --device /dev/kvm:/dev/kvm`

This exposes the `/dev/kvm` device to the container.

Note

- You can also start an *unprivileged* and *rootless* container. In this case, you have to add read and write privileges for all users to the `/dev/kvm` device via `sudo chmod 666 /dev/kvm` first.
  You can then start a container without the `-u root` option, but keeping `--device /dev/kvm:/dev/kvm`.
- Simulating archive-based adaptive V-ECUs in an *unprivileged* container requires more complex modifications. However, you can create a QEMU image from the archives and use this as the implementation layer for the adaptive V-ECU. You can then simulate it in a Docker container created with one of the above commands.
  For more information on creating a QEMU image from archives, refer to Creating a QEMU image on page 243.

- The `--name` option substitutes `dspace-veos` for the container ID.
- `--rm` automatically removes the Docker container when it exits.
- `-p 7216:7216/tcp` maps the TCP port 7216 in the Docker container to port 7216 of the host.
- `--env DS_CMU_SERVER=<license-server-address>` sets the IP address of the CodeMeter license server.

Result

You prepared a Linux Docker container for running VEOS 5.2 with adaptive V-ECU support.

Next steps

You can run a VEOS simulation in the container. Refer to How to Load and Start a VEOS Simulation in a Linux Docker Container on page 312.

Related topics

Basics

Basics on Adaptive V-ECU Implementation Layers 243
Basics on Running VEOS 5.2 in a Linux Docker Container 308
Hypervisor Configuration for Adaptive V-ECU Simulation 296

How to Load and Start a VEOS Simulation in a Linux Docker Container

Objective

To load and start a VEOS simulation in a Linux Docker Container.

Preconditions

You have prepared a Docker container for running VEOS 5.2 as described in How to Prepare a Linux Docker Container for Running VEOS 5.2 on page 310.

Possible methods

There are two possible methods to run a VEOS simulation in a Linux Docker container:

- Using Bash in the container. Refer to Method 1 on page 313.
- Using a remote VEOS Simulator. Refer to Method 2 on page 314.

Method 1

To run a simulation using Bash in the container

1 Copy the OSA file to the Docker container, e.g., by using the

```
docker cp <OSA file path> dspace-veos:/home/dspace
```

command, where you substitute the path to the OSA file on the host for `<OSA file path>`.

2 Open a Bash in the Docker container with the

```
docker exec -it dspace-veos bash
```

command.

The terminal now displays the Bash prompt of the container for the `dspace` user.

```
[redacted]:~$ docker exec -it dspace-veos bash
dspace@7b8b08601d23:~$
```

3 In the container Bash, change to the `/opt/dspace/veos5.2/bin` directory:

```
cd /opt/dspace/veos5.2/bin
```

You can now load and start a simulation with the following command:

```
./veos sim load -s /home/dspace/<OSA file>
```

Replace `<OSA file>` with the OSA file name.

For more information on controlling the VEOS Simulator via the command line, refer to Working with the VEOS Command Line Interface on page 315.

4 When you are finished, exit Bash with the `exit` command.

Tip

Instead of working with Bash in the container, you can also use the `docker exec` command:

```
docker exec dspace-veos /opt/dspace/veos5.2/bin/veos sim
load -s /home/dspace/<OSA file>
```

Method 2

To run a simulation using a remote VEOS Simulator

1 Depending on the operating system of the remote host, execute one of the following commands in a terminal:
   - Linux: `/opt/dspace/veos5.2/bin/veos sim load -s <OSA file path> -h <IP address>`
   - Windows: `veos.exe sim load -s <OSA file path> -h <IP address>`

   Replace `<IP address>` with the IP address of the Docker host.

   VEOS Simulator commands are passed through to the container via the mapped port 7216.

   Note

   The remote simulator must be part of the same network as the Docker host.

Result

You loaded and started an offline simulation application with VEOS 5.2 in a Linux Docker container.

Related topics

Basics

Basics on Running VEOS 5.2 in a Linux Docker Container..... 308
Loading and Running an Offline Simulation Application..... 184

HowTos

How to Prepare a Linux Docker Container for Running VEOS 5.2..... 310

References

VEOS Simulator Console Command Reference..... 320

Working with the VEOS Command Line Interface

Where to go from here

Information in this section

Basics on the VEOS Command Line Interface.....315
The VEOS command line interface lets you build and control an offline simulation application without using the VEOS Player.
You can also use the command line interface to collect support information.

VEOS Build Console Command Reference.....318
Provides an overview of the VEOS Build Console parameters and options.

VEOS Simulator Console Command Reference.....320
Provides an overview of all the VEOS Simulator Console commands and options.

VEOS Support Console Command Reference.....322
Provides an overview of all the VEOS Support Console commands and options.

Basics on the VEOS Command Line Interface

Introduction

The VEOS command line interface lets you build and control an offline simulation application without using the VEOS Player.

You can also use the command line interface to collect support information.

Building an offline simulation application

Building an offline simulation application is possible only with VEOS on Windows. To access the VEOS System Builder via the command line interface, use the `.\veos.exe build` command in the `\Bin` subfolder of the VEOS installation folder.

Instead of specifying parameters and options directly in the command line, you can also write them to a text file and use the `@` character to pass the file to the VEOS System Builder, i.e., `.\veos.exe build @C:\temp\test.txt`.

For a complete list of parameters and options, refer to VEOS Build Console Command Reference on page 318.

Example Executing the following command in the Command Prompt window builds the classic V-ECU implementation container `TestEcu.vecu` and integrates it into the `TestOsa.osa` offline simulation application:

```
.\veos.exe build classic-vecu C:\Temp\TestEcu.vecu –o
C:\Temp\TestOsa.osa
```

Controlling a simulation

Both the Windows and Linux versions of VEOS let you control an offline simulation via command line interface. Use the following commands to access the VEOS Simulator Console:

- Windows: `.\veos.exe sim` in the `\Bin` subfolder of the VEOS installation folder.
- Linux: `./veos sim` in the `/opt/dspace/veos5.2/bin/` folder.

For a complete list of parameters and options, refer to VEOS Simulator Console Command Reference on page 320.

Example Executing the following commands in the command line starts the VEOS Simulator and loads the `MySimulation.osa` offline simulation application:

VEOS Simulator on Windows	VEOS Simulator on Linux
1. `chdir <VEOS installation folder>\Bin` 2. `.\veos.exe sim up` 3. `.\veos.exe sim load MySimulation.osa`	1. `cd /opt/dspace/veos5.2/bin/` 2. `./veos sim up` 3. `./veos sim load MySimulation.osa`

Collecting support information

Both the Windows and Linux versions of VEOS provide the VEOS Support Console for writing support information to an archive file. This makes it easier for you to send information to dSPACE Support for troubleshooting. The support information can include log files, config files, crash dumps, and system information.

The following table shows you how to access the VEOS Support Console and in which format the output is saved, depending on your operating system.

Operating System	Call	Output Format and Folder
Windows	`.\veos.exe support collect` from the `\Bin` subfolder of the VEOS installation folder	ZIP file saved to `C:\Users\<user>\AppData\Local\Temp`
Linux	`./veos support collect` from the `/opt/dspace/veos5.2/bin` folder	TAR.XZ file saved to `/tmp`

For a complete list of commands and options, refer to VEOS Support Console Command Reference on page 322.

Example Executing the following command from the `\Bin` subfolder of the VEOS installation folder on Windows saves support information to the `C:\Users\<user>\AppData\Local\Temp` folder:

```
.\veos.exe support collect --no-system-info
```

System information is omitted, which can accelerate the collection process.

Refer to the following illustration.

```
PS C:\Program Files\dSPACE VEOS 5.2\Bin> .\veos.exe support collect --no-system-info
Collecting files from "C:\Users\      \AppData\Local\dspace\5914CD6B-6194-458E-9535-AA948F4CC590\VEOS" ...
Collecting files from "C:\ProgramData\dspace\5914CD6B-6194-458E-9535-AA948F4CC590\dSPACEInstaller" ...
Collecting files from "C:\ProgramData\dspace\5914CD6B-6194-458E-9535-AA948F4CC590\Log" ...
Collecting files from "C:\ProgramData\dspace\veos\logs" ...
Collecting files from "C:\ProgramData\dspace\5914CD6B-6194-458E-9535-AA948F4CC590\VEOS" ...
Collecting files from "C:\ProgramData\dspace\veos\config" ...
Collecting files from "C:\Program Files\dSPACE VEOS 5.2\installationset\info" ...
Support information written to "C:\Users\      \AppData\Local\Temp\VeosSupportInfo20210127090542024.zip".
```

VEOS environment

On Linux, you can activate the VEOS environment with the `. /opt/dspace/veos5.2/tools/shell/veos-env.sh` command or, in Bash, with `source /opt/dspace/veos5.2/tools/shell/veos-env.sh`. This lets you use the `veos sim` and `veos support` commands from any directory as well as all the regular shell commands. In Bash, you can also use the **Tab** key for autocompletion.

Use the `dspace_veos_deactivate` command to deactivate the VEOS environment.

Related topics

Basics

Basics on Integrating the Simulation System 62
Basics on Signal Connection Dependencies 100
Introduction to VEOS on Linux 290
Loading and Running an Offline Simulation Application 184

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player 298
How to Load an Offline Simulation Application 301

References

VEOS Build Console Command Reference 318
VEOS Simulator Console Command Reference 320
VEOS Support Console Command Reference 322

VEOS Build Console Command Reference

Building an offline simulation application

The VEOS Build Console is available only for Windows.

Use the `.\veos.exe build` command in the `\Bin` subfolder of the VEOS installation folder to call the VEOS Build Console.

Parameters

You must use one of the following parameters to specify the container type to be built:

Parameter	Description
classic-vecu	Classic V-ECU implementation (VECU or CTLGZ file).
adaptive-vecu	Adaptive V-ECU implementation (VECU or CTLGZ file).
fmu	Functional Mock-up Unit (FMU file).
smc	System model container (SMC file).
sic	Simulink implementation container (SIC file).
bsc	Bus simulation container (BSC file).

Options

Depending on the container type you specified, the following options are available:

Option	Description
--help	Displays the available options for the specified parameter.
--version	Displays version information on the VEOS Build Console.
show-targets	Displays all available simulation targets and indicates the default target.
-o, --output-file	Lets you specify the path to the OSA output file. Per default, the OSA file is saved in the directory from which you call the VEOS Build Console under the same name as the container from which it is built. If the OSA file already exists, the VPU is integrated into the existing OSA file.
--osa-author	Lets you specify the author of the OSA to be built. Per default, the current user name is used.
--build-directory	Lets you specify the directory where all build-related files are saved. If you do not specify a directory, VEOS creates a temporary directory under `\Users\<current user>\AppData\Local\Temp\dSPACE\VEOS\5.2\veos-build`.
--clean-after-successful-build	Removes all temporary files after a successful build. After a failed build, the files are not removed.
--log-file-path	Lets you specify the path to the build log file. If not set, no log file is created.
-t, --target	Lets you specify the simulation target. If not specified, the default target it used.
--compiler-options	Lets you specify the C compiler options as a single string, for example: `--compiler-options="-OPT1 -OPT2"`.
--cpp-compiler-options	Lets you specify the C++ compiler options as a single string, for example: `--cpp-compiler-options="/OPT1 /OPT2"`.
--preprocessor-defines	Lets you specify the preprocessor definitions, for example: `--preprocessor-defines="MY_DEFINE1=VALUE1 MY_DEFINE2=VALUE2"`.
--additional-code-files	Lets you specify additional code files to be used during build. You can pass multiple files. Invalid entries are ignored.

Option	Description
--code-coverage-level	*Available only for classic V-ECUs.* Lets you specify the code coverage level. Possible values are: ▪ `None` (default) ▪ `FunctionCoverage` ▪ `DecisionCoverage` ▪ `MultiConditionCoverage`
--show-all-warnings	Lets you specify whether all warnings are shown during the build. The default value is `false`.
--configuration	Lets you specify whether debug information is created during the build. Possible values are `Release` and `Debug` (default).
--print-options	Displays the selected options. The default value is `false`.
--hypervisor-connection-state	*Available only for adaptive V-ECUs.* Lets you specify the connection type of the hypervisor for accessing the network. Possible values are: ▪ `Disconnected` ▪ `HostOnly` ▪ `Bridged` ▪ `NAT` For more information, refer to Hypervisor Connection on page 156. **Note** Only one adaptive V-ECU per simulation can have a hypervisor connection other than Disconnected.
--mac-address-range	*Available only for adaptive V-ECUs.* Lets you specify the range of MAC addresses to be used for the virtual machine hosted by the hypervisor. You must provide the value in hex notation, e.g., `00:AB:12:CD`. When you select a connection type other than `Disconnected`, the specified range is used to generate MAC addresses for the hypervisor Ethernet adapter in the local network of the host PC. The default value is `00:50:56:00`.
--xcp-service-port	*Available only for classic V-ECUs and Simulink implementation containers.* Lets you specify the XCP service port. If you do not specify a port number, a random port number is generated during the build process.

Related topics

Basics

Basics on Integrating the Simulation System..... 62
Basics on the VEOS Command Line Interface..... 315

HowTos

How to Build an Offline Simulation Application for Linux with the VEOS Player..... 298

VEOS Simulator Console Command Reference

Controlling a simulation

Use the following commands to access the VEOS Simulator Console:

- On Windows: `.\veos.exe sim` in the `\Bin` subfolder of the VEOS installation folder.
- On Linux: `./veos sim` in the `/opt/dspace/veos5.2/bin` folder or simply `veos sim` from within any directory if the VEOS environment is active.

VEOS environment on Linux The VEOS environment is available only for Linux. You can activate it via `. /opt/dspace/veos5.2/tools/shell/veos-env.sh` or, in Bash, via `source /opt/dspace/veos5.2/tools/shell/veos-env.sh`.

The prompt is modified by the `(dspace-veos)` prefix to indicate that the VEOS environment is active.

To deactivate the VEOS environment, use the `dspace_veos_deactivate` command.

The VEOS environment lets you use the veos-sim and veos-model commands from within any directory as well as all the regular shell commands. In the case of Bash, you can also use the **Tab** key for autocompletion.

Simulation control commands

Command	Options	Description
up		Starts the VEOS Simulator.
	Standard options	Refer to Standard options on page 322.
down		Stops the VEOS Simulator.[1)]
	Standard options	Refer to Standard options on page 322.
load <OSA>		Loads offline simulation application <OSA>.
	Standard options	Refer to Standard options on page 322.
	-b, --break	Breaks after starting the host processes.
	-c, --continue	Continues after starting the host processes.
	-s, --start	Starts the simulation after load.
unload		Unloads the offline simulation application.
	Standard options	Refer to Standard options on page 322.
config		Lets you configure the simulation.
	Standard options	Refer to Standard options on page 322.
	-a, --acceleration-factor	Sets the simulation acceleration factor.
	-p, --pause-time	Pauses the simulation at the specified time.
	-t, --stop-time	Stops the simulation at the specified time.
	-e, --step-time	Sets the time interval of a simulation step.
	--enable-sim-log	Enables simulation logging.
	--disable-sim-log	Disables simulation logging.
	--enable-bus-log	Enables bus logging.
	--disable-bus-log	Disables bus logging.
	--enable-xcp-log	Enables XCP logging.
	--disable-xcp-log	Disables XCP logging.
start		Starts the simulation.
	Standard options	Refer to Standard options on page 322.
stop		Stops the simulation.
	Standard options	Refer to Standard options on page 322.
pause		Pauses the simulation.
	Standard options	Refer to Standard options on page 322.
step		Advances the simulation by one step.
	Standard options	Refer to Standard options on page 322.
show-log		Displays logging information if you activated logging when you loaded an OSA.
	Standard options	Refer to Standard options on page 322.
info		Displays information on the following items: ▪ The VEOS host ▪ Simulation settings and state ▪ The loaded OSA and its components
	Standard options	Refer to Standard options on page 322.
scan		Scans for VEOS Simulators in the same subnet.
	Standard options	Refer to Standard options on page 322.

Command	Options	Description
help		Displays the list of available commands.
	Standard options	Refer to Standard options on page 322.
version		Displays version information.
	Standard options	Refer to Standard options on page 322.

[1] You can also stop the simulator by pressing **`Ctrl+C`** in the terminal that runs the simulator.

Standard options The following standard options are available for all simulation control commands:

Option	Description
-h, --host	(Default: 127.0.0.1) Lets you specify the host name or IP address of the VEOS Simulator.
-v, --verbosity	Lets you specify which messages are printed to standard output. Possible values are: ▪ Trace: All messages are printed. ▪ Info (default): Info messages, warnings, and errors are printed. ▪ Warning: Warnings and errors are printed. ▪ Error: Errors are printed.
--help	Displays a list of available options.
--version	Displays version information.

Related topics

Basics

Basics on Simulating Adaptive V-ECUs..... 236
Basics on the VEOS Command Line Interface..... 315
Loading and Running an Offline Simulation Application..... 184

HowTos

How to Load an Offline Simulation Application..... 301
How to Select a Remote Simulator..... 186

VEOS Support Console Command Reference

Collecting support information

Use the following commands to access the VEOS Support Console:

- On Windows: `.\veos.exe support` in the `\Bin` subfolder of the VEOS installation folder.
- On Linux: `./veos support` in the `/opt/dspace/veos5.2/bin` folder, or `veos support` from any directory if the VEOS environment is active.

VEOS environment on Linux The VEOS environment is available only for Linux. You can activate it via `. /opt/dspace/veos5.2/tools/shell/veos-env.sh` or, in Bash, via `source /opt/dspace/veos5.2/tools/shell/veos-env.sh`.

The prompt is modified by the `(dspace-veos)` prefix to indicate that the VEOS environment is active.

To deactivate the VEOS environment, use the `dspace_veos_deactivate` command.

The VEOS environment lets you use the veos-sim and veos-model commands from any directory as well as all the regular shell commands. In Bash, you can also use the **Tab** key for autocompletion.

Commands

Command	Options	Description
collect		Collects support information.
	--no-crash-dumps	Excludes crash dumps.
	--no-system-info	Excludes system information.[1]
	--no-config-files	Excludes config files.
	--no-log-files	Excludes log files.
	--help	Displays a list of available options.
	--version	Displays version information.
help		Displays the list of available commands.
	Standard options	Refer to Standard options on page 323.
version		Displays version information.
	Standard options	Refer to Standard options on page 323.

[1] Collecting system information can take a long time.

Standard options The following standard options are available for all veos-support commands:

Option	Description
--help	Displays a list of available options.
--version	Displays version information.

