SIL Testing

# Overview

Release 2021-A – May 2021

dSPACE

## How to Contact dSPACE

| | |
|---|---|
| Mail: | dSPACE GmbH<br>Rathenaustraße 26<br>33102 Paderborn<br>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

# Contents

# About This Document

**Content**

This document introduces you to SIL testing and provides an overview of the software tools which are used to:

- Develop and integrate a simulation system consisting of environment models connected to one or more virtual ECUs
- Perform tests with virtual ECUs

The document describes the workflow for using virtual ECUs in offline simulations on VEOS, in hardware-in-the-loop (HIL) simulations on SCALEXIO, and in rapid prototyping simulations on a MicroAutoBox III.

**Required knowledge**

The objective of this document is to provide overview of the use cases and features of SIL testing, such as the frontloading of HIL tests to a PC-based 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. |

| Symbol | Description |
|---|---|
| ❓ | 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.

# New SIL Testing Features and Product Compatibility

**Where to go from here**

Information in this section

## New Features of SIL Testing in dSPACE Release 2021-A

**New features**

The following SIL testing features are new in dSPACE Release 2021-A:

| Feature | Provided by | More Information |
|---|---|---|
| **Linux Execution Tool Chain** | | |
| Running VEOS in a Linux Docker container | VEOS 5.2 | Basics on Running VEOS 5.2 in a Linux Docker Container (VEOS Manual 📖) |
| GPU support for adaptive V-ECU simulation | VEOS 5.2 | Introduction to the GPU Passthrough Demo (VEOS Manual 📖). |
| dSPACE XIL API .NET implementation for VEOS on Linux | dSPACE XIL API .NET implementation 2021-A | New Features of dSPACE XIL API .NET 2020-B (New Features and Migration 📖) |
| Real-Time Testing for VEOS on Linux | Real-Time Testing 5.0 | New Features of Real-Time Testing 5.0 (New Features and Migration 📖) |
| **Further Improvements** | | |
| Support of variable-size signals and binary signals | VEOS 5.2 | Basics on Signal Connection Dependencies (VEOS Manual 📖) |

| | |
|---|---|
| **Related topics** | Basics<br>Basics on Running VEOS 5.2 in a Linux Docker Container (VEOS Manual 🕮)<br>Basics on Signal Connection Dependencies (VEOS Manual 🕮)<br>Introduction to the GPU Passthrough Demo (VEOS Manual 🕮)<br>New Features of dSPACE XIL API .NET 2020-B (New Features and Migration 🕮)<br>New Features of Real-Time Testing 5.0 (New Features and Migration 🕮) |

## Compatibility of Products in the SIL Testing Tool Chain

| | |
|---|---|
| **Compatibility in general** | dSPACE recommends using only software products from the same dSPACE Release. This ensures maximum run-time compatibility. |
| **Compatibility of VEOS** | For compatibility aspects specific to VEOS, refer to Compatibility of VEOS 5.2 (VEOS Manual 🕮). |
| **Compatibility of ConfigurationDesk** | For compatibility aspects specific to ConfigurationDesk, refer to Compatibility of ConfigurationDesk 6.7 (ConfigurationDesk Real-Time Implementation Guide 🕮). |
| **Compatibility of Real-Time Testing** | For compatibility aspects specific to Real-Time Testing (RTT), refer to Compatibility with Real-Time Testing (ControlDesk Introduction and Overview 🕮). |

# SIL Testing Concepts

**Where to go from here**

Information in this section

## Basics on SIL Testing

**Introduction**

SIL testing supports an ECU development and testing process that is based on virtual ECUs and environment models.