Related topics

Basics

Basics on the VEOS Command Line Interface..... 315

References

VEOS Simulator Console Command Reference..... 320

Automating the VEOS Player

Where to go from here

Information in this section

Basics and Instructions.....326
Migrating Python Scripts from Python 3.6 to Python 3.9.....332
API Reference Information.....337
Limitations.....380

Basics and Instructions

Where to go from here

Information in this section

Basics of the VEOS Player Automation Interface.....326
VEOS Player lets you automate most of its features via its automation interface.

Demos for Automating the VEOS Player.....328
The VEOS installation provides some Python demo scripts that show you how to automate the VEOS Player.

Automating Different VEOS Player Versions.....329
To automate a specific VEOS Player version, you can supply version information when opening the COM connection.

Programming VEOS Player Automation.....330
Provides an overview of how to program VEOS Player automation.

Basics of the VEOS Player Automation Interface

Introduction

VEOS Player lets you automate most of its features via its automation interface.

VEOS Player features that can be automated

The VEOS Player lets you automate most of its features:

- Loading an offline simulation application (OSA) file
- Configuring the simulation
- Configuring the VPUs in the simulation system
- Importing VPUs to the simulation system
- Connecting VPU ports
- Starting, pausing, and stopping a simulation run
- Saving the configuration as an OSA file.

Accessing the VEOS Player's automation interface

You can access the VEOS Player's automation interface:

- Via an external Python interpreter such as PythonWin
- Via applications or further scripting environments

Note

Scripts are case-sensitive. Script text must match the spelling of the API documentation.

Accessing the running VEOS Player To access the running VEOS Player from an external interpreter, you have to open a COM connection:

```
from win32com.client import Dispatch
player = Dispatch("VeosPlayer.Application")
```

When you execute the listing above in an external Python interpreter, the VEOS Player is started if it is not already running.

Refer to IApplication on page 343.

Supported automation languages

The VEOS Player's automation interface is an API which is implemented as a COM object model. The Microsoft Component Object Model (COM) supports communication between objects from different applications. It can be used by any COM-compatible application, regardless of the programming language in which it was developed.

You can automate the VEOS Player with programming languages such as the following:

- Python scripts running in an external interpreter, for example, PythonWin
- M scripts running in MATLAB
- VBA scripts running in Microsoft Office

You can automate the VEOS Player by applications developed, for example, with the following programming languages:

- C#
- C++
- Visual Basic

Tip

If you use a programming environment that supports the IntelliSense feature, such as Microsoft Visual Studio, this feature provides information about the VEOS Player's API.

Related topics

Basics

Automating Different VEOS Player Versions 329
Demos for Automating the VEOS Player 328
Programming VEOS Player Automation 330

References

IApplication 343

Demos for Automating the VEOS Player

Introduction

The VEOS installation provides some Python demo scripts that show you how to automate the VEOS Player.

Features covered by the demos

The following table lists features of the VEOS Player's automation interface, and shows in which demo each of them is covered (✓) or not (—).

Feature	1Ecu_Standard	2Ecu_CanBusSimulation	2Ecu_LinBusSimulation	2Ecu_VpuPorts	AdaptiveSamples	AR_Fuelsys_Smc	MDAutomotive	ThrottleControl
Loading an offline simulation application	✓	✓	✓	✓	✓	—	—	✓
Importing V-ECU implementations	—	—	—	—	✓	—	—	✓
Importing Simulink implementation containers	—	—	—	—	—	—	✓	—
Importing VPUs	—	✓	✓	—	—	—	—	—
Connecting VPUs	—	—	—	✓	✓	—	—	—
Starting and controlling an offline simulation	✓	—	—	—	✓	—	—	✓
Getting version information on VEOS Player	—	—	—	—	—	—	—	✓
Handling VEOS Player events	✓	—	—	—	—	—	—	—

Decrypting Windows demos and copying them to a user-specific folder

You can work with the Windows demos only if they have been decrypted in dSPACE Installation Manager. Decryption has to be performed only once. For information on decryption, refer to How to Decrypt Encrypted Archives of dSPACE Software Installations (Managing dSPACE Software Installations 🕮).

The Copy Demo Files command lets you copy the demo files to a user-specific folder.

Related topics

Basics

Basics of the VEOS Player Automation Interface..... 326

References

Copy Demo Files..... 392

Automating Different VEOS Player Versions

Introduction

To automate a specific VEOS Player version, you can supply version information when opening the COM connection.

Automating the VEOS Player

To automate VEOS Player, you use the `Dispatch` command to open a COM connection for using the API of the VEOS Player. The Python listing below shows how you can open a COM connection to the VEOS Player.

```
# Import required modules.
from win32com.client import Dispatch
# Open a COM connection to the VEOS Player.
player =  Dispatch("VeosPlayer.Application")
```

Automating a specific VEOS Player version

You can automate a specific version of VEOS Player by supplying its version number via the `Dispatch` command.

The Python listing below shows how you can automate a specific version of VEOS Player. VEOS Player version 5.2 is used as an example.

```
# Open a COM connection to VEOS Player.
player =  Dispatch("VeosPlayer.Application.5.2")
```

Automating the active VEOS Player version

Active VEOS Player version If you have several VEOS Player versions on your PC, one is the *active version*. The latest installed version of VEOS Player is the active version by default regardless of its version number.

- When you remove the currently active version of VEOS Player, there is no active version because VEOS Player does not activate a version automatically. See below for information on activating another VEOS Player version.
- When you remove a currently inactive version of VEOS Player, the currently active version of VEOS Player remains the same.

The *active VEOS Player version* is the version that you access via `VeosPlayer.Application`.

Activating another VEOS Player version To use a script for a specific VEOS Player version without supplying version information via the `Dispatch` command, you can activate that version with dSPACE Installation Manager.

For instructions, refer to How to Activate a Single dSPACE Installation (Managing dSPACE Software Installations 🕮).

Related topics

Basics

Basics of the VEOS Player Automation Interface..... 326

References

IApplication..... 343

Programming VEOS Player Automation

User interface handling

The following table shows where you can find information on how to automate user interface handling:

Purpose	Refer to
You can automate the start of the VEOS Player.	Automating the Start of the VEOS Player, and Getting Version Information on page 414

Integrating the simulation system

The following table shows where you can find information on how to automate simulation system integration:

Purpose	Refer to
You can automate: ▪ Opening an offline simulation ▪ Saving an offline simulation	Automating the Opening and Saving of an Offline Simulation Application on page 169
You can automate: ▪ The import of VPUs ▪ The import of a model implementation such as a V-ECU implementation ▪ The removal of VPUs	Automating the Import and Removal of VPUs on page 170
You can automate the connection of VPU ports.	Automating the Connection of VPUs on page 172

Loading and running an offline simulation application

The following table shows where you can find information on how to automate the loading and running of an offline simulation application:

Purpose	Refer to
You can automate the configuration of an offline simulation.	Automating the Simulation Configuration on page 226

Purpose	Refer to
You can automate: ▪ The download of an offline simulation application ▪ The control of an offline simulation by starting, pausing and stopping it	Automating the Download and Control of an Offline Simulation Application on page 227
You can handle events such as `OnSimulationStarted` in automation scripts.	Automating VEOS Player Event Handling with Python on page 228

Related topics

Basics

Basics of the VEOS Player Automation Interface..... 326

Migrating Python Scripts from Python 3.6 to Python 3.9

Where to go from here

Information in this section

Main Changes in Python 3.9 332
The end of life of Python 3.6 is scheduled for the end of December 2021. Therefore, dSPACE switches to Python 3.9 with dSPACE Release 2021-A.

Main Changes in Handling Python 3.9 with dSPACE Software 333
With Python 3.9, some aspects of handling Python with dSPACE software have changed.

General Information on Using Python Installations 334
If you work with dSPACE software from dSPACE Release 2020-B or earlier, which supports Python 3.6, and dSPACE software from dSPACE Release 2021-A or later, which supports Python 3.9, both Python versions are installed on the PC and can be used in parallel.

Technical Changes 335
Migrating from Python 3.6 to Python 3.9 has caused minor changes in the Python API.

Main Changes in Python 3.9

Main reason for using Python 3.9 instead of Python 3.6

The end of life of Python 3.6 is scheduled for the end of December 2021. Therefore, dSPACE switches to Python 3.9 with dSPACE Release 2021-A.

Related documents available on the Python website

For a short overview of the main changes in Python 3.9, refer to Technical Changes on page 335. For information on all the changes in the Python language and environment from Python 3.6 to Python 3.9, refer to Python Software Foundation at www.python.org.

The Python Software Foundation provides the following documents:

- What's New from Python 3.6 to 3.7:
  https://docs.python.org/3.7/whatsnew/3.7.html
- What's New from Python 3.7 to 3.8:
  https://docs.python.org/3.8/whatsnew/3.8.html

- What's New from Python 3.8 to 3.9:
  https://docs.python.org/3.9/whatsnew/3.9.html

Related topics

Basics

Technical Changes..... 335

Main Changes in Handling Python 3.9 with dSPACE Software

Introduction

With Python 3.9, some aspects of handling Python with dSPACE software have changed.

Libraries

The libraries and components used with Python 3.9 and distributed on dSPACE DVDs have changed as shown in the following table.

Package	Python 3.6 (Release 2020-B)	Python 3.9 (Release 2021-A)
comtypes	1.1.7	1.1.7
Core	3.6.8	3.9.1
cycler	0.10.0	0.10.0
future	0.18.2	0.18.2
grpcio	1.29.0	1.34.0
grpcio_tools	1.29.0	1.34.0
kiwisolver	1.2.0	1.3.1
lxml	–	4.6.2
matplotlib	3.2.1	3.3.3
numpy	1.18.4	1.19.3
pillow	7.1.2	8.0.1
pip	20.1.1	20.3.1
protobuf	3.12.2	3.14.0
pycparser	–	2.20
pyglet	1.5.5	1.5.11
pyparsing	2.4.7	2.4.7
pypubsub	4.0.3	4.0.3
Python-dateutil	2.8.1	2.8.1
pythonnet	2.4.2	2.5.3
pytz	2020.1	2020.4

Package	Python 3.6 (Release 2020-B)	Python 3.9 (Release 2021-A)
pywin32	227.10	300.10
scipy	1.4.1	1.5.4
six	1.15.0	1.15.0
wxPython	4.1.0	4.1.1
yapsy	1.12.2	1.12.2

General Information on Using Python Installations

Introduction

If you work with dSPACE software from dSPACE Release 2020-B or earlier, which supports Python 3.6, and dSPACE software from dSPACE Release 2021-A or later, which supports Python 3.9, both Python versions are installed on the PC and can be used in parallel.

Limitations when using Python 3.6 and Python 3.9 in parallel

You can use both Python versions in parallel. However, you must observe the following limitations:

- You can set the file associations for PY and PYW files to only one Python version. This is usually the latest Python version you installed.

  Note

  If you install dSPACE Release 2021-A including SystemDesk, Python 3.6 will be installed last. Therefore, PythonWin 3.6 is registered to open Python files. To set the file associations to PythonWin 3.9, execute the `PythonInstaller.exe` from `<DVDRoot>\Products\Common\Python3.9` again or use the **`Edit with PythonWin 3.9`** command on the context menu of a Python file.

- Environment variables are used by both Python versions. You must set their values, for example, for PYTHONHOME, to the Python installation you want to work with. For an overview of environment variables set by Python, refer to: https://docs.python.org/3.9/using/windows.html.

Note

To avoid unintentional effects, which can be difficult to identify, you are recommended to not use environment variables. For the same reason, you should not extend the system path by a Python installation path.

Using dSPACE test automation with both Python versions in parallel

If a test automation script uses dSPACE Python Modules and you do not want to migrate the script, you have to work with both Python versions in parallel. The dSPACE Python Modules for Python 3.6 are available up to and including Release 2020-B.

Note

It is recommended to use only dSPACE software from the same Release when using Python scripts. Using both Python versions in the same application context, such as automating the access to an older version of ControlDesk via AutomationDesk or using AutomationDesk with an activated Real-Time Testing version that does not support Python 3.9, might cause a conflict.

Technical Changes

Introduction

Migrating from Python 3.6 to Python 3.9 has caused minor changes in the Python API.

Note

Independently of the number of required manual modifications, you must test the migrated code.

Migration issues

The following changes in Python 3.9 require manual migration of your Python scripts.

Python.NET: Return value of IList type is converted to PyList Since Python.NET 2.4, return values of `SystemCollections.Generic.IList` type have been converted to the `PyList` type. Because some methods and properties of an `IList` object are not available with a `PyList` object, dSPACE provided the modified Python.NET 2.4.1 and 2.4.2 packages, which suppressed the conversion.

As of Python.NET 2.5.3, the modifications by dSPACE are no longer provided. You must therefore migrate the methods and properties of an `IList` object to the related methods and properties of a `PyList` object. For example, you must use `len(MyObject)` instead of `MyObject.Count`.

Note

The created Python list is a copy of the .NET object. Modifications to the Python list therefore have no effect on the associated .NET object. To apply the modifications to the .NET object, assign the Python list afterward.

ctypes: Loading library after changing the PATH variable Loading a library via `ctypes` can fail if the path to the library was added to the PATH variable beforehand.

For loading a library in a safe manner, use `ctypes.WinDLL(NameOfDll, winmode=8)` instead of `ctypes.windll.LoadLibrary(NameOfDll)`.

API Reference Information

Automation Interfaces

Where to go from here

Information in this section

BuildStatusEnum 339
Provides enumeration values for related automation interfaces.

BusType 340
Provides enumeration values for related automation interfaces.

CommunicationPackageProtocolType 340
Provides enumeration values for related automation interfaces.

CommunicationPackageType 341
Provides enumeration values for related automation interfaces.

IAdaptiveVEcu 341
Provides properties and methods for automation.

IAdaptiveVEcus 342
Provides properties and methods to manage related automation interfaces.

IAdaptiveVEcuSettings 343
Provides properties and methods for automation.

IApplication 343
Provides properties and methods for automation.

IAutomationBase 345
Provides properties and methods for automation.

IBuildResult 345
Provides properties and methods for automation.

IBusChannel 346
Provides properties and methods for automation.

IBusChannels 346
Provides properties and methods to manage related automation interfaces.

IBusCluster 347
Provides properties and methods for automation.

IBusClusters 348
Provides properties and methods to manage related automation interfaces.

IBusConnector.....349
Provides properties and methods for automation.

IBusConnectors.....350
Provides properties and methods to manage related automation interfaces.

ICommunicationPackage.....351
Provides properties and methods for automation.

ICommunicationPackages.....351
Provides properties and methods to manage related automation interfaces.

IConnectionList.....353
Provides properties and methods for automation.

IConnectionResult.....353
Provides properties and methods for automation.

IExportSettings.....354
Provides properties and methods for automation.

IImportSettings.....354
Provides properties and methods for automation.

IMessage.....356
Provides properties and methods for automation.

IMessages.....357
Provides properties and methods to manage related automation interfaces.

IPlayerAutomationInterface.....359
Provides properties and methods for automation.

IPlayerAutomationRegistrationInterface.....359
Provides properties and methods for automation.

IProject.....359
Provides properties and methods for automation.

IProjects.....363
Provides properties and methods to manage related automation interfaces.

ISimulationConfiguration.....364
Provides properties and methods for automation.

ISimulator.....365
Provides properties and methods for automation.

ISimulatorEvents.....367
Provides properties and methods for automation.

IVersionInformation.....368
Provides properties and methods for automation.

IVpu....368
Provides properties and methods for automation.

IVpuItem....370
Provides properties and methods for automation.

IVpuPort....372
Provides properties and methods for automation.

IVpuPortConnection....373
Provides properties and methods for automation.

IVpuPortGroup....374
Provides properties and methods for automation.

IVpuPortGroups....375
Provides properties and methods to manage related automation interfaces.

IVpuPorts....376
Provides properties and methods to manage related automation interfaces.

IVpus....377
Provides properties and methods to manage related automation interfaces.

SerialBaudRate....378
Provides enumeration values for related automation interfaces.

SimulationProgress....378
Provides enumeration values for related automation interfaces.

VpuItemType....379
Provides enumeration values for related automation interfaces.

VpuPortDirection....379
Provides enumeration values for related automation interfaces.

BuildStatusEnum

Description The build status.

Enumeration values The enumeration has the following values:

Name	Description	Value
Valid	Build was successfully.	0
Invalid	Build stopped with errors.	1

Returned by

The element is returned by properties or methods of the following elements:

- IBuildResult (refer to IBuildResult on page 345)

BusType

Description

The bus types enumeration.

Enumeration values

The enumeration has the following values:

Name	Description	Value
Can	The CAN bus type.	0
Idealized	The idealized bus type.	1
Lin	The LIN bus type.	2
Eth	The Ethernet bus type.	3

CommunicationPackageProtocolType

Description

Enumeration for the communication package protocol

Enumeration values

The enumeration has the following values:

Name	Description	Value
NonBlocking	The non-blocking communication package type.	0
Blocking	The blocking communication package type.	1

Returned by

The element is returned by properties or methods of the following elements:

- ICommunicationPackage (refer to ICommunicationPackage on page 351)

CommunicationPackageType

Description

Enumeration of all communication package types.

Enumeration values

The enumeration has the following values:

Name	Description	Value
LegacyCommunicationPackage	The Legacy communication package type.	0
StandardCommunicationPackage	The Standard communication package type.	1
UserCommunicationPackage	The User communication package type.	2

Returned by

The element is returned by properties or methods of the following elements:

- ICommunicationPackage (refer to ICommunicationPackage on page 351)

IAdaptiveVEcu

Description

Specifies the COM interface for adaptive VEcu entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Author	Gets the author.	Get	*String*
CommunicationControllers	Gets the bus communication controllers to this VPU.	Get	IBusConnectors (refer to IBusConnectors on page 350)
Description	Gets the description.	Get	*String*
HypervisorConnectionType	Gets the HypervisorConnectionType.	Get	*String*
MacRangePrefix	Gets the MacRangePrefix.	Get	*String*
MemorySize	Gets the size of memory (RAM) used by the adaptive VEcu.	Get	*Unsigned 32 Bit Integer*
Name	Gets the name.	Get	*String*
NumberOfCpuCores	Gets the number of CPU Cores used by the adaptive VEcu.	Get	*Unsigned 32 Bit Integer*
StepSize	Gets the step size.	Get	*Double*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IAdaptiveVEcus (refer to IAdaptiveVEcus on page 342)
- IProject (refer to IProject on page 359)

IAdaptiveVEcus

Description

Specifies the COM interface for a collection of adaptive VEcus.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IAdaptiveVEcu[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Item	Gets the element with the specified name within the collection.	▪ *<String>* `nameParam`: The name of the element to get.	Returns the element with the specified name within the collection, if found; otherwise, null. ▪ IAdaptiveVEcu (refer to IAdaptiveVEcu on page 341)

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

IAdaptiveVEcuSettings

Description

This interface provides adaptive VEcu settings.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
HypervisorConnectionType	Gets or sets the HypervisorConnectionType.	Get/Set	*String*
HypervisorConnectionTypeList	Gets the HypervisorConnectionTypeList by string.	Get/Set	*System.String[]*
MacRangePrefix	Gets or sets the MacRangePrefix.	Get/Set	*String*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
SetHypervisorConnectionTypeList	Sets the HypervisorConnectionTypeArray.	▪ *<System.String[]>* `hypervisorConnectionTypeArray`: The HypervisorConnectionTypeArray.	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IImportSettings (refer to IImportSettings on page 354)

IApplication

Description

General information on the Application element.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
ApplicationRootDir	Gets the root directory of the application. Determined from the path name of the VEOS Player executable.	Get	*String*
ExportVariableDescriptionFilesAfterSaving	Gets or sets a value indicating whether export variable description files after saving.	Get/Set	*Boolean*
Projects	Gets the project collection element.	Get	IProjects (refer to IProjects on page 363)

Name	Description	Get/Set	Type
Simulator	Gets the simulator element.	Get	ISimulator (refer to ISimulator on page 365)
Version	Gets the VEOS Player version.	Get	*String*
VersionInformation	Gets the VEOS Player version information.	Get	IVersionInformation (refer to IVersionInformation on page 368)
Visible	Gets or sets a value indicating whether the GUI is visible.	Get/Set	*Boolean*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
CopyDemoFiles	This method copies all available demos archives to the veos user folder and also extracts the archives.	None	None
Quit	This method closes the applications main form.	None	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IAutomationBase (refer to IAutomationBase on page 345)
- IPlayerAutomationInterface (refer to IPlayerAutomationInterface on page 359)
- ISimulator (refer to ISimulator on page 365)

Related topics

Basics

Automating Different VEOS Player Versions..... 329
Automating the Opening and Saving of an Offline Simulation Application..... 169
Automating the Start of the VEOS Player, and Getting Version Information..... 414

IAutomationBase

Description

The base element of the automation.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Root	Gets or sets the VEOS Player application element.	Get/Set	IApplication (refer to IApplication on page 343)

Methods

The element has no methods.

IBuildResult

Description

Provides all build information.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
BuildOutput	Gets the compiler and linker output (make) generated during build process.	Get	*String*
BuildStatus	Gets the Status of the build (e.g. valid/invalid).	Get	BuildStatusEnum (refer to BuildStatusEnum on page 339)
ImportedAdaptiveVEcus	Gets the imported adaptive vecus.	Get	*System.String[]*
ImportedApplicationProcesses	Gets the imported application processes.	Get	*System.String[]*
Messages	Gets the messages occurred during build procedure.	Get	IMessage (refer to IMessage on page 356)
OsaFile	Gets the Offline Simulation Application (OSA) file.	Get	*String*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)
- IVpu (refer to IVpu on page 368)

IBusChannel

Description

Specifies the COM interface for bus cluster entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Connectors	Gets a collection of bus connectors, were this channel is connected to.	Get	IBusConnectors (refer to IBusConnectors on page 350)
Id	Gets the ID of the bus channel.	Get	*System.Guid*
Name	Gets the name of the bus channel.	Get/Set	*String*
Type	Gets the type of the bus channel, e.g. CAN.	Get	*String*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IBusChannels (refer to IBusChannels on page 346)

IBusChannels

Description

Specifies the COM interface for a collection of bus clusters.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IBusChannel[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Item	Gets the element with the specified name within the collection.	▪ *<String>* `nameParam`: The name of the element to get.	The element with the specified name within the collection, if found; otherwise, null. ▪ IBusChannel (refer to IBusChannel on page 346)

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IBusCluster (refer to IBusCluster on page 347)

IBusCluster

Description

Specifies the COM interface for bus cluster entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Channels	Gets a collection of bus channels that are used in the cluster.	Get	IBusChannels (refer to IBusChannels on page 346)
Id	Gets the ID of the bus cluster.	Get	*System.Guid*
Name	Gets the name of the bus cluster.	Get/Set	*String*
Type	Gets the type of the bus cluster, e.g. CAN.	Get	*String*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
AddBusChannel	Adds a new bus channel.	None	None
RemoveBusChannel	Removes a bus channel.	▪ <IBusChannel (refer to IBusChannel on page 346)> `busChannel`: Bus channel to remove.	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IBusClusters (refer to IBusClusters on page 348)

IBusClusters

Description

Specifies the COM interface for a collection of bus clusters.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IBusCluster[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Add	Adds a bus cluster.	▪ <BusType (refer to BusType on page 340)> `busType`: The bus cluster type. ▪ <*String*> `clusterName`: Name of the bus cluster.	None
Add	Adds a bus cluster with a default name.	▪ <BusType (refer to BusType on page 340)> `busType`: The bus cluster type.	None
Item	Gets the element with the specified name within the collection.	▪ <*String*> `nameParam`: The name of the element to get.	The element with the specified name within the collection, if found; otherwise, null. ▪ IBusCluster (refer to IBusCluster on page 347)
Remove	Removes the cluster with the specific name. Is considered to be thread safe	▪ <*String*> `nameParam`: Name of the cluster.	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

IBusConnector

Description

Specifies the COM interface for bus connector entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
BaudRate	Gets the baud rate of the bus connector in bits per second.	Get	*Unsigned 64 Bit Integer*
BusConfigurationName	Gets the bus configuration name.	Get	*String*
BusConfigurationPart	Gets the bus configuration part.	Get	*String*
CommunicationMatrixContentHash	Gets the communication matrix content hash.	Get	*String*
CommunicationMatrixName	Gets the communication matrix name.	Get	*String*
ConnectedTo	Gets the name of the bus channel this controller is connected to.	Get	*String*
Id	Gets the bus controller ID on the bus.	Get	*Unsigned 32 Bit Integer*
Name	Gets the name of the bus connector.	Get	*String*
Origin	Gets the original name of the element.	Get	*String*
Type	Gets the bus type, e.g. CAN.	Get	*String*
UniqueId	Gets the unique ID of this bus controller. This ID is used the reference the controller that are connected to a VPU.	Get	*String*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
ConnectToChannel	Changes the bus channel connection.	▪ <IBusChannel (refer to IBusChannel on page 346)> `busChannelToConnect`: The bus channel.	None
ConnectToCluster	Connects the connector to the given bus cluster. This function always connects the connector to the first channel in the cluster.	▪ <IBusCluster (refer to IBusCluster on page 347)> `busClusterToConnect`: The element of the bus cluster.	None
RemoveConnection	Removes the connection to the bus channel.	None	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IBusConnectors (refer to IBusConnectors on page 350)

IBusConnectors

Description

Specifies the COM interface for a collection of bus controllers.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IBusConnector[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Item	Gets the element with the specified name within the collection.	▪ *<String>* `nameParam`: The name of the element to get.	The element with the specified name within the collection, if found; otherwise, null. ▪ IBusConnector (refer to IBusConnector on page 349)

[1)] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IAdaptiveVEcu (refer to IAdaptiveVEcu on page 341)
- IBusChannel (refer to IBusChannel on page 346)
- IVpu (refer to IVpu on page 368)

ICommunicationPackage

Description

Specifies the COM interface for a communication package.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Name	Gets the name.	Get/Set	*String*
PackageType	Gets the communication package type.	Get	CommunicationPackageType (refer to CommunicationPackageType on page 341)
Type	Gets the communication package protocol type.	Get/Set	CommunicationPackageProtocolType (refer to CommunicationPackageProtocolType on page 340)
VpuPortGroups	Gets the VPU port groups.	Get	IVpuPortGroups (refer to IVpuPortGroups on page 375)

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- ICommunicationPackages (refer to ICommunicationPackages on page 351)

ICommunicationPackages

Description

Specifies the COM interface for a collection of communication packages.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*ICommunicationPackage[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Add	Adds a communication package.	▪ <CommunicationPackageProtocolType (refer to CommunicationPackageProtocolType on page 340)> `protocolType`: The protocol type. ▪ <*String*> `clusterName`: Name of the bus cluster.	None
Add	Adds a communication package with a default name.	▪ <CommunicationPackageProtocolType (refer to CommunicationPackageProtocolType on page 340)> `protocolType`: The protocol type.	None
Item	Gets the element with the specified name within the collection.	▪ <*String*> `nameParam`: The name of the element to get.	The element with the specified name within the collection, if found; otherwise, null. ▪ ICommunicationPackage (refer to ICommunicationPackage on page 351)
Remove	Removes the communication package with the specific name. All VPU port groups that are connected to this communication package will be moved to the standard communication package. Is considered to be thread safe	▪ <*String*> `nameParam`: Name of the communication package.	None

[1)] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

Related topics

Basics

Automating the Connection of VPUs..... 172

IConnectionList

Description

This interface provides all a list of connections.

Properties

The element has no properties.

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
AddConnection	Add VPU port connection.	▪ <*String*> `sourceVpu`: Outgoing VPU. ▪ <*String*> `sourcePortOrigin`: Outgoing port origin. ▪ <*String*> `destVpu`: Incoming VPU. ▪ <*String*> `destPortOrigin`: Incoming port origin.	True, if connection is possible. ▪ *Boolean*
AddVpuConnection	Add VPU port connection.	▪ <IVpuPort (refer to IVpuPort on page 372)> `vpuSourcePort`: The VPU source port. ▪ <IVpuPort (refer to IVpuPort on page 372)> `vpuDestinationPort`: The VPU destination port.	True, if connection is possible. ▪ *Boolean*

[1] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

IConnectionResult

Description

Provides information about connecting vpu ports.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
NumberConnectedVpuPorts	Gets the number of connected vpu ports.	Get	*Signed 32 Bit Integer*
NumberUnConnectedVpuPorts	Gets the number of unconnected vpu ports.	Get	*Signed 32 Bit Integer*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
ConnectedVpuPort	Get connected vpu port.	▪ *<Signed 32 Bit Integer>* `index`: Index of connection.	Return value of the method. ▪ IVpuPortConnection (refer to IVpuPortConnection on page 373)
UnConnectedVpuPort	Get unconnected vpu port.	▪ *<Signed 32 Bit Integer>* `index`: Index of connection.	Return value of the method. ▪ IVpuPortConnection (refer to IVpuPortConnection on page 373)

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

IExportSettings

Description

This interface provides all available options for exports of filesZ.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
ExportSmcFilePath	Gets or sets Full file path to the export SMC file.	Get/Set	*String*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

IImportSettings

Description

This interface provides all available options for imports of the type OSA, FMU or CTLGZ.

Properties The element has the following properties:

Name	Description	Get/Set	Type
AdaptiveVEcuSettings	Gets or sets adaptive VEcu.	Get/Set	IAdaptiveVEcuSettings (refer to IAdaptiveVEcuSettings on page 343)
BoardName	Gets or sets board name.	Get/Set	*String*
BuildDirectoryPath	Gets or sets the path to the working directory of system builder. Default working directory is the OSA file path.	Get/Set	*String*
CodeDescriptorList	Gets or sets Code descriptor list.	Get/Set	*System.String[]*
CompilerName	Gets or sets the name of the compiler that should be used for the import.	Get/Set	*String*
CompilerOptions	Gets or sets Options for the C compiler.	Get/Set	*String*
CppCompilerOptions	Gets or sets Options for the C++ compiler.	Get/Set	*String*
EnableDebugBuild	Gets or sets a value indicating whether enable Debug Build.	Get/Set	*Boolean*
EnableLinkerWarnings	Gets or sets a value indicating whether linker warnings should be enabled.	Get/Set	*Boolean*
EnableLogging	Gets or sets a value indicating whether logging should be enabled.	Get/Set	*Boolean*
ImportConnectedBusses	Gets or sets a value indicating whether the bus settings should be imported too.	Get/Set	*Boolean*
ImportFilePath	Gets or sets Full file path to the import file.	Get/Set	*String*
SerialBaudRate	Gets or sets the serial baud rate.	Get/Set	SerialBaudRate (refer to SerialBaudRate on page 378)
StepSize	Gets or sets step size.	Get/Set	*Double*
VpusToImport	Gets or sets Vpus to import from a osa file.	Get/Set	*System.String[]*
XcpAssignPortsAutomatically	Gets or sets automatic port assigning.	Get/Set	*Boolean*
XcpBufferSize	Gets or sets XCP buffer size.	Get/Set	*Unsigned 32 Bit Integer*
XcpPort	Gets or sets XCP service port.	Get/Set	*Unsigned 16 Bit Integer*
XcpPort1	Gets or sets XCP service port 1.	Get/Set	*Unsigned 16 Bit Integer*
XcpPort2	Gets or sets XCP service port 2.	Get/Set	*Unsigned 16 Bit Integer*
XcpStimulationEnabled	Gets or sets a value indicating whether XCP stimulation should be enabled.	Get/Set	*Boolean*

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
AddCodeDescriptorFile	Adds a code descriptor file to code file collection.	▪ *<String>* `codeDescriptorFile`: The code file.	None
CreateNewAdaptiveVEcuSettings	Creates a new adaptive VEcu settings object.	None	The IAdaptiveVEcuSettings (refer to IAdaptiveVEcuSettings on page 343) that can be used to configure the adaptive VEcu. ▪ IAdaptiveVEcuSettings (refer to IAdaptiveVEcuSettings on page 343)
RemoveCodeDescriptorFile	Removes a code descriptor to code file collection.	▪ *<String>* `codeDescriptorFile`: The code file.	*true* item is successfully removed; otherwise, *false*. This method also returns false if item was not found in the List. ▪ *Boolean*
SetCodeDescriptorList	Sets the code descriptor list.	▪ *<System.String[]>* `codeDescriptorArray`: The code descriptor array.	None
SetVpusToImport	Sets Vpus to import from a osa file.	▪ *<System.String[]>* `vpuArray`: The parameter vpuArray.	None

[1] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

IMessage

Description

Provides all information for a message

Properties

The element has the following properties:

Name	Description	Get/Set	Type
AnyError	Gets a value indicating whether a child message with severity 'Error' or higher exists.	Get	*Boolean*
AnyWarning	Gets a value indicating whether a child message with severity 'Warning' or higher exists.	Get	*Boolean*
Category	Gets the category of the message.	Get	*String*

Name	Description	Get/Set	Type
Children	Gets the direct children of this message. Does not include messages on deeper hierarchy levels.	Get	IMessages (refer to IMessages on page 357)
ErrorMessages	Gets a list of all child messages with severity = Error or higher. Messages on lower hierarchy levels are NOT included.	Get	IMessages (refer to IMessages on page 357)
MessageIdentifier	Gets a unique message identifier as a formatted string, e.g., "Error(100,10,42)".	Get	*String*
Messages	Gets a collection of all child messages. Messages on lower hierarchy levels are NOT included.	Get	IMessages (refer to IMessages on page 357)
MessageText	Gets the localized message text.	Get	*String*
NumberOfChildren	Gets the number of direct child messages. This number does not include messages on lower hierarchy levels.	Get	*Signed 32 Bit Integer*
Parent	Gets the parent element.	Get	*IMessage*
Severity	Gets the severity of the message.	Get	*String*
Time	Gets the time the message was created.	Get	*String*
WarningMessages	Gets a collection of all child messages with severity = Warning or higher. Messages on lower hierarchy levels are NOT included.	Get	IMessages (refer to IMessages on page 357)

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IBuildResult (refer to IBuildResult on page 345)
- IMessages (refer to IMessages on page 357)

IMessages

Description

Manages a list of message elements.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
AnyError	Gets a value indicating whether a child message with severity 'Error' or higher exists.	Get	*Boolean*
AnyWarning	Gets a value indicating whether a child message with severity 'Warning' or higher exists.	Get	*Boolean*
Count	Gets the number of elements currently contained in the Messages collection.	Get	*Signed 32 Bit Integer*

Name	Description	Get/Set	Type
Elements	Gets an array of elements currently contained in the Messages collection.	Get	*IMessage[]*
Names	Gets an array of names of elements currently contained in the Messages collection.	Get	*System.String[]*
Parent	Gets the parent element.	Get	*Object*
RootMessage	Gets the root message.	Get	IMessage (refer to IMessage on page 356)

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
Contains	Determines whether an element is in the Messages collection.	▪ *<String>* `msgId`: The element to locate in the Messages collection.	Returns true if item is found in the Messages collection; otherwise, false. ▪ *Boolean*
IndexOf	Returns the zero-based index of the first occurrence of a specific element in the Messages collection.	▪ *<String>* `msgId`: The element to locate in the Messages collection.	Returns the zero-based index of the first occurrence of the item within the entire Messages collection, if found; otherwise, -1. ▪ *Signed 32 Bit Integer*
Item	Gets the element with the specified name in the Messages collection.	▪ *<String>* `msgId`: The name of the element to get.	Returns the element with the specified name within the Messages collection, if found; otherwise, null. ▪ IMessage (refer to IMessage on page 356)

[1] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IMessage (refer to IMessage on page 356)

IPlayerAutomationInterface

Description Basic interface of the automation facade component.

Properties The element has the following properties:

Name	Description	Get/Set	Type
Application	Gets the application.	Get	IApplication (refer to IApplication on page 343)

Methods The element has no methods.

IPlayerAutomationRegistrationInterface

Description Basic interface of the automation facade component.

Properties The element has no properties.

Methods The element has the following methods:

Name	Description	Parameter[1)]	Returns
ChangeRegistrationOnComAutomationServer	Change Registration of the automation facade.	▪ *<Boolean>* `doRegister`: True if registration should be done. False if deregistration should be done.	True if successful, false otherwise. ▪ *Boolean*

[1)] <Type> `Name`: Description

IProject

Description The project element is the root node of an entire VEOS Player project.