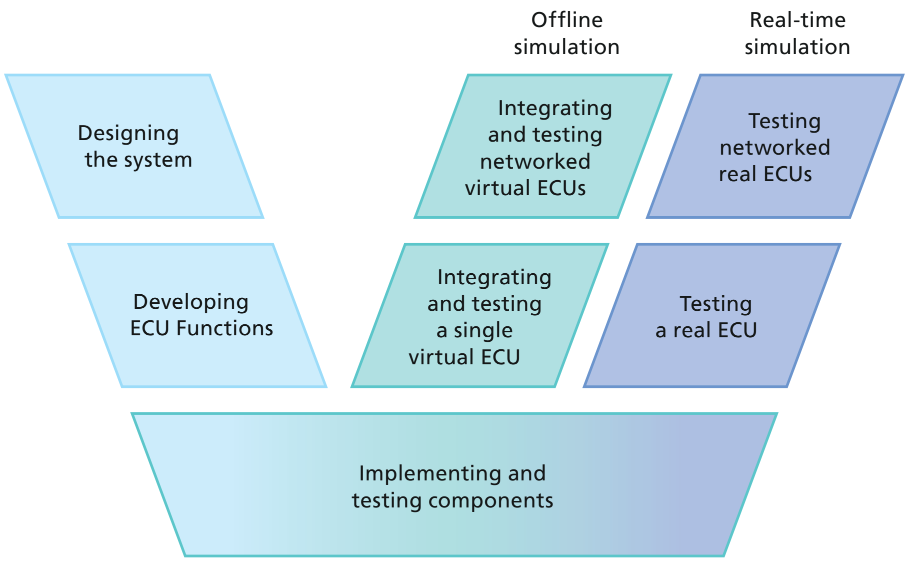
Using the *SIL testing* technology allows you to:

- Run PC-based simulations to validate, verify, and test ECU software with no other hardware
- Prepare hardware-in-the-loop (HIL) tests and scenarios on a PC
- Use V-ECUs in HIL simulation if the ECU hardware prototype is not available yet

## ECU software development with SIL testing

SIL testing lets you verify and test ECU functions and ECUs by means of virtual ECU models and models for their environment. SIL testing extends the classic V-cycle for developing ECU software, and fills the gap between specification and design on the left side of the V-cycle and integration and test on the right side.

You can use virtual ECUs (V-ECUs) in simulation scenarios throughout the entire ECU development process. Unlike a soft ECU, which uses only a simplified function model, a V-ECU usually has the same software components as those that will run on the final ECU. Thus in a simulation scenario, a V-ECU represents the real ECU more realistically than a soft ECU.


Offline simulation
Real-time simulation
Designing the system
Integrating and testing networked virtual ECUs
Testing networked real ECUs
Developing ECU Functions
Integrating and testing a single virtual ECU
Testing a real ECU
Implementing and testing components

**Performing offline and real-time simulations** Virtual ECUs and their environment models are used in simulation scenarios without or with a connection to a physical system:

- *Offline simulations* are pure PC-based simulation scenarios without a connection to a physical system. No HIL hardware and no ECU hardware prototypes are needed for performing offline simulation.
- *Real-time simulations* are simulation scenarios with a connection to a physical system. They are performed on a hardware-in-the-loop (HIL) simulator and can also comprise real ECUs.

In parallel to the specification and design phase of ECU software, offline simulation allows early systematic PC-based tests. Offline simulation is deterministic so that software problems can easily be reproduced and debugged.

Offline simulation with V-ECUs therefore results in higher ECU software quality to start the HIL-based integration and test phases with.

**Early simulation** SIL testing lets you perform early PC simulations of single and networked ECU functions, and also of how the overall system of networked V-ECUs behaves. SIL testing lets you inject errors to verify a function's or system's behavior in critical situations.

Without SIL testing, the final testing and validation of the ECU software could not begin until the prototype hardware of ECUs was available. This had clear disadvantages, as prototypes are not usually available until fairly late in the development process and are also expensive. Moreover, prototypes cannot be multiplied at will or used in several projects simultaneously. Using SIL testing, you need no ECU hardware and can frontload the HIL test phase to the virtual world, including parallel execution of huge numbers of tests in the cloud.

**Reusing models in the entire development process** dSPACE provides a comprehensive tool chain for the development process of ECUs and their virtual counterparts. SIL testing is based on the same dSPACE tools, protocols and standards as those used in a classic HIL testing environment. Thus, it is possible to reuse models, data, parameters and tests throughout the ECU development process. For example, experiment layouts from HIL simulation can be used in PC-based offline simulation and vice versa.