Properties The element has the following properties:

Name	Description	Get/Set	Type
AdaptiveVEcus	Gets a collection of adaptive VEcus referenced in the project.	Get	IAdaptiveVEcus (refer to IAdaptiveVEcus on page 342)
CommunicationClusters	Gets a collection of communication clusters that are used in the simulation system.	Get	IBusClusters (refer to IBusClusters on page 348)
CommunicationPackages	Gets a collection of communication packages that are used in the simulation system.	Get	ICommunicationPackages (refer to ICommunicationPackages on page 351)
Configuration	Gets the simulation configuration of this project.	Get	ISimulationConfiguration (refer to ISimulationConfiguration on page 364)
IsModified	Gets a value indicating whether project was modified.	Get	*Boolean*
Name	Gets or sets the name of the project.	Get/Set	*String*
ProjectAuthor	Gets or sets the author of this project.	Get/Set	*String*
ProjectDate	Gets the creation date of this project.	Get	*String*
ProjectDescription	Gets or sets the description of this project.	Get/Set	*String*
SystemAuthor	Gets the author of the simulation system.	Get	*String*
SystemDescription	Gets the description of the simulation system.	Get	*String*
SystemFile	Gets the simulation application (OSA) file.	Get	*String*
Vpus	Gets a collection of VPUs referenced in the project.	Get	IVpus (refer to IVpus on page 377)

Methods The element has the following methods:

Name	Description	Parameter[1)]	Returns
AdaptiveVEcuItem	Gets the adaptive VEcu item with the specified name.	▪ *<String>* `nameParam`: The name of the adaptive VEcu item to get.	The item with the specified name within the collection, if found; otherwise, null. ▪ IAdaptiveVEcu (refer to IAdaptiveVEcu on page 341)
Close	Closes the project.	▪ *<Boolean>* `saveChanges`: Property is obsolete, will always discard changes.	None
ConnectBusConnectorToChannel	Connects a bus connector to a bus channel.	▪ *<String>* `conApplProcessName`: Name of the application	None

Name	Description	Parameter[1)]	Returns
		process the bus connector belongs to. ▪ *<String>* `busConnectorName`: Name of the bus connector. ▪ *<String>* `busClusterName`: Name of the bus cluster the bus channel belongs to. ▪ *<String>* `busChannelName`: Name of the bus channel.	
ConnectUnconnectedCommunicationControllers	Auto connects communication controller with same name.	None	None
ConnectUnconnectedCommunicationControllersWithSameBaudrate	Auto connects communication controller with same name and baud rate.	None	None
ConnectUnconnectedPortsByName	Connect unconnected signals by name.	None	Number of connected signals. ▪ *Signed 32 Bit Integer*
ConnectUnconnectedSignalsAndCommunicationControllers	Auto connects signals and communication controller with same name.	None	Number of connected signals. ▪ *Signed 32 Bit Integer*
ConnectVpuPortGroupToCommunicationPackage	Connects a VPU port group to a communication package.	▪ *<String>* `applicationProcessName`: Name of the application process the VPU port group belongs to. ▪ *<String>* `vpuPortGroupHierarchy`: Hierarchy of the VPU port group. (Groups are separated by the slash character) ▪ *<String>* `communicationPackageName`: Name of the communication package.	None
ConnectVpuPortList	Connect a list of VPU Ports.	▪ <IConnectionList (refer to IConnectionList on page 353)> `connectionList`: A element list of in- and out ports.	True, if they are connected. ▪ IConnectionResult (refer to IConnectionResult on page 353)
ConnectVpuPorts	Connect VPU Ports.	▪ *<String>* `sourceVpu`: Outgoing VPU. ▪ *<String>* `sourcePortOrigin`: Outgoing port origin. ▪ *<String>* `destVpu`: Incoming VPU. ▪ *<String>* `destPortOrigin`: Incoming port origin.	True, if they are connected. ▪ *Boolean*

Name	Description	Parameter[1)]	Returns
CreateNewConnectionList	Creates a new connection list object.	None	The IConnectionList (refer to IConnectionList on page 353). ▪ IConnectionList (refer to IConnectionList on page 353)
CreateNewExportSettings	Creates a new export settings object.	None	The IExportSettings (refer to IExportSettings on page 354) that can be used to configure the export. ▪ IExportSettings (refer to IExportSettings on page 354)
CreateNewImportSettings	Creates a new import settings object.	None	The IImportSettings (refer to IImportSettings on page 354) that can be used to configure the import. ▪ IImportSettings (refer to IImportSettings on page 354)
Export	Export method, that handles file exports.	▪ <IExportSettings (refer to IExportSettings on page 354)> `settings`: The settings object provides all necessary information about the specific export.	None
Import	Generic Import method, that handles FMU, OSA and CTLGZ imports.	▪ <IImportSettings (refer to IImportSettings on page 354)> `settings`: The settings object provides all necessary information about the specific import.	The IBuildResult (refer to IBuildResult on page 345) of the import. ▪ IBuildResult (refer to IBuildResult on page 345)
ImportNamedVpu	Imports a named VPU.	▪ *<String>* `osaFilePath`: The OSA file path. ▪ *<String>* `vpuName`: The VPU Name. ▪ *<Boolean>* `importConnectedBus`: The import connected bus flag. Is always true	None
ImportVpus	Imports the given VPU OSA file to current OSA.	▪ *<String>* `osaFilePath`: The OSA file path. ▪ *<Boolean>* `importConnectedBuses`: The import connected buses. Is always true	None
Item	Gets the item with the specified name.	▪ *<String>* `nameParam`: The name of the item to get.	The item with the specified name within the collection, if found; otherwise, null. ▪ IVpu (refer to IVpu on page 368)

Name	Description	Parameter[1)]	Returns
RemoveAdaptiveVEcu	Remove the given adaptive VEcu.	▪ *<String>* `adaptiveVEcuVpuName`: The adaptive VEcu name.	None
RemoveVpu	Remove the given VPU from current OSA.	▪ *<String>* `vpuName`: The VPU name.	None
Save	Saves the project.	▪ *<String>* `filename`: The file name.	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProjects (refer to IProjects on page 363)

Related topics

Basics

Automating the Connection of VPUs..... 172
Automating the Import and Removal of VPUs..... 170

IProjects

Description

Manages the list of the Project elements

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Active	Gets the active project.	Get	IProject (refer to IProject on page 359)
Count	Gets the number of elements currently contained in the Projects collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the Projects collection.	Get	*IProject[]*
Names	Gets an array of names of elements currently contained in the Projects collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
CreateNew	Creates new OSA file.	▪ *<String>* `filePath`: The file path to the OSA file.	Returns the element with the specified name within the

Name	Description	Parameter[1)]	Returns
			Projects collection, if found; otherwise, null. ▪ IProject (refer to IProject on page 359)
Item	Gets the element with the specified name within the Projects collection.	▪ *<String>* `nameParam`: The name of the element to get.	Returns the element with the specified name within the Projects collection, if found; otherwise, null. ▪ IProject (refer to IProject on page 359)
Open	Opens the offline simulation application (OSA) file with the given name.	▪ *<String>* `projectFileName`: Specifies the file to open.	Returns the Project element, if the project successful opened, otherwise null. ▪ IProject (refer to IProject on page 359)

[1)] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IApplication (refer to IApplication on page 343)

Related topics

Basics

Automating the Opening and Saving of an Offline Simulation Application..... 169

ISimulationConfiguration

Description

Specifies the COM interface for IConfiguration entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
AccelerationFactor	Gets or sets a factor that should be used to slow the simulation down. A factor of 1.0 emulates a real-time simulation. A factor of 0.0 does not apply any factor at all to the simulation time.	Get/Set	*Double*
BreakAtTime	Gets or sets a time-based breakpoint of the simulation.	Get/Set	*Double*
EnableBusSimulationLog	Gets or sets a value indicating whether Enable Bus Simulation Log.	Get/Set	*Boolean*

Name	Description	Get/Set	Type
EnableSimulationLog	Gets or sets a value indicating whether or not a simulation log file should be written during simulation.	Get/Set	*Boolean*
EnableWaitForDebugOnLoad	Gets or sets a value wait for debug on load is enabled.	Get/Set	*Boolean*
EnableXcpSimulationLog	Gets or sets a value indicating whether or not a XCP simulation log file should be written during simulation.	Get/Set	*Boolean*
LogOutputFolder	Gets the logging output folder path for all logging files.	Get	*String*
Name	Gets the name.	Get	*String*
SimulationTime	Gets or sets the duration of the simulation.	Get/Set	*Double*
SingleStepTime	Gets or sets a value that indicates the amount of time in seconds the simulation time is advanced for each simulation step if a single step simulation is executed. A value of zero indicates that the next simulation step is executed, with no regard to the time between the last and the next simulation step.	Get/Set	*Double*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

Related topics

Basics

Automating the Simulation Configuration..... 226

ISimulator

Description

Specifies access to the simulation control methods of the VEOS Player.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
CurrentSimulationTime	Gets the current simulation time of a running simulation. If the simulation is not running, 0 is returned.	Get	*Double*

Name	Description	Get/Set	Type
ElapsedRealTime	Gets the elapsed real time. If the simulation is not running, 0 is returned.	Get	*Double*
Root	Gets the VEOS Player application element.	Get	IApplication (refer to IApplication on page 343)
SimulatorState	Gets the current state of the simulator kernel server.	Get	*String*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Download	Starts the simulator kernel server on the given endpoint and downloads the simulation. The simulation system can be accessed using XCP now.	▪ <*String*> `endpoint`: Contains endpoint information.(Can be an IP address or host name)	True if succeeded, false otherwise. ▪ *Boolean*
Pause	Pauses the simulation.	None	Returns true, if the simulation pause succeeded; otherwise, false. ▪ *Boolean*
Start	Starts the simulation.	None	Returns true, if the simulation start succeeded; otherwise, false. ▪ *Boolean*
Step	Single-step simulation.	None	Returns true, if the simulation step succeeded; otherwise, false. ▪ *Boolean*
StepAsync	Asynchronous Single-step simulation.	None	Returns true, if the simulation step succeeded; otherwise, false. ▪ *Boolean*
Stop	Stops the simulation.	None	Returns true, if the simulation stop succeeded; otherwise, false. ▪ *Boolean*
Unload	Unloads the simulation if one is running.	None	Returns true, if the unload succeeded; otherwise, false. ▪ *Boolean*

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IApplication (refer to IApplication on page 343)

Related topics

Basics

Automating the Download and Control of an Offline Simulation Application..... 227

ISimulatorEvents

Description

Provides simulator-events for use in Python script.

Properties

The element has no properties.

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
OnSimulationContinued	Event occurs when a simulation was continued.	None	None
OnSimulationPaused	Event occurs when a simulation was paused.	None	None
OnSimulationStarted	Event occurs when a simulation was started.	None	None
OnSimulationStopped	Event occurs when a simulation was stopped.	None	None
OnSimulationSystemDownloaded	Event occurs when a simulation system was downloaded.	None	None
OnSimulationSystemTerminated	Event occurs when a simulation was terminated.	None	None
OnSimulationSystemUnloaded	Event occurs when a simulation system was unloaded.	None	None
OnVpuSettingChanged	Event occurs when a VPU setting was changed.	None	None

1) <Type> `Name`: Description

Related topics

Basics

Automating VEOS Player Event Handling with Python..... 228

IVersionInformation

Description

This class provides all available information about the active version.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
get	Gets the complete VEOS version information.	Get	*String*
SpecialBuild	Gets the VEOS Player special build if available. This property can contain the following values: 1. "P_1" for patch 1 2. "HF_2" for hotfix 1 3. "P_1 HF_1" for patch 1 hotfix 1 4. empty if no special build is available	Get	*String*
Version	Gets the VEOS Player version.	Get	*String*
VersionBuildNumber	Gets the VEOS Player build number version.	Get	*Signed 32 Bit Integer*
VersionMaintenanceNumber	Gets the VEOS Player maintenance number version.	Get	*Signed 32 Bit Integer*
VersionMajorNumber	Gets the VEOS Player major number version.	Get	*Signed 32 Bit Integer*
VersionMinorNumber	Gets the VEOS Player minor number version.	Get	*Signed 32 Bit Integer*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IApplication (refer to IApplication on page 343)

IVpu

Description

Specifies the COM interface for VPU entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Author	Gets the author.	Get	*String*
BuildDate	Gets the build time of the source container as a local string.	Get	*String*
CommunicationControllers	Gets the bus communication controllers to this VPU.	Get	IBusConnectors (refer to IBusConnectors on page 350)
ComPortId	Gets or sets the ID of the serial COM port.	Get/Set	*Unsigned 32 Bit Integer*

Name	Description	Get/Set	Type
ContentInfo	Gets the content entries.	Get	*System.String[]*
CreationDate	Gets the creation time of the source container as a local string.	Get	*String*
Date	Gets the date of creation.	Get	*String*
Description	Gets the description.	Get	*String*
Drift	Gets or sets the VPU timer drift.	Get/Set	*Double*
ExportingTool	Gets the source container creation tool.	Get	*String*
HasCosimulationSupported	Defines whether the ApplicationProcess supports cosimulation.	Get	*Boolean*
HasRealTimeTesting	Gets a value indicating whether real-time testing is enabled.	Get	*Boolean*
ModelCreationDate	Gets the build time as a local string.	Get	*String*
ModelFilePath	Gets the file path of the source container as a local string.	Get	*String*
Name	Gets the name.	Get	*String*
NumberOfCosimulationParticipants	Max. number of cosimulation participants.	Get/Set	*Byte*
Offset	Gets or sets the VPU timer offset.	Get/Set	*Double*
PreCompiledFor	Gets the precompiled for entries.	Get	*System.String[]*
ProcessId	Gets the Process ID of the VPU host process.	Get	*Unsigned 32 Bit Integer*
SerialBaudRate	Gets the baud rate the simulator kernel will communicate with the PIL VPU.	Get	*Unsigned 32 Bit Integer*
SourceType	Gets the source container format information.	Get	*String*
SourceVersion	Gets the source container format version information.	Get	*String*
TargetSimulationName	Gets the target simulation name.	Get	*String*
VpuPortGroups	Gets the list of VPU port groups of this VPU.	Get	IVpuPortGroups (refer to IVpuPortGroups on page 375)
VpuPorts	Gets the list of VPU ports of this VPU.	Get	IVpuPorts (refer to IVpuPorts on page 376)
XcpOnEthPort	Gets the XCP port.	Get	*Unsigned 32 Bit Integer*
XcpOnEthPort1	Gets the XCP port 1.	Get	*Unsigned 32 Bit Integer*
XcpOnEthPort2	Gets the XCP port 2.	Get	*Unsigned 32 Bit Integer*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
AddVpuPortConnections	Add VPU port connections.	▪ *<IVpu>* `otherVPU`: The connect to VPU.	Number of connected signals. ▪ *Signed 32 Bit Integer*
GetVpuPortByOrigin	Gets the VPU port by Origin.	▪ *<String>* `origin`: The origin.	The found VPU port. ▪ IVpuPort (refer to IVpuPort on page 372)

Name	Description	Parameter[1]	Returns
Item	Gets the item with the specified name.	▪ *<String>* `nameParam`: The name of the item to get.	The item with the specified name within the collection, if found; otherwise, null. ▪ IVpuItem (refer to IVpuItem on page 370)
RemoveAllVpuPortConnections	Removes all VPU port connections.	None	None
Replace	Replace method, that handles FMU, CTLGZ, etc. rplaces.	▪ <IImportSettings (refer to IImportSettings on page 354)> `settings`: The settings object provides all necessary information about the specific replace.	The IBuildResult (refer to IBuildResult on page 345) of the replace. ▪ IBuildResult (refer to IBuildResult on page 345)
Update	Update method, that handles FMU, CTLGZ, etc. updates.	None	The IBuildResult (refer to IBuildResult on page 345) of the update. ▪ IBuildResult (refer to IBuildResult on page 345)

[1] <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)
- IVpus (refer to IVpus on page 377)

Related topics

Basics

Automating the Connection of VPUs..... 172
Automating the Simulation Configuration..... 226

IVpuItem

Description

Specifies the COM interface for VPU port entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
DataType	Gets the data type.	Get	*String*
DataWidth	Gets the data width. For scalar ports, this value is always 1.	Get	*Signed 32 Bit Integer*
Description	Gets the DAP signal description.	Get	*String*

Name	Description	Get/Set	Type
Direction	Gets the direction of the VPU port, that is, whether it is a receiving, sending or bidirectional port.	Get	VpuPortDirection (refer to VpuPortDirection on page 379)
InitValues	Gets the initial values of the VPU port variable.	Get/Set	*System.Double[]*
ItemType	Gets the item type.	Get	VpuItemType (refer to VpuItemType on page 379)
Name	Gets the name.	Get	*String*
Origin	Gets the path [DapContext/DapGroup/DapBlock/DapSignal].	Get	*String*
ReceivesFrom	Gets the fully qualified VPU port name of the port that this VPU port is connected to. Only sending ports are displayed this way. If this port is the sending part of a connection, it is displayed at the receiving part.	Get	IVpuPort (refer to IVpuPort on page 372)
SamplingInterval	Gets the sampling interval (in seconds) of the in signal task.	Get	*Double*
SendsTo	Gets a list of fully qualified VPU port names that this VPU port is connected to. Only receiving ports are displayed this way. If this port is the receiving part of a connection, it is displayed at the sending part.	Get	IVpuPorts (refer to IVpuPorts on page 376)
Unit	Gets the DAP unit.	Get	*String*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
AddConnection	Add item connection.	▪ *<IVpuItem>* `vpuItem`: The item.	Number of connected signals. ▪ *Signed 32 Bit Integer*
Item	Gets the item with the specified name.	▪ *<String>* `nameParam`: The name of the item to get.	The item with the specified name within the collection, if found; otherwise, null. ▪ *IVpuItem*
RemoveAllConnections	Removes all connections recursive.	None	None
RemoveConnection	Remove item connection.	▪ *<IVpuItem>* `vpuItem`: The port.	None
SetInitValues	Sets the initial values of the VPU port variable.	▪ *<System.Double[]>* `values`: Initial values.	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IVpu (refer to IVpu on page 368)
- IVpuPortGroup (refer to IVpuPortGroup on page 374)

Related topics

Basics

Automating the Connection of VPUs..... 172

IVpuPort

Description

Specifies the COM interface for VPU port entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
DataType	Gets the data type.	Get	*String*
DataWidth	Gets the data width. For scalar ports, this value is always 1.	Get	*Signed 32 Bit Integer*
Description	Gets the DAP signal description.	Get	*String*
Direction	Gets the direction of the VPU port, that is, whether it is a receiving, sending or bidirectional port.	Get	VpuPortDirection (refer to VpuPortDirection on page 379)
InitValues	Gets the initial values of the VPU port variable.	Get/Set	*System.Double[]*
InitValuesInt64	Gets/Sets the initial values of the VPU port Int64 variable.	Get/Set	*System.Int64[]*
InitValuesUInt64	Gets/Sets the initial values of the VPU port UInt64 variable.	Get/Set	*System.UInt64[]*
MimeType	Gets the mime type.	Get	*String*
Name	Gets the name.	Get	*String*
Origin	Gets the path [DapContext/DapGroup/DapBlock/DapSignal].	Get	*String*
ReceivesFrom	Gets the fully VPU port of the port that this VPU port is connected to. Only sending ports are displayed this way. If this port is the sending part of a connection, it is displayed at the receiving part.	Get	*IVpuPort*
SamplingInterval	Gets the sampling interval (in seconds) of the in signal task.	Get	*Double*
SendsTo	Gets a list of VPU ports that this VPU port is connected to. Only receiving ports are displayed this way. If this port is the receiving part of a connection, it is displayed at the sending part.	Get	IVpuPorts (refer to IVpuPorts on page 376)
Unit	Gets the DAP unit.	Get	*String*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
AddVpuPortConnection	Add VPU port connection.	▪ *<IVpuPort>* `vpuPort`: The VPU port.	True, if they are connected. ▪ *Boolean*
RemoveVpuPortConnection	Removes a VPU port connection.	▪ *<IVpuPort>* `vpuPort`: The VPU port.	None
SetInitValues	Sets the initial values of the VPU port variable.	▪ *<System.Double[]>* `values`: Initial values.	None
SetInitValuesInt64	Sets the initial values of the VPU port Int64 variable.	▪ *<System.Int64[]>* `initialValuesInt64`: Initial Int64 values.	None
SetInitValuesUInt64	Sets the initial values of the VPU port Int64 variable.	▪ *<System.UInt64[]>* `initialValuesUInt64`: Initial UInt64 values.	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IVpu (refer to IVpu on page 368)
- IVpuItem (refer to IVpuItem on page 370)
- IVpuPorts (refer to IVpuPorts on page 376)

Related topics

Basics

Automating the Connection of VPUs..... 172

IVpuPortConnection

Description

This interface provides a vpu port connection.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
DestPortOrigin	Gets the incoming port origin.	Get	*String*
DestVpu	Gets the incoming Vpu.	Get	*String*
SourcePortOrigin	Gets the outgoing port origin.	Get	*String*
SourceVpu	Gets the outgoing Vpu.	Get	*String*

Methods

The element has no methods.

Returned by

The element is returned by properties or methods of the following elements:

- IConnectionResult (refer to IConnectionResult on page 353)

IVpuPortGroup

Description

Specifies the COM interface for VPU port group entities.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Name	Gets the name.	Get	*String*
VpuPortGroups	Gets the list of VPU ports of this port group.	Get	IVpuPortGroups (refer to IVpuPortGroups on page 375)
VpuPorts	Gets the list of VPU ports of this port group.	Get	IVpuPorts (refer to IVpuPorts on page 376)

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
AddVpuPortGroupConnection	Connects all ports under this group with the given group, if possible. To connect all ports following requirements must be valid. 1. both groups have same structure (same port count and hierarchy count) 2. the incoming port group should not contain an already connected port.	▪ *<IVpuPortGroup>* `vpuPortGroup`: The VPU port group.	Number of connected signals. ▪ *Signed 32 Bit Integer*
Item	Gets the item with the specified name.	▪ *<String>* `nameParam`: The name of the item to get.	The item with the specified name within the collection, if found; otherwise, null. ▪ IVpuItem (refer to IVpuItem on page 370)
MoveToCommunicationPackage	Moves this VPU port group to the given communication package if allowed.	▪ <ICommunicationPackage (refer to ICommunicationPackage on page 351)> `communicationPackage`:	None

Name	Description	Parameter[1)]	Returns
		The communication package.	
RemoveAllVpuPortConnections	Removes all VPU port connections of this group recursive.	None	None

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IVpuPortGroups (refer to IVpuPortGroups on page 375)

Related topics

Basics

Automating the Connection of VPUs..... 172

IVpuPortGroups

Description

Specifies the COM interface for a collection of VPU ports.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IVpuPortGroup[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Item	Gets the element with the specified name within the collection.	▪ *<String>* `nameParam`: The name of the element to get.	The element with the specified name within the collection, if found; otherwise, null. ▪ IVpuPortGroup (refer to IVpuPortGroup on page 374)

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- ICommunicationPackage (refer to ICommunicationPackage on page 351)
- IVpu (refer to IVpu on page 368)
- IVpuPortGroup (refer to IVpuPortGroup on page 374)

Related topics

Basics

Automating the Connection of VPUs..... 172

IVpuPorts

Description

Specifies the COM interface for a collection of VPU ports.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IVpuPort[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Item	Gets the element with the specified name within the collection.	▪ *<String>* `nameParam`: The name of the element to get.	The element with the specified name within the collection, if found; otherwise, null. ▪ IVpuPort (refer to IVpuPort on page 372)

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IVpu (refer to IVpu on page 368)
- IVpuItem (refer to IVpuItem on page 370)

- IVpuPort (refer to IVpuPort on page 372)
- IVpuPortGroup (refer to IVpuPortGroup on page 374)

Related topics

Basics

Automating the Connection of VPUs..... 172

IVpus

Description

Specifies the COM interface for a collection of VPUs.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
Count	Gets the number of elements currently contained in the collection.	Get	*Signed 32 Bit Integer*
Elements	Gets an array of elements currently contained in the collection.	Get	*IVpu[]*
Names	Gets an array of names of elements currently contained in the collection.	Get	*System.String[]*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
Item	Gets the element with the specified name within the collection.	▪ *<String>* `nameParam`: The name of the element to get.	Returns the element with the specified name within the collection, if found; otherwise, null. ▪ IVpu (refer to IVpu on page 368)

1) <Type> `Name`: Description

Returned by

The element is returned by properties or methods of the following elements:

- IProject (refer to IProject on page 359)

SerialBaudRate

Description

Enumeration for all provided serial baud rates that can be used while source import.

Enumeration values

The enumeration has the following values:

Name	Description	Value
Nothing	Value for not defined. (default)	0
B9600	The b 9600.	9600
B19200	The b 19200.	19200
B38400	The b 38400.	38400
B57600	The b 57600.	57600
B115200	The b 115200.	115200

Returned by

The element is returned by properties or methods of the following elements:

- IImportSettings (refer to IImportSettings on page 354)

SimulationProgress

Description

Simulation Progress enumeration.

Enumeration values

The enumeration has the following values:

Name	Description	Value
OsaDownloaded	The OSA file is downloaded.	0
SimulationStarted	The simulation has started.	1
SimulationResumedFromPause	The simulation resumed from pause.	2
SimulationCompleted	The simulation is finished.	3
SimulationPaused	The simulation is paused.	4
SimulationTerminated	Simulation terminated.	5

VpuItemType

Description

The VPU item types enumeration.

Enumeration values

The enumeration has the following values:

Name	Description	Value
Port	The VPU port type.	0
Group	The VPU group type.	1

Returned by

The element is returned by properties or methods of the following elements:

- IVpuItem (refer to IVpuItem on page 370)

Related topics

Basics

Automating the Connection of VPUs..... 172

VpuPortDirection

Description

The VPU port direction enumeration.

Enumeration values

The enumeration has the following values:

Name	Description	Value
In	The in direction.	0
Out	The out direction.	1
NA	The direction is not available.	2

Returned by

The element is returned by properties or methods of the following elements:

- IVpuItem (refer to IVpuItem on page 370)
- IVpuPort (refer to IVpuPort on page 372)

Limitations

Limitations for Automating the VEOS Player

Limitation for automating the VEOS Player

All object references such as VPUs or VPU ports are invalid after you import a file of one of the following file types to the VEOS Player:

- BSC (bus simulation container)
- FMU (Functional mock-up unit)
- SIC (Simulink implementation container)
- SMC (system model container)
- VECU (V-ECU implementation)

Related topics

Basics

Automating the VEOS Player..... 325

User Interface Handling

Where to go from here

Information in this section

Basics and Instructions.....382
Reference Information.....388
Automation.....414

Basics and Instructions

Where to go from here

Information in this section

User Interface of the VEOS Player.....382
Provides an overview of the VEOS Player's graphical user interface.

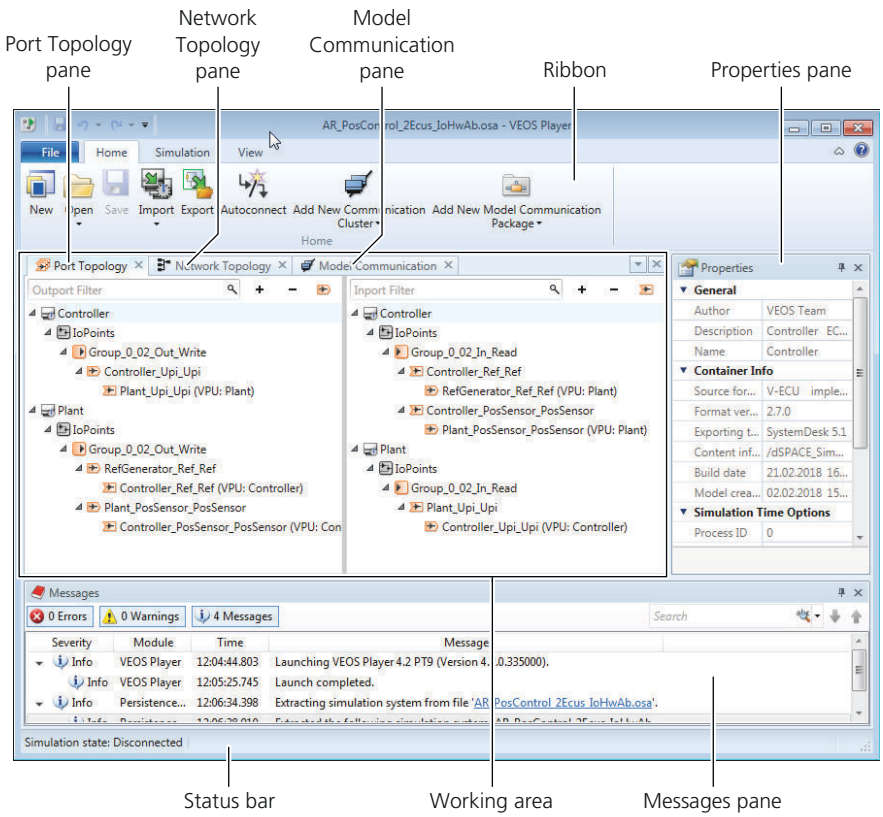
Basics on Ribbons.....384
Ribbons are user interface elements that provide access to common commands and dialogs.

How to Customize the Screen Arrangement.....386
Instructions on creating a custom pane arrangement.

User Interface of the VEOS Player

Overview

The following illustration shows the user interface of the VEOS Player.



Panes

Messages pane The Messages pane provides a history of all the info, advice, error and warning messages, and all the questions that occur when you work with the product. This helps you check the system state.

For reference information, refer to Messages Pane on page 401.

Model Communication pane The Model Communication pane displays the communication packages of the simulation system.

For reference information, refer to Model Communication Pane on page 116.

Network Topology pane The Network Topology pane displays the bus simulation configuration of the simulation system.

For reference information, refer to Network Topology Pane on page 118.

Port Topology pane The Port Topology pane displays the VPUs and signals of the simulation system.

For reference information, refer to Port Topology Pane on page 120.

Properties pane Via the Properties pane, you can view and change the properties of the element currently selected in one of the panes.

For reference information, refer to Elements and Properties on page 122.

Ribbon

Via the ribbon, you can quickly access the VEOS Player commands to edit and simulate the configuration.

For information on using the ribbon, refer to Basics on Ribbons on page 384.

Status bar

Displays the following information from left to right (if available):

- Simulation state (disconnect, running, paused, stopped)
- Simulation time in seconds
- Elapsed real time in seconds

Related topics

Basics

Basics on Ribbons..... 384

References

Commands and Dialogs..... 134
Elements and Properties..... 122
Messages Pane..... 401
Network Topology Pane..... 118
Port Topology Pane..... 120
Status Bar..... 411

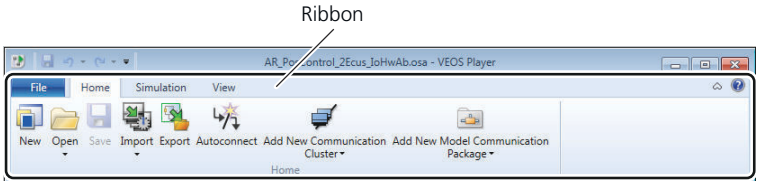
Basics on Ribbons

Introduction

Ribbons are user interface elements that provide access to common commands and dialogs.

Ribbon

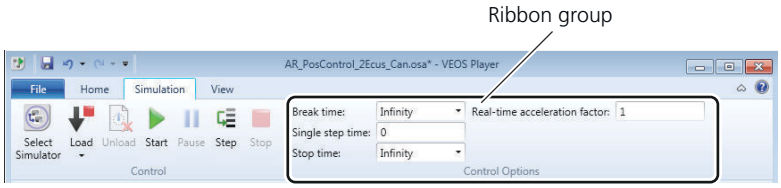
The ribbon organizes and groups commands of a program. The ribbon is located at the top of the user interface, see the following example.



Ribbon group

A ribbon group is a part of a tabbed ribbon. It consists of a set of related commands.

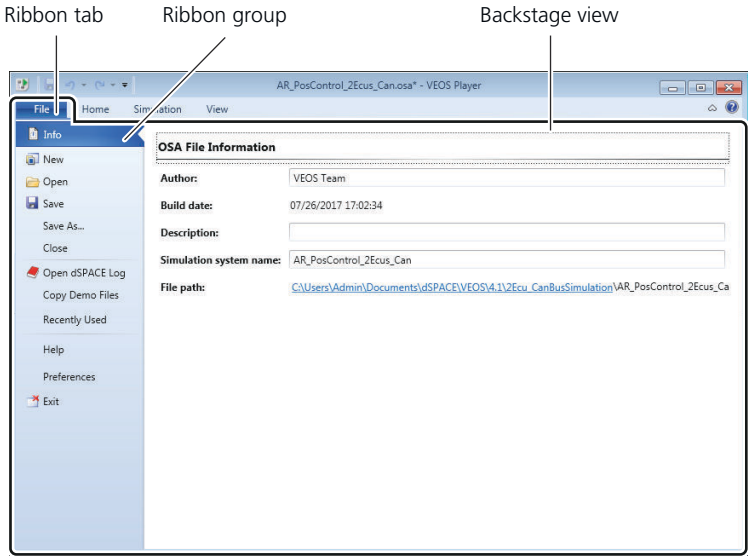
The following illustration shows a ribbon group in the VEOS Player as an example.



Backstage view

The Backstage view provides basic commands of a software program, such as Save, Open, Close, Exit. The Backstage view can be identified by the colored ribbon tab. When the Backstage view is opened, it covers the entire user interface.

The following illustration shows the Backstage view of the Info ribbon group as an example.



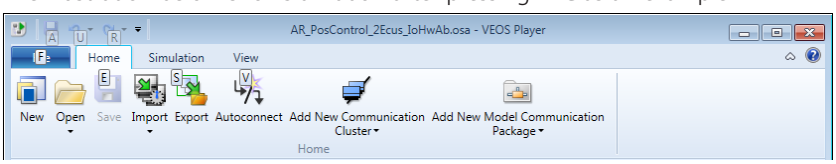
Navigating the ribbon

You can navigate the ribbon via mouse and via keyboard.

Navigation via mouse You can navigate the ribbon with the mouse scroll wheel.

Navigation via keyboard If you want to navigate the ribbon via keyboard, press the **`Alt`** key. Each command in the Quick Access Toolbar and each ribbon tab then is marked by an access key.

The illustration below shows a ribbon after pressing **`Alt`** as an example.



If you then press one of the ribbon tab access keys, each command in the selected ribbon is also marked by an access key.

You can remove the access keys by pressing **`Alt`** again.

Related topics

Basics

User Interface of the VEOS Player..... 382

How to Customize the Screen Arrangement

Objective

The screen arrangement defines which panes are displayed and how they are arranged. The first time you execute the application, it starts with a default screen arrangement that you can modify.

Screen modifications

The screen arrangement contains the following information:

- Display states and positions of the panes such as the **Properties** pane.
- Pane settings, such as the docking state.

Resetting the screen arrangement

To discard all the modifications that you made in the screen arrangement, you can reset it to its default by using the **Reset View Set command** (refer to Reset View Set on page 410) command.

Commands for customizing the screen arrangement

The application provides various commands, such as the **Floating** command, to modify the state of each pane. You can execute all of these commands quickly and flexibly using the mouse. The following instructions describe how to do this.

Method

To customize the screen arrangement

1 Move the mouse pointer to the title bar of the pane whose position you want to change.

If you want to move a tabbed pane, you must select its tab instead of the title bar.

2 Drag the pane to another position while holding the left mouse button down.

The docking state of the pane is automatically changed to *floating* and the screen displays *docking stickers* that you can use to specify the new position.

Docking Sticker	Description
	The pane is docked to the top of your application's main window.
	The pane is docked to the bottom of your application's main window.
	The pane is docked to the left of your application's main window.
	The pane is docked to the right of your application's main window.
	The pane is docked above, below, to the left, or to the right of the selected pane. If you drag the mouse onto the middle docking sticker, the pane is docked as a new tab.

3 Move the mouse pointer onto a docking sticker. When the area of the new position is displayed, release the left mouse button.

Result

The pane is moved to the new position in the user interface and docked to another pane.

If you release the mouse button in an area outside a docking sticker, the docking state of the pane remains floating.

Tip

If you want to change the order of pane tabs, you can drag them to new positions.

Related topics

References

Close All But This 392
Dock 393
Float 398
Move to Next Tab Group 404
Move to Previous Tab Group 404
New Horizontal Tab Group 405
New Vertical Tab Group 406
Reset View Set 410
Switch Panes 411

Reference Information

Where to go from here

Information in this section

About VEOS Player.....390
To display information on the VEOS Player version currently installed on your system.

Add to Quick Access Toolbar.....390
To add the selected command to the Quick Access Toolbar.

Auto Hide.....391
To hide the pane automatically if it is not focused.

Clear Filter.....391
To clear a filter that is set for a column.

Close All But This.....392
To close all panes except for the selected one in the working area.

Copy Demo Files.....392
To copy and unpack the demo files for the VEOS Player to the user-specific Documents folder.

Dock.....393
To place a floating pane at a certain location in the user interface and connect it to its neighboring panes.

dSPACE Help.....394
To open the user documentation of VEOS.

dSPACE Log Viewer.....394
To display the dSPACE Log.

Exit.....396
To exit the current VEOS Player session.

Filter Panel.....397
To filter the message list and search for text in the message list.

Float.....398
To make a pane movable.

Help (Backstage View).....399
To provide access to help commands.

Hide.....400
To hide the currently selected pane.

Keyboard Help.....400
To display a list of shortcut keys.

Message Dialog.....401
To display a message of VEOS.

Messages Pane.....401
To provide a history of info, error and warning messages, and questions.

Minimize the Ribbon.....403
To toggle between a minimized and expanded ribbon.

Move to Next Tab Group.....404
To move a pane to the tab group right or below the pane.

Move to Previous Tab Group.....404
To move a pane to the previous tab group.

New Features and Migration.....405
To display new features and required migration steps for all the products in the current dSPACE Release.