**Summary**

SIL testing offers the following features:

- Early verification of ECU software and functions without real hardware prototypes
- Early simulation and frontloading of tests based on virtual ECUs
- Seamless transition between SIL and HIL, e.g., for reusing SIL/HIL tests mutually
- Reuse of V-ECUs for restbus simulation in a hardware-in-the-loop simulation with a real ECU
- Using identical tools (ControlDesk, AutomationDesk, etc.)
- Using standards (AUTOSAR, XIL API, XCP, etc.)
- Parallel execution of huge numbers of tests in the cloud

# Basics on Simulation Systems

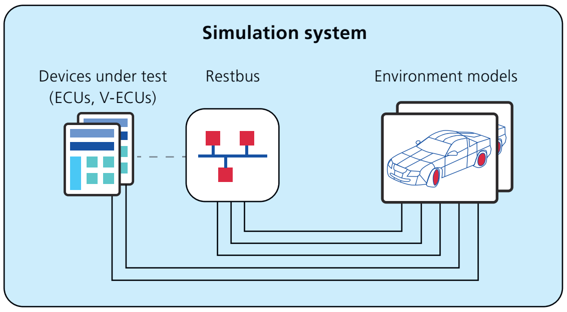
**Introduction**

SIL testing deals with *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:

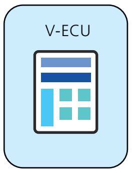
- Devices under test:
  - Virtual ECUs (V-ECUs) to be validated and tested
  - Real ECUs to be validated and tested
- Restbus
- Environment models
- Controller models
- Interconnections between these elements



You can build a platform-dependent simulation application for the simulation system to execute it on a dSPACE simulation platform.

**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 on page 14.

**Implementing a V-ECU** A V-ECU implementation (VECU) can be generated with SystemDesk and TargetLink. The generation output is a VECU file.

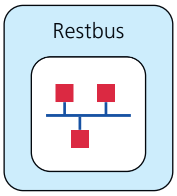
**Building a V-ECU** A V-ECU implementation is the basis for building a V-ECU. Depending on the simulation platform, a V-ECU is built with:

- VEOS on page 88 for offline simulation
- ConfigurationDesk on page 67 for real-time simulation on SCALEXIO or a MicroAutoBox III

**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.



**Implementing a bus model** The basis for a restbus is a *bus model*. An implementation of a bus model contains the platform-independent code that describes a bus model in a selected programming language.

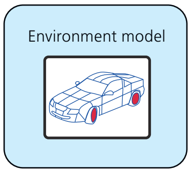
An implementation of a bus model can be generated with the Bus Manager. This results in a bus simulation container (BSC).

**Building a bus model** Depending on the simulation platform, a bus model is built with:

- VEOS for offline simulation
- Bus Manager in ConfigurationDesk for real-time simulation on SCALEXIO or a MicroAutoBox III

**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.



**Implementing an environment model** An implementation of an environment model contains the platform-independent code that describes an environment model in a selected programming language such as C.

Depending on the modeling environment, an implementation of an environment model can be generated with the following products:

- Model Interface Package for Simulink (Simulink® modeling environment)

  This results in a Simulink implementation container (SIC) file and an SDF/TRC variable description file.
- Functional Mock-up Interface (FMI)--compliant tools

  This results in a Functional Mock-up Unit (FMU) file.

**Building an environment model** Depending on the simulation platform, an environment model is built with:

- VEOS for offline simulation
- ConfigurationDesk for real-time simulation on SCALEXIO or a MicroAutoBox III

**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.

**Implementing a controller model** An implementation of a controller model contains the platform-independent code that describes a controller model in a selected programming language such as C.

An implementation of a controller model can be generated with the Model Interface Package for Simulink. This results in a Simulink implementation container (SIC) file and a related A2L variable description file.

**Building a controller model** Depending on the simulation platform, a controller model is built with:

- VEOS for offline simulation
- ConfigurationDesk for real-time simulation on SCALEXIO or a MicroAutoBox III

**Executing a simulation system**

A simulation system can be executed on a dSPACE simulation platform. You can configure and build a simulation system for:

- *Offline simulation* with VEOS

  For more information, refer to Basics on Offline Simulation with VEOS on page 28.
- *Real-time simulation* with SCALEXIO or a MicroAutoBox III

  For more information, refer to Basics on Real-Time Simulation with SCALEXIO/MicroAutoBox III on page 30.

**Related topics**

Basics

## Basics on Classic V-ECUs

**Software of classic ECUs**

Classic ECU software consists of resources, such as the operating system and network communication, that are statically configured. Therefore, in contrast to an adaptive 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 current classic ECUs also use service-oriented communication via Ethernet.

**Classic V-ECUs**

With the dSPACE SIL testing tool chain, you can create, simulate, and test the following classic V-ECUs