New Horizontal Tab Group.....405
To create another horizontal tab.

New Vertical Tab Group.....406
To create another vertical tab group.

Open dSPACE Log.....407
To open the dSPACE Log.

PDF Files.....407
To open a folder containing documentation PDF files of the current dSPACE Release.

Preferences.....408
To configure global settings of the VEOS Player.

Redo.....408
To redo the most recent commands or actions.

Remove from Quick Access Toolbar.....409
To remove the selected command from the Quick Access Toolbar.

Reset View Set.....410
To reset the view set to its default state.

Show Quick Access Toolbar Above / Below the Ribbon.....410
To show the Quick Access Toolbar above or below the ribbon.

Status Bar.....411
To show or hide VEOS Player's status bar at the bottom of the main window.

Switch Panes.....411
To show or hide panes.

Undo.....412
To undo the most recent edit commands or actions.

Using dSPACE Help.....413
To get information on working with dSPACE Help.

About VEOS Player

Access

You can access this command via:

Ribbon	File – Help
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To display information on the VEOS Player version currently installed on your system.

Related topics

Basics

Compatibility of VEOS 40

Add to Quick Access Toolbar

Access

You can access this command via:

Ribbon	None
Context menu of	Command in a ribbon
Shortcut key	None
Icon	None

Purpose

To add the selected command to the Quick Access Toolbar.

Result

The command is added to the Quick Access Toolbar.

Related topics

References

Remove from Quick Access Toolbar 409

Auto Hide

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in docked mode
Shortcut key	None
Icon	None
Others	in the pane's title bar

Purpose

To hide the pane automatically if it is not focused.

Description

The Auto Hide command is enabled only for panes in docked mode. If you enable the Auto Hide feature, a tab representing the pane is created in the user interface. You can click the tab to display the pane again. The position of the tab depends on the original position of the pane you want to hide. The Auto Hide symbol in the title bar changes to .

Related topics

References

Dock..... 393

Clear Filter

Access

You can access this command via:

Ribbon	None
Context menu of	Column header in: ▪ Model Communication pane ▪ Select Simulator dialog ▪ Select Simulator
Shortcut key	None
Icon	

Purpose

To clear a filter that is set for a column.

Related topics

References

Filter Editor 147
Model Communication Pane 116
Select Simulator 219

Close All But This

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in the working area
Shortcut key	None
Icon	None

Purpose

To close all panes except for the selected one in the working area.

Result

All panes except for the selected one in the working area are closed.

Related topics

HowTos

How to Customize the Screen Arrangement 386

Copy Demo Files

Access

You can access this command via:

Ribbon	File
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To copy and unpack the demo files for the VEOS Player to the user-specific Documents folder.

Result

Demo files for the VEOS Player are copied and unpacked to the user-specific *Documents folder*.

Description

Demo files are copied to the *Documents folder*.

Documents folder A standard folder for user-specific documents.

```
%USERPROFILE%\Documents\dSPACE\<ProductName>\
<VersionNumber>
```

Related topics

Basics

Demos for VEOS 37

Dock

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in floating mode
Shortcut key	None
Icon	None

Purpose

To place a floating pane at a specific area in the user interface and connect it to its neighboring panes.

Description

The Dock command from the title bar's context menu is available for a pane that is in the floating state.

If you drag a pane and drop it on the selected positioning symbol, it is automatically docked.

Related topics

HowTos

How to Customize the Screen Arrangement 386

References

Float 398

dSPACE Help

Access

You can access this command via:

Ribbon	File – Help
Context menu of	None
Shortcut key	None
Icon	

Purpose

To open the user documentation of VEOS.

Related topics

References

Help (Backstage View)..... 399

dSPACE Log Viewer

Access

You can access the viewer via:

Ribbon	▪ File - Open dSPACE Log ▪ View - Open dSPACE Log
Context menu of	None
Shortcut key	None
Icon	

Purpose

To display the dSPACE Log.

Result

The dSPACE Log Viewer opens. It displays the dSPACE Log.

Description

The dSPACE Log is a collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

Filter panel

The filter panel lets you filter the message list and search for text in the message list.

Refer to Filter Panel on page 397.

Message list

The message list provides the following information for each message:

Information	Description
Date[1)]	The date the message was issued.
Main Module Number[1)]	The main module that issued the message.
Message	The content of the message.
Message Code[1)]	The code of the message.
Message Number[1)]	The number of the message.
Module[1)]	The module that issued the message.
Process ID[1)]	The ID of the process in which the message was issued.
Product[1)]	The name of the product that issued the message.
Severity[1)]	The severity level of the message, indicated by one of the following symbols: ▪ errors ▪ warnings ▪ other messages, i.e., infos, advice, and questions
Submodule Number[1)]	The submodule that issued the message.
Thread ID[1)]	The ID of the thread in which the message was issued.
Time[1)]	The time the message was issued.

[1)] You can specify whether this information is displayed via the Show Columns command.

Show/Hide Messages of Log Session (Available in the column header) Lets you expand/collapse the messages in the message list according to log sessions. A log session starts when a dSPACE product is started.

Fit Column Width (Available from the context menu of column headers) Lets you optimize the width of the selected column.

Fit All Columns (Available from the context menu of column headers) Lets you optimize the widths of all columns to fit the width of the list.

Show Columns (Available from the context menu of column headers) Lets you add/remove a column to/from the message list.

Reset Columns Lets you reset the display of message list columns to the default.

Copy (Available from the context menu of messages) Lets you copy the selected messages to the Clipboard.

Show Message (Available from the context menu of messages) Lets you display the selected message in a separate dialog.

Show Filter Panel (Available from the context menu of messages) Lets you show/hide the filter panel.

Refer to Filter Panel on page 397.

Show Details (Available from the context menu of messages) Lets you display/hide all the details of the currently selected message.

Lock Scrolling (Available from the context menu of messages) Lets you disable the automatic horizontal scrolling mechanism in the viewer.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

References

Open dSPACE Log..... 407

Exit

Access

You can access this command via:

Ribbon	File
Context menu of	None
Shortcut key	`Alt` + `F4`
Icon	

Purpose

To exit the current VEOS Player session.

Description

You are prompted in the following cases when you exit:

- If you made any changes to the current offline simulation application, you are prompted to save them before exiting the session.

- If an offline simulation application is running, you are prompted to unload the simulation system before exiting. If you click **No**, the simulation system remains loaded and VEOS Player is closed.

Related topics

References

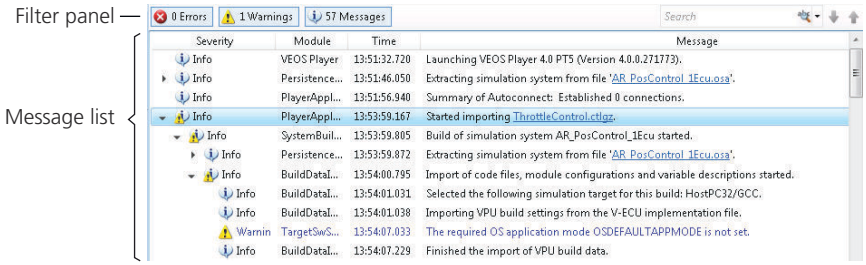
Save..... 161

Filter Panel

Access

- The filter panel is part of the **Messages** pane. Refer to Messages Pane on page 401.

  The following illustration shows the **Messages** pane with the filter panel as an example:



- The filter panel is part of the **dSPACE Log Viewer**. Refer to dSPACE Log Viewer on page 394.

Purpose

To filter the message list and search for text in the message list.

Filter panel

The filter panel lets you filter the message list and search for text in the message list.

Edit Product Filter (button - only in dSPACE Log) Lets you specify a list of products for filtering the message list.

To apply the product filter, click the Enable/Disable Product Filter button next to the button.

Enable/Disable Product Filter (Products button - only in dSPACE Log) Lets you enable/disable the product filter.

Show/Hide Errors (Errors button) Lets you display or hide errors.

Show/Hide Warnings (Warnings button) Lets you display or hide warnings.

Show/Hide other Messages (Messages button) Lets you display or hide other messages, i.e., infos, advice, and questions.

Search Lets you enter a text string for searching the message list.

You can use the following wildcards in the text string:

- **?** (wildcard for one character)
- ***** (wildcard for any number of characters)

To mask a wildcard, enter the \ character before the wildcard.

To select the next occurrence of the search string, click next to the edit field, or press the **Enter** key while the search field has the focus.

To select the previous occurrence of the search string, click next to the edit field.

Related topics

References

dSPACE Log Viewer..... 394
Messages Pane..... 401

Float

Access

You can access this command via:

Menu bar	None
Context menu of	Pane's title bar in docked mode
Shortcut key	None
Icon	None

Purpose

To make a pane movable.

Description

The Float command from the title bar's context menu is available for a component that is in docked state.

If you drag a docked pane but do not drop it on a positioning symbol, it is automatically in the floating state.

Note

The command makes a pane floating, but you can move it only within the main window.

Related topics

HowTos

How to Customize the Screen Arrangement 386

References

Dock 393

Help (Backstage View)

Access

You can access this ribbon group via:

Ribbon	File
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To provide access to help commands.

Description

You have access to the following commands:

- dSPACE Help on page 394
- Using dSPACE Help on page 413
- New Features and Migration on page 405
- PDF Files on page 407
- Keyboard Help on page 400

Related topics

References

dSPACE Help 394
Keyboard Help 400

New Features and Migration 405
PDF Files 407
Using dSPACE Help 413

Hide

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in docked mode
Shortcut key	None
Icon	None
Others	None

Purpose

To hide the currently selected pane.

Related topics

References

Dock 393

Keyboard Help

Access

You can access this command via:

Ribbon	File – Help
Context menu of	None
Shortcut key	None
Icon	

Purpose

To display a list of shortcut keys.

Result

VEOS Player opens the Active Keyboard Shortcuts dialog that lists the shortcut keys.

Active Keyboard Shortcuts dialog

Displays a list of shortcut keys.

Related topics

References

Help (Backstage View)..... 399

Message Dialog

Purpose

To display a message of VEOS.

Description

Copy (Available from the context menu of a message dialog) Lets you copy the selected text from the message to the Clipboard.

Select all (Available from the context menu of a message dialog) Lets you select the entire message text.

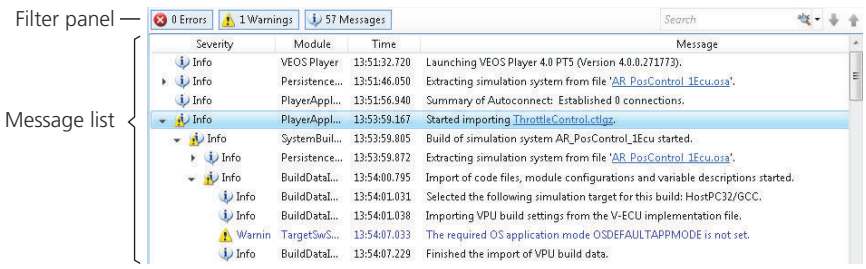
Copy Message (Available from the context menu of a message dialog) Lets you copy the entire message and its attributes to the Clipboard.

Messages Pane

Description

The **Messages** pane provides a history of all the info, advice, error and warning messages, and all the questions that occur when you work with the product. This helps you check the system state.

The **Messages** pane looks like this:



Filter panel

The filter panel lets you filter the message list and search for text in the message list.

Refer to Filter Panel on page 397.

Message list

The message list provides the following information for each message:

Information	Description
Date[1)]	The date the message was issued.
Main Module Number[1)] (Main Module)	The main module that issued the message.
Message	The content of the message.
Message Code[1)]	The code of the message.
Module[1)]	The module that issued the message.
Severity	The severity level of the message, indicated by one of the following symbols: ▪ Errors ▪ Warnings ▪ Other messages, i.e., infos, advice, and questions
Submodule Number[1)] (Submodule)	The submodule that issued the message.
Time[1)]	The time the message was issued.

1) You can specify whether this information is displayed via the Show Columns command.

Fit Column Width (Available from the context menu of column headers) Lets you optimize the width of the selected column.

Fit All Columns (Available from the context menu of column headers) Lets you optimize the widths of all columns to fit the width of the list.

Show Columns (Available from the context menu of column headers) Lets you add/remove a column to/from the message list.

Reset Columns Lets you reset the display of message list columns to the default.

Copy (Available from the context menu of messages) Lets you copy the selected messages to the Clipboard.

Help (Available from the context menu of messages) Lets you display message-specific help.

Expand (Available from the context menu of messages; available only if the tree view of messages is active) Lets you expand all the collapsed subelements of the selected element.

The hidden subelements of the selected element are displayed.

Collapse (Available from the context menu of messages; available only if the tree view of messages is active) Lets you collapse all the subelements of the selected element.

The subelements of the selected element are hidden.

Show Message (Available from the context menu of messages) Lets you display the selected message in a separate dialog.

Tree View (Available from the context menu of messages) Lets you toggle between flat and tree views of the messages.

Show Filter Panel (Available from the context menu of messages) Lets you show/hide the filter panel.

Refer to Filter Panel on page 397.

Clear Messages (Available from the context menu of messages) Lets you clear all the messages in the list.

Lock Scrolling (Available from the context menu of messages) Lets you disable the automatic horizontal scrolling mechanism in the viewer.

Open File (Available from the context menu of messages containing a linked file) Lets you open the linked file.

Open Containing Folder (Available from the context menu of messages containing a linked file) Lets you browse the folder of the linked file.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432
User Interface of the VEOS Player..... 35

Minimize the Ribbon

Access

You can access this command via:

Ribbon	None
Context menu of	Commands in the ribbon
Shortcut key	None
Icon	(on the right of the ribbon) or

Purpose

To toggle between a minimized and expanded ribbon.

Result

If the ribbon is minimized, only the ribbon tabs are shown on the user interface.

Description

To access a command when the ribbon is minimized, click a ribbon tab. The commands remain on the user interface until you execute a command from the ribbon.

Move to Next Tab Group

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in the working area
Shortcut key	None
Icon	None
Others	None

Purpose

To move a pane to the tab group right or below the pane.

Description

You can move a pane to the next tab group, if there is a tab group below or right of this pane`s tab group.

Related topics

HowTos

How to Customize the Screen Arrangement..... 386

Move to Previous Tab Group

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in the working area
Shortcut key	None
Icon	None
Others	None

Purpose

To move a pane to the previous tab group.

Description

You can move a pane to the previous tab group if there are more than one tab groups in the working area.

Related topics

HowTos

How to Customize the Screen Arrangement 386

New Features and Migration

Access

You can access this command via:

Ribbon	File – Help
Context menu of	None
Shortcut key	None
Icon	

Purpose

To display new features and required migration steps for all the products in the current dSPACE Release.

Result

dSPACE Help opens with New Features and Migration 🕮 displayed. Navigate to the specific product information to read about the new features of a specific product. If there are migration steps required, the necessary steps are described.

New Horizontal Tab Group

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in docked mode in the top tab group
Shortcut key	None
Icon	None
Others	None

Purpose

To create another horizontal tab.

Description

A new horizontal tab group is created below the top tab group in the working area.

Related topics

HowTos

How to Customize the Screen Arrangement..... 386

New Vertical Tab Group

Access

You can access this command via:

Ribbon	None
Context menu of	Pane's title bar in docked mode in the leftmost tab group
Shortcut key	None
Icon	None
Others	None

Purpose

To create another horizontal tab group.

Description

A new horizontal tab group is created on the right of the leftmost tab group in the working area.

Related topics

HowTos

How to Customize the Screen Arrangement..... 386

Open dSPACE Log

Access

You can access this command via:

Ribbon	▪ File ▪ View - Show
Context menu of	None
Shortcut key	None
Icon	

Purpose

To open the dSPACE Log.

Result

The dSPACE Log is opened.

Refer to dSPACE Log Viewer on page 394.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

References

dSPACE Log Viewer..... 394

PDF Files

Access

You can access this command via:

Ribbon	File – Help
Context menu of	None
Shortcut key	None
Icon	

Purpose

To open a folder containing documentation PDF files of the current dSPACE Release.

Preferences

Access

You can access this command via:

Ribbon	File
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To configure global settings of the VEOS Player.

Description

Log Options You have access to the following settings:

- Generate Bus Log Files on page 214
- Generate Simulation Log Files on page 215
- Generate XCP Log Files on page 215

General Options You have access to the following settings:

- Wait for Debugger on Load on page 224
- Export the Variable Description Files (A2L or TRC) when the OSA File is Saved on page 146
- Open Simulation Target Manager (refer to Simulation Target Manager on page 163)

Related topics

References

Export the Variable Description Files (A2L or TRC) when the OSA File is Saved..... 146
Generate Bus Log Files..... 214
Generate Simulation Log Files..... 215
Generate XCP Log Files..... 215
Simulation Target Manager..... 163
Wait for Debugger on Load..... 224

Redo

Access

You can access this command via:

Ribbon	None
Context menu of	None
Shortcut key	`Ctrl` + `Y`

Icon	
Others	Quick Access Toolbar

Purpose

To redo the most recent commands or actions.

Description

If you carry the command out once, one command or action is redone; if you carry it out twice, two are redone, and so on.

Tip

You can click the down arrow to get a list of the actions that can be redone. If you select one of the actions from the list, all actions up to this action are redone.

Related topics

References

Undo..... 412

Remove from Quick Access Toolbar

Access

You can access this command via:

Ribbon	None
Context menu of	Quick Access Toolbar command
Shortcut key	None
Icon	None

Purpose

To remove the selected command from the Quick Access Toolbar.

Result

The command is removed from the Quick Access Toolbar.

Related topics

References

Add to Quick Access Toolbar..... 390

Reset View Set

Access

You can access this command via:

Ribbon	View – View Set
Context menu of	None
Shortcut key	None
Icon	

Purpose

To reset the view set to its default state, i.e., the state of the first installation.

Related topics

HowTos

How to Customize the Screen Arrangement 386

Show Quick Access Toolbar Above / Below the Ribbon

Access

You can access this command via:

Ribbon	None
Context menu of	▪ Commands in the ribbon ▪ Commands in the Quick Access Toolbar
Shortcut key	None
Icon	None

Purpose

To show the Quick Access Toolbar above or below the ribbon.

Result

The Quick Access Toolbar is shown above or below the ribbon.

Related topics

Basics

Basics on Ribbons 384

Status Bar

Access

You can access this command via:

Ribbon	View – Show
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To show or hide VEOS Player's status bar at the bottom of the main window.

Description

The status bar displays the following information from left to right (if available):

- Simulation state (disconnect, running, paused, stopped)
- Simulation time in seconds
- Elapsed real time in seconds

Result

The status bar is shown or hidden.

Related topics

Basics

User Interface of the VEOS Player..... 382

Switch Panes

Access

You can access this command via:

Ribbon	View – Controlbar
Context menu of	None
Shortcut key	None
Icon	

Purpose

To show or hide panes.

Result

Panes are either shown or hidden.

Description

Opens a submenu that shows the following panes available in VEOS:

- Messages Pane on page 401
- Model Communication Pane on page 116
- Network Topology Pane on page 118
- Port Topology Pane on page 120
- Properties pane (refer to User Interface of the VEOS Player on page 35)

To hide a pane, use the **Close** symbol in the pane's header.

Related topics

Basics

User Interface of the VEOS Player 35

HowTos

How to Customize the Screen Arrangement 386

References

Messages Pane 401
Model Communication Pane 116
Network Topology Pane 118
Port Topology Pane 120

Undo

Access

You can access this command via:

Ribbon	None
Context menu of	None
Shortcut key	**Ctrl** + **Z**
Icon	
Others	Quick Access Toolbar

Purpose

To undo the most recent edit commands or actions.

Result

The previously performed commands and actions are undone.

Description

This lets you undo the most recent commands or actions. If you carry the command out once, one command or action is undone; if you carry it out twice, two are undone, and so on.

Tip

You can click the down arrow to get a list of the actions that can be undone. If you select one of the actions from the list, all actions up to that are undone. The undo list contains up to 20 actions.

Related topics

References

Redo..... 408

Using dSPACE Help

Access

You can access this command via:

Ribbon	File – Help
Context menu of	None
Shortcut key	**F1**
Icon	

Purpose

To get information on working with dSPACE Help.

Result

dSPACE Help opens. It provides information on general handling and instructions on using dSPACE Help.

Automation

Automating the Start of the VEOS Player, and Getting Version Information

Introduction

You can automate the start of the VEOS Player.

The program listings below consist of excerpts from the Python demo scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: [...].

Opening a COM connection to the VEOS Player

The following listing shows how you can import required modules to the interpreter namespace.

```
# Import required modules.
import win32com
from win32com.client import Dispatch
```

The following listing shows how you can open a COM connection to the VEOS Player to use its API functions.

```
# Open a COM conection to the VEOS Player.
player = Dispatch("VeosPlayer.Application")
```

The `Dispatch("VeosPlayer.Application")` command opens a COM connection to the VEOS Player. When you execute the listing above in an external Python interpreter, the VEOS Player is started if it is not already running.

Supplying version information To automate a specific VEOS version, you can supply version information when you open the COM connection.

Refer to Automating Different VEOS Player Versions on page 329.

Appending additional folders to the Python search path

The following listing shows how you can append additional folders to the Python search path.

```
# Append the VEOS scripting folder to the Python search path.
ApplicationRootDir = player.ApplicationRootDir
ApplicationScriptingDir = os.path.join(ApplicationRootDir, "Tools", "Scripting")
sys.path.insert(0, ApplicationScriptingDir)
```

Python searches the `.\Tools\Scripting` folder for modules and script files.

Adding VEOS' enumeration module

The following listing shows how you can add VEOS' enumeration module, which is located in the `.\Tools\Scripting` folder of the VEOS installation.

```
# The enum definitions are needed later.
import VeosEnums
import importlib
importlib.reload(VeosEnums)
```

You can use enumerations, for example, to compare VPU port directions via `VeosEnums.VpuPortDirection.Out` and `VeosEnums.VpuPortDirection.In`.

Getting version information on VEOS Player

The following listing shows how you can get information on the VEOS Player version including the major, minor, and maintenance version.

```
def PrintVersionInformations(player):
    print("VEOS Player version is '%s'."% player.VersionInformation.Version)
    print("VEOS Player maintenance version is '%s'." % player.VersionInformation.VersionMaintenanceNumber)
    print("VEOS Player major version is '%s'." % player.VersionInformation.VersionMajorNumber)
    print("VEOS Player minor version is '%s'." % player.VersionInformation.VersionMinorNumber)
    print("VEOS Player get version information: ")
    print(player.VersionInformation.get)
```

Related topics

Basics

Basics of the VEOS Player Automation Interface..... 326

References

IApplication..... 343

Glossary

Introduction

Briefly explains the most important expressions and naming conventions used in the VEOS documentation.

Where to go from here

Information in this section

A....418
B....419
C....420
D....420
E....421
F....421
H....422
I....422
L....422
M....423
N....423
O....423
P....424
Q....424
R....424
S....425
T....426
V....427

X.....428
Z.....428

A

A2L file A description of the variables available for measurement, calibration, and stimulation on a (V-)ECU in the standardized ASAM MCD-2MC format. It also describes the interface used to connect to the platform/device. A2L descriptions are often used in connection with an XCP service.

Adaptive V-ECU A V-ECU that allows for the *dynamic* integration of new applications and changes to the network communication at run time, which is not possible with a *statically* configured classic V-ECU. Communication is service-oriented via Ethernet.

An example are V-ECUs developed according to the AUTOSAR Adaptive Platform.

Application software The functional part of the ECU software according to the AUTOSAR Classic Platform. It is the layer above the run-time environment (RTE) and the basic software.

The application software consists of software components that implement the controller behavior, for example.

AUTOSAR Abbreviation of *AUT*omotive *O*pen *S*ystem *AR*chitecture.

The AUTOSAR partnership is an alliance in which the majority of OEMs, suppliers, tool providers and semiconductor companies work together to develop and establish a de-facto open industry-standard for automotive electric/electronics (E/E) architecture and to manage the growing E/E complexity.

AUTOSAR Adaptive Platform A standardized ECU software architecture for high-performance ECUs that handle complex tasks, such as tasks for autonomous driving.

AUTOSAR Classic Platform A standardized ECU software architecture that can be separated in the following parts:

- Application software
- Run-time environment (RTE)
- Basic software

B

Basic software The generic term for the following software that is part of the ECU software according to the AUTOSAR Classic Platform:

- Operating system (OS)
- AUTOSAR Services
- Communication stack
- I/O stack (ECU abstraction and microcontroller abstraction)
- Memory stack
- Complex device driver

Together with the run-time environment, the basic software is the platform for the ECU application software.

Binary signal A variable-size signal of the binary data type. Unlike variable-size signals that are mathematical arrays with independent array elements, binary signals are byte arrays with undefined content, such as sensor or audio data. The content is described by a MIME type.

Blocking communication A communication protocol type in which the VEOS Simulator transfers data between VPUs in the same (virtual) sampling step. If the VPU that provides the data has not calculated the data yet, the VPU that receives the data must wait until the other provides the data.

BSC file Abbreviation of Bus simulation container (BSC) file

Bus model A model that represents a restbus (or parts of a restbus) for the ECUs to be tested in a simulation scenario. The bus model is part of the simulation system.

Bus simulation container (BSC) A container that contains an implementation of bus communication which is configured with the Bus Manager.

- BSC files can be imported to VEOS Player to implement the configured bus communication in offline simulation application and perform offline simulation on VEOS.
- BSC files can be imported to ConfigurationDesk to implement the configured bus communication in a real-time application and perform real-time simulation on SCALEXIO or a MicroAutoBox III.

Bus type A structural element in the right tree view of the Network Topology pane. It contains communication clusters and their connections to communication controllers.

Bus VPU The executable of a bus model built for the VEOS platform. A bus VPU is part of an offline simulation application (OSA).

C

Classic V-ECU A V-ECU whose resources, such as the operating system and network communication, are statically configured. Therefore, in contrast to an adaptive V-ECU, you cannot add applications or make changes to the network communication at run time. Communication is mainly signal-based, as in network communication via CAN or FlexRay, but later classic V-ECUs also support service-oriented communication via Ethernet and SOME/IP.

An example are V-ECUs developed according to the AUTOSAR Classic Platform.

Code-based V-ECU A classic V-ECU that is based on external AUTOSAR or non-AUTOSAR code files.

Communication cluster An element that specifies the topological connection of communicating ECUs. In a communication cluster, the communication controllers of different ECUs are interconnected.

Communication controller A hardware device of an ECU for sending information to and receiving information via a communication channel.

Communication package An element that specifies the communication protocol type (either blocking or non-blocking) of the assigned ports.

Container A running instance of a container image. A container represents a largely isolated environment that typically runs a single process by using the resources of the host operating system.

Container image A template for a container, i.e., a set of instructions for creating a container that can run on the container platform.

Controller model A model that represents a control function of an ECU in a simulation scenario. The controller model is a part of the simulation system.

Controller VPU The executable of a controller model built for the VEOS platform. A controller VPU is part of an offline simulation application (OSA).

Controller-to-cluster connection A connection between a communication controller and a communication cluster.

D

Data access point An access point defined in a platform-independent Dap module configuration for accessing I/O signals specified in the IoHwAb module configuration and/or simulation-specific RTE interventions.

Dockerfile A text file containing all the instructions that are required for creating a Docker container image.

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

DSKVM dSPACE-specific hypervisor that is used by VEOS on Linux for adaptive V-ECU simulation.

dSPACE Linux kernel A dSPACE-specific Linux kernel to run Linux-based applications, such as adaptive V-ECUs. During offline simulation on VEOS, the VEOS Simulator and the dSPACE Linux kernel are synchronized, i.e., the Linux-based application is executed synchronously to the rest of the simulation system.

dSPACE Run-Time Target (DSRT) Target for Simulink® Coder™ for generating a Simulink implementation container (SIC file) based on a Simulink® model.

E

ECU Abbreviation of *electronic control unit*.

Environment model A model that represents a part or all of the ECU's environment in a simulation scenario.

The environment model is a part of the simulation system.

Environment VPU The executable of an environment model built for the VEOS platform. An environment VPU is part of an offline simulation application (OSA).

F

Fixed-size signal A scalar signal (data width =1) or an array signal with a data width >1. The width, i.e., the number of array elements is constant during run time.

FMI Abbreviation of Functional Mock-up Interface (FMI)

FMU Abbreviation of Functional Mock-up Unit (FMU)

Functional Mock-up Interface (FMI) Functional Mock-up Interface (FMI) is a tool-independent standard that supports both the co-simulation and the exchange of dynamic models by using archive files containing a combination of XML files and C functions provided in source and/or binary form. The FMI

standard defines the interface to be implemented by a Functional Mock-up Unit (FMU).

Functional Mock-up Unit (FMU) A Functional Mock-up Unit (FMU) implements the Functional Mock-up Interface (FMI). An FMU describes and implements the functionality of a model. It is an archive file with the FMU file name extension. An FMU file contains:

- The functionality defined in C source code files or binary files (e.g., DLL files) or both
- The `modelDescription.xml` file with the description of the interface data
- Additional resources needed for simulation

H

Hypervisor A virtual operating platform for running one or more virtual machines. A hypervisor manages the execution of the operating systems of the virtual machines. Hypervisors abstract from the host computer hardware such as the CPU and memory, which makes the execution of the virtual machines independent of the hardware.

Hypervisors are also known as *virtual machine monitors* (VMM) and *virtualizers*.

There are two types of hypervisors:

- Type-1 (native or bare-metal) hypervisors

  Hypervisors that run directly on the hardware of the host computer. They require no operating system on the host computer.
- Type-2 (hosted) hypervisors

  Hypervisors that run on the operating system of the host computer.

I

Init script A shell script that is executed during the boot process of a Linux operating system. The purpose of the script is to start required system processes in the correct order.

L

Local simulator The VEOS simulation platform installed on the local machine.

M

Model A description of the behavior and/or the design of a system in a high-level language, often with graphical elements. A model abstracts from details of the real system and provides a simplified view for the intended purpose.

In the context of SIL testing, for example, Simulink® can be used for environment modeling and AUTOSAR/TargetLink for V-ECU models. A model is the basis for the generation of a model implementation.

Model implementation Implementation of a model in a selected programming language such as C or C++.

Model-based V-ECU A V-ECU that is based on an ECU in an AUTOSAR system that is modeled in or imported to SystemDesk.

N

Network adapter A device that connects a computer to a network.

Non-blocking communication A communication protocol type in which the VEOS Simulator transfers data between VPUs after a (virtual) sampling step.

O

Offline simulation A PC-based simulation in which the simulator is not connected to a physical system and is thus independent of the real time.

Offline simulation application (OSA) An offline simulation application (OSA) file is an executable file for VEOS. After the build process with a tool such as the VEOS Player, the OSA file can be downloaded to VEOS.

An OSA contains one or more VPUs, such as V-ECUs and/or environment VPUs.

Unlike a model implementation, such as a V-ECU implementation (VECU), an OSA file does not contain source code files. It is therefore suitable for exchanging offline simulation applications while protecting intellectual property (IP).

However, the OSA file also contains a variable description file with symbol names for each VPU, which may impair IP protection.

OSA Abbreviation of offline simulation application (OSA)

P

Port A structural element of a VPU.

A port can contain the following elements:

- Signal groups
- Signals

Port group A structural element of a VPU.

A port group can contain the following elements:

- Port groups
- Ports

Q

QEMU image A disk image supported by the QEMU hypervisor software. An example is the RAW format, which represents a highly portable plain binary image.

R

Real-time acceleration factor A factor to accelerate (> 1) or decelerate (< 1) an offline simulation run in relation to real time or to approximate real time (= 1).

Specifying a real-time acceleration factor that is not equal to 0 enables soft real time simulation.

Remote simulator The VEOS simulation platform installed on a remote machine, i.e., a physical or virtual machine in the local network.

Restbus The simulated bus communication of ECUs that are *not* the devices under test (DUTs) in a simulation system. A restbus is needed to perform restbus simulation.

Restbus simulation A use scenario in which some ECUs of a simulation system are tested, and the bus communication of other ECUs of the simulation system is simulated by a restbus. The ECUs under test are connected to the restbus. The bus communication of the ECUs in the restbus can be manipulated, and the effects on the ECUs under test can be observed.

RTE intervention A mechanism that lets you access the RTE internal communication of sender-receiver and client-server interfaces. You can read and modify the data elements and operation arguments transmitted between the

interfaces. You can also modify the status return values of RTE API functions. All this lets you, for example, stimulate ports or inject error states during a simulation run to test the behavior of application software components.

Run-time environment (RTE) A generated software layer that connects the ECU application software to the basic software. It also interconnects the different software components of the ECU application software. There is one RTE per ECU.

S

SDF file The system description file that describes the files to be loaded to the individual processing units of a simulation platform. It also contains the variable description of the relevant simulation application.

The SDF file is generated automatically when the TRC file is built.

SIC file Abbreviation of *Simulink implementation container* file.

A file archive that contains the model code of a Simulink® model and a description of all the model's external dependencies. An SIC file also contains an XML description of the container content, makefiles and variable description files (SDF/TRC or A2L).

An SIC file represents a Simulink implementation container (SIC).

Signal An element of a VPU that allows direct communication with other VPUs.

There are different signal types:

- Fixed-size signals
- Variable-size signals
- Binary signals

Signal group A structural element of a VPU.

A signal group can contain the following elements:

- Signal groups
- Signals

Simulation system A description of the composition of V-ECU models, environment models, real ECUs, and their interconnections required for simulating the behavior of a system. A simulation system is the basis for the generation of a simulation application for a given simulator platform.

Simulator A system that imitates the characteristics or behaviors of a selected physical or abstract system.

Simulink implementation container (SIC) An implementation of a Simulink® model. It contains the source code files, a variable description and descriptions of the interface to the simulator platform.

A Simulink model implementation container can be generated with the *Model Interface Package for Simulink*. The generation output is an SIC file.

SMC file Abbreviation of system model container (SMC) file.

A file archive that contains an arbitrary number of the following containers, which belong to a specific system model:

- Bus simulation container (BSC)
- Functional Mock-up Unit (FMU)
- Simulink implementation container (SIC)
- V-ECU implementation (VECU)

An SMC file additionally contains the following information:

- Description of the data flow connections between the components of the system model
- Description of the bus communication connections between the components of the system model

An SMC file represents a system model container (SMC).

Soft real-time simulation An offline simulation in which VEOS tries to keep the *virtual simulation time* as close to *real-time* (or a multiple of it) as possible with regard to the constraints of the host PC's operating system. Soft real time is enabled for an offline simulation when its real-time acceleration factor is not equal to 0.

System An ECU network and its environment.

System model container (SMC) An implementation of a system model. It contains the implementations of the individual models of a specific system, including the model interconnections.

A system model container can be exported from SYNECT Model Management and VEOS. The output is an SMC file.

T

Tarball A type of archive file. The filen ame extension of a tarball is TAR. A tarball that was compressed can have an additional file name extension, such as GZ.

TRC file A variable description file with information on the variables available in an environment model running on a dSPACE platform.

V

Variable-size signal An array signal with a data width >1. The width, i.e., the number of array elements varies during run time.

V-ECU Abbreviation of *virtual ECU*.

ECU software that can be executed in a software-in-the-loop (SIL) testing environment such as a local PC or highly parallel in the cloud independently of real-time constraints and real ECU hardware.

V-ECU is also the generic term for:

- Adaptive V-ECU
- Classic V-ECU

VECU file A container file that contains all the files associated with a V-ECU implementation (VECU).

V-ECU implementation (VECU) An implementation of a V-ECU model. It can be created both for classic and adaptive V-ECUs (refer to Basics on Exporting FMUs from TargetLink (TargetLink Interoperation and Exchange Guide)).

A V-ECU implementation is created with SystemDesk and TargetLink. It is exported as a VECU file.

VEOS A simulator which is part of the PC and allows the user to run an offline simulation application (OSA) without relation to real time.

VEOS basically consists of the following components:

- VEOS System Builder
- VEOS Simulator

VEOS Player is the graphical user interface for VEOS.

VEOS Player The graphical user interface for VEOS on Windows.

VEOS Player provides an automation interface.

VEOS Simulator The VEOS component for:

- Controlling the execution of offline simulations.
- Interconnecting all the VPUs of a simulation system during offline simulation.

VEOS Simulator is executed in a separate task of the PC's operating system.

VEOS System Builder The VEOS component for:

- Building a model implementation for offline simulation on VEOS.
- Integrating these model implementations into a simulation system.

View set A configuration of the geometry, visibility, and docking or floating state of the panes of VEOS Player. A view set has a default state and a current state that can differ from the default state.

Virtual machine A virtual computer that runs on a real computer and that provides the same functions as a real computer. Like real computers, virtual machines run an operating system and applications.

Host computer hardware resources required by a virtual machine are managed by a *hypervisor*.

VPU Abbreviation of *virtual processing unit*. A VPU is part of an offline simulation application in VEOS. Each VPU runs in a separate process of the PC.

VPU is also the generic term for:

- V-ECUs
- Environment VPUs
- Controller VPUs
- Bus VPUs

X

XCP Abbreviation of *Universal Measurement and Calibration Protocol*. A protocol that is implemented on electronic control units (ECUs) and provides access to ECUs with measurement and calibration systems (MCS) such as ControlDesk.

XCP is based on the *master-slave principle*:

- The ECU is the slave.
- The measurement and calibration system is the master.

The "X" stands for the physical layers for communication between the ECU and the MCS, such as CAN (Controller Area Network) and Ethernet.

The basic features of XCP are:

- ECU parameter calibration (CAL)
- Synchronous data acquisition (DAQ)
- Synchronous data stimulation (STIM), i.e., for bypassing
- ECU flash programming (PGM)

The XCP protocol was developed by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems e.V.). For the protocol specification, refer to http://www.asam.net.

Z

Zero execution time assumption An assumption for offline simulation, which is common for most offline simulators such as Simulink and VEOS.

According to the zero execution time assumption, all tasks are executed *instantly* on the virtual simulation time scale, i. e., all tasks are assumed to run with an execution time of $\Delta t_{Virtual} = 0$. As a consequence, reproducible simulation results that are independent of the target hardware can be achieved.

After the offline simulator has executed all the tasks for a discrete simulation event, it jumps to the next simulation event in the simulation queue.

Appendix: Message Reader API

Where to go from here

Information in this section

Introduction to the Message Reader API.....432
dSPACE.Common.MessageHandler.Logging Reference.....439

Introduction to the Message Reader API

Where to go from here

Information in this section

Reading dSPACE Log Messages via the Message Reader API.....432
You can read log messages of the dSPACE Log via the Message Reader API.

Supported dSPACE Products and Components.....434
Provides an overview of all dSPACE products and components whose messages you can access via the Message Reader API.

Example of Reading Messages with Python.....434
You can read the log messages via Python. You can combine multiple filters to display only messages according to your specifications.

Example of Reading Messages with C#.....436
You can read the log messages via C#. You can combine multiple filters to display only messages according to your specifications.

Reading dSPACE Log Messages via the Message Reader API

Introduction

You can read log messages of the dSPACE Log via the Message Reader API.

dSPACE Log

The dSPACE Log is a collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

The dSPACE Log is saved as a collection of binary message log files. These files are created when a dSPACE product is running. A single run of a dSPACE product is called a *log session*.

Note

If the maximum file size for the binary message log file is reached, messages at the beginning of the dSPACE Log might get deleted. Contact dSPACE Support to solve this.

Message Reader API

You can use the Message Reader API to access all binary message log files of the dSPACE Log. You can combine multiple filters to display only log messages according to your specifications. For example, you can configure the Message Reader API to display only log messages from a specific dSPACE product.

The Message Reader API is available as of dSPACE Release 2020-A. For information on the dSPACE products and components that support the Message Reader API, refer to Supported dSPACE Products and Components on page 434.

dSPACE.Common.MessageReader.dll The Message Reader API is implemented by the `dSPACE.Common.MessageReader.dll` file. It is located in the `bin` subfolder of the installation folder of each dSPACE product that supports the Message Reader API.

Supported dSPACE Releases

The Message Reader API lets you access log messages written by dSPACE products since dSPACE Release 2016-B.

Message Reader API change in dSPACE Release 2021-A

There is a migration issue specific to the Message Reader API. The issue occurs if you use the API with Python. The issue was caused by the migration to Python 3.9/pythonnet 2.5.3 with dSPACE Release 2021-A.

There is no migration issue to consider if you use the API with C#.

Specifying a product filter As of dSPACE Release 2021-A, the `Products` property of the `MessageReaderSettings` class can no longer be used to set the list of products for which to filter in the log sessions. The Message Reader API provides the `SetProducts` method for this purpose.

The following table shows how to specify a product filter before and after migration:

Using Message Reader API of ...	
... dSPACE Release 2020-B and Earlier (Python 3.6)	**... dSPACE Release 2021-A and Later (Python 3.9)**

`# Specify products whose messages to read:`
`Settings = MessageReaderSettings()`
`Settings.Products.Add('ControlDesk')`
`Settings.Products.Add('AutomationDesk')`
	`# Specify products whose messages to read:`
	`Settings = MessageReaderSettings()`
	`Settings.SetProducts(['ControlDesk', 'AutomationDesk'])`

Related topics

Basics

Supported dSPACE Products and Components 434

Examples

Example of Reading Messages with C# 436
Example of Reading Messages with Python 434

References

MessageReaderSettings Class 443

Supported dSPACE Products and Components

Supported dSPACE products and components

You can use the Message Reader API to access messages from the following dSPACE products and components:

- ASM KnC
- AutomationDesk
- Bus Manager (stand-alone)
- cmdloader
- ConfigurationDesk
- Container Management
- ControlDesk
- dSPACE AUTOSAR Compare
- dSPACE XIL API .NET Implementation
- Firmware Manager
- ModelDesk
- MotionDesk
- Real-Time Testing
- RTI Bypass Blockset
- SYNECT client
- SystemDesk
- TargetLink Property Manager
- VEOS

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Example of Reading Messages with Python

Introduction

You can read the log messages via Python by using the `clr` module. You can combine multiple filters to display only messages according to your specifications.

Referencing a message reader assembly

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to dSPACE.Common.MessageReader.dll on page 433.

In the following examples it is assumed that the dSPACE Installation Manager is installed and that the message reader assembly is installed in `C:\Program Files\Common Files\dSPACE\InstallationManager\bin`.

The following code references and imports the message reader assembly.

```
# Insert path of message log file access assembly:
import sys
AssemblyPath = r'C:\Program Files\Common Files\dSPACE\InstallationManager\bin'
if not sys.path.count(AssemblyPath):
    sys.path.insert(1, AssemblyPath)

# Add reference to assembly and import it:
import clr
clr.AddReference('dSPACE.Common.MessageReader')
from dSPACE.Common.MessageHandler.Logging import *
```

Reading all messages

The following example reads all existing message log files and prints all messages via Python. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 434.

```
# Create message reader and print text of each message:
Reader = MessageReader(None)
for Message in Reader.ReadMessages():
    print(Message.MessageText)
Reader.Dispose()
```

Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of ControlDesk and AutomationDesk are read and printed. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 434.

```
# Define error severities:
SEVERITY_ERROR = 3
SEVERITY_SEVERE_ERROR = 4
SEVERITY_SYSTEM_ERROR = 5

# Configure products and sessions whose messages to read:
Settings = MessageReaderSettings()
Settings.MaximalSessionCount = 1
Settings.SetProducts(['ControlDesk', 'AutomationDesk'])

# Create message reader and print text of each error message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    # Print error messages only:
    if Message.Severity == SEVERITY_ERROR or \
       Message.Severity == SEVERITY_SEVERE_ERROR or \
       Message.Severity == SEVERITY_SYSTEM_ERROR:
        print('%s: %s' % (Message.Session.ProductName, Message.MessageText))
Reader.Dispose()
```

Note

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to MessageReader Class on page 442.

Filtering messages by time

Times are given by .NET DateTime objects. Times are given as UTC times (Coordinated Universal Time). You can obtain the current UTC time by `System.DateTime.UtcNow`.

The following example reads all messages after a certain start time. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 434.

```
import System
Settings = MessageReaderSettings()
Settings.MessageTimeAfter = System.DateTime.UtcNow # Read messages after now

# Create message reader and print time and text of each message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    print('%s: %s' % (Message.UtcTimeStamp, Message.MessageText))
Reader.Dispose()
```

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API 432
Supported dSPACE Products and Components 434

References

MessageReaderSettings Class 443

Example of Reading Messages with C#

Introduction

You can read the log messages via C#. You can combine multiple filters to display only messages according to your specifications.

Referencing a message reader assembly

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to dSPACE.Common.MessageReader.dll on page 433.

Reading all messages

The following example reads all existing message log files and prints the messages:

```
using dSPACE.Common.MessageHandler.Logging;
...

// Create message reader and print text of each message:
using (MessageReader reader = new MessageReader(null))
{
    foreach (message in reader.ReadMessages())
    {
        Console.WriteLine(message.MessageText);
    }
}
```

Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of ControlDesk and AutomationDesk are read and printed.

```
using dSPACE.Common.MessageHandler.Logging;
...

// Read the last log sessions of ControlDesk and AutomationDesk only:
MessageReaderSettings settings = new MessageReaderSettings();
settings.MaximalSessionCount = 1;
settings.Products.Add("ControlDesk");
settings.Products.Add("AutomationDesk");

using (MessageReader reader = new MessageReader(settings))
{
    foreach (ILogMessage message in reader.ReadMessages())
    {
        // Print error messages only:
        if (message.Severity == Severity.Error
            || message.Severity == Severity.SevereError
            || message.Severity == Severity.SystemError)
        {
            Console.WriteLine(message.Session.ProductName + ": " + message.MessageText);
        }
    }
}
```

Note

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to MessageReader Class on page 442.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Supported dSPACE Products and Components..... 434

References

MessageReaderSettings Class..... 443

dSPACE.Common.MessageHandler.Logging Reference

Where to go from here

Information in this section

ILogMessage Interface 439
To access information about a message as written to a log file.

ILogSession Interface 440
To access information about a message log session.

MessageReader Class 442
To read serialized messages written by dSPACE products.

MessageReaderSettings Class 443
To define the settings of a message reader.

Severity Enumeration 445
To specify the severity of a message.

ILogMessage Interface

Namespace

`dSPACE.Common.MessageHandler.Logging`

Description

To access information about a message as written to a log file.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
IsStartMessage	Gets a value indicating whether the message is a session start message.	Get	*Boolean*
IsStopMessage	Gets a value indicating whether the message is a session stop message.	Get	*Boolean*
MainModuleNumber	Gets the main module number of the message.	Get	*Integer*
MessageCode	Gets the error code of the message.	Get	*Integer*
MessageText	Gets the text of the message.	Get	*String*
ModuleName	Gets the module name of the message.	Get	*String*
Session	Gets the log session which issued the message.	Get	ILogSession (refer to ILogSession Interface on page 440)
Severity	Gets the severity of the message.	Get	Severity (refer to Severity Enumeration on page 445)

Name	Description	Get/Set	Type
SubmoduleNumber	Gets the submodule number of the message.	Get	*Integer*
ThreadId	Gets the thread ID of the submitting thread.	Get	*Integer*
TimeStamp	Gets the time when the message was submitted. Given as local time in the time zone of the session.	Get	*DateTime*
UtcTimeStamp	Gets the time when the message was submitted in UTC time.	Get	*DateTime*

Methods

The element has no methods.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Examples

Example of Reading Messages with C#..... 436
Example of Reading Messages with Python..... 434

References

ILogSession Interface..... 440
Severity Enumeration..... 445

ILogSession Interface

Namespace

`dSPACE.Common.MessageHandler.Logging`

Description

To access information about a message log session.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
CloseTime	Gets the time when the session was closed. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully. Given as local time in the time zone of the session.	Get	*DateTime*

Name	Description	Get/Set	Type
IsOpen	Gets a value indicating whether the session is still open. If true, the session is still open and new messages can be written.	Get	*Boolean*
IsValid	Gets a value indicating whether the session is valid. A session can become invalid if its log files are corrupted.	Get	*Boolean*
MetaData	Gets the products metadata as read from log file session info.	Get	*Dictionary< String, String >*
ProcessId	Gets the process ID of the log session.	Get	*Integer*
ProductName	Gets the product name of the log session.	Get	*String*
SessionId	Gets the ID of the log session. This ID is unique in the context of its session reader.	Get	*Integer*
StartTime	Gets the sessions start time. Given as local time in the time zone of the session.	Get	*DateTime*
TimezoneName	Gets the standard time zone name of the session.	Get	*String*
TimezoneOffset	Gets the time zone offset of the session relative to UTC.	Get	*TimeSpan*
UtcCloseTime	Gets the time when the session was closed as UTC time. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully.	Get	*DateTime*
UtcStartTime	Gets the start time of the log session as UTC time.	Get	*DateTime*

Methods

The element has the following methods:

Name	Description	Parameter[1)]	Returns
ToSessionTime	Converts UTC time to time zone used when the session was written.	▪ *<DateTime>* `utcTime`: Specifies the UTC time to convert.	Time in the time zone of the logging session. ▪ DateTime

[1)] <Type> `Name`: Description

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Examples

Example of Reading Messages with C#..... 436
Example of Reading Messages with Python..... 434

MessageReader Class

Description

To read serialized messages written by dSPACE products.

Constructor

The element has the following constructor:

Name	Description	Parameter[1]	Returns
MessageReader	Initializes a new instance of the MessageReader class.	▪ <MessageReaderSettings>[2] `settings`: Settings which allow to specify which sessions and messages are read. Can be null, causing all existing log files to be read.	None

[1] <Type> `Name`: Description
[2] Refer to MessageReaderSettings Class on page 443

Properties

The element has no properties.

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
Dispose	Performs application-specific tasks associated with freeing, releasing, or resetting unmanaged resources.	None	None

Name	Description	Parameter[1]	Returns
ReadMessages	Reads the messages written to the log files of the sessions up to now. The messages are returned in chronological order according to their time stamps. **Note** The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time.	None	Messages read from log file. ▪ IEnumerable< ILogMessage (refer to ILogMessage Interface on page 439) >

[1] <Type> `Name`: Description

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Examples

Example of Reading Messages with C#..... 436
Example of Reading Messages with Python..... 434

References

MessageReaderSettings Class..... 443

MessageReaderSettings Class

Description

To define the settings of a message reader.

Used to filter the log sessions and messages read.

Constructor

The element has the following constructor:

Name	Description	Parameter[1)]	Returns
MessageReaderSettings	Initializes a new instance of the MessageReaderSettings class.	None	None

1) <Type> `Name`: Description

Properties

The element has the following properties:

Name	Description	Get/Set	Type
DirectoryNames	Gets a list of specific directory names from which to read log files. If the list is empty, all standard directories are searched for log files.	Get	*List< String >*
MaximalSessionCount	Gets or sets the maximal number of log sessions read for each product. If the count is a positive number n, only the last n sessions are read. If the count is not positive, an unlimited number of sessions is read. The default value is zero, i.e., unlimited.	Get/Set	*Integer*
MessageTimeAfter	Gets or sets the minimal time for which messages are read, given as UTC time. Only messages submitted after the message time are read. The message time may be in the past. The message time must be given as valid UTC time. The default time is undefined, i.e., each message time is allowed.	Get/Set	*DateTime*
Products	Gets the list of product names for which to read log sessions. If the list is empty sessions of all products are read.	Get	*List< String >*
StartTimeAfter	Gets or sets the minimal start time for which sessions are read, given as UTC time. Only sessions which started after the start time are read. The start time may be in the past. The start time must be given as valid UTC time. The default time is undefined, i.e., each start time is allowed.	Get/Set	*DateTime*

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
SetDirectoryNames	Sets the list of specific directory names from which to read log files. You do not have to specify a list. If the list is empty, all standard directories are searched for log files.	*<string[]>* `names`: Array of directory names.	None
SetProducts	Sets the list of product names for which to read log sessions.	*<string[]>* `products`: Array of product names.	None

[1] <Type> `Name`: Description

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Examples

Example of Reading Messages with C#..... 436
Example of Reading Messages with Python..... 434

Severity Enumeration

Description

To specify the severity of a message.

Enumeration values

The enumeration has the following values:

Value	Name	Description
0	Trace	A trace message. Trace messages are usually not created. It depends on the host application if it is possible to configure the message handler to create trace messages.
1	Info	An information message.
2	Warning	A warning message.
3	Error	An error message.
4	SevereError	A severe error message.
5	SystemError	A system error message.
6	Question	A question message.
7	Advice	An advice message.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 432

Examples

Example of Reading Messages with C#..... 436
Example of Reading Messages with Python..... 434

A

Adaptive Platform
  AUTOSAR Demonstrator 266
  demos 248
adaptive V-ECU
  basics on simulating 236
  workflow 241
all child connections
  disconnect 143
Auto Hide command 391
autoconnect
  VPU signals 138
automation
  accessing the VEOS Player's automation interface 326
  automating different versions 329
  basic features of the automation interface 326
  supported automation languages 327

B

basics
  basics features of the automation interface 326
  ribbons 384
  VEOS Player 30
BSCs
  importing 78
bus communication
  simulating 199

C

Clear Recently Exported Files 140
Clear Recently Imported Files 140
Clear Recently Used Files 141
Clear simulator list 213
close all but this command 392
commands
  VEOS Player 134
Common Program Data folder 14
communication cluster
  delete 143
communication controllers/clusters
  connecting 111
communication package
  delete 143
connecting
  communication controllers/clusters 111
  VPU signals 100
customizing
  screen arrangement 386

D

delete
  communication cluster 143
  communication package 143
  signal 143
  VPU 143
dialogs
  VEOS Player 134
disconnect
  all child connections 143
Dock command 393
Documents folder 14, 393, 421

E

elements
  VEOS Player 122
Export Variable Description Files on save 146

F

Filter Editor 147
Float 398
FMI 84
FMU 84
FMUs
  import 152
  importing 90
Functional Mock-up Unit 84

H

Hide command 400
hide connected communication clusters 149
hide connected communication controllers 150
Hide Connected Incoming Ports 150
Hide Connected Outgoing Ports 151

I

import
  FMUs 152
  Simulink implementation containers 152
  V-ECU implementations 152
  VPUs 152
importing
  BSCs 78
  FMUs 90
  Simulink implementations 71
  System model containers 95
  V-ECU implementations 67
  VPUs 97

K

keyboard help 400

L

Local Program Data folder 14

M

Messages 401
move to next tab group command 404
move to previous tab group command 404

N

new
  OSA file 159
new features
  VEOS 16
New Features and Migration command 405
new horizontal tab group command 405
new vertical tab group command 406

O

offline simulation application 30
open
  OSA file 159
opening
  OSA file 33
OSA file
  new 159
  open 159
  opening 33
  save 161
  save as 162

P

PDF files command 407

R

Recently used 160
Redo command 408
replace 161
Reset View Set 410
ribbons
  basics 384
running an offline simulation application 188

S

save
  OSA file 161
save as
  OSA file 162
screen arrangement
  customizing 386
simulating bus communication 199
simulation
  specific variables 189
Simulation Target Manager 163
simulation-specific variables
  simulation 189
Simulink implementations
  importing 71
Status Bar 411
Switch Panes 411
System model containers
  importing 95

U

Undo command 412
update 166
user interface
  VEOS Player 35, 382

V

V-ECU implementations
  import 152
V-ECU implementations
  importing 67
VEOS
  new features 16
VEOS Player
  automation features 326
  basics 30
  commands 134
  dialogs 134
  elements 122
  user interface 35, 382
VPU
  delete 143
VPU signals
  autoconnect 138
  connecting 100
VPUs
  import 152
  importing 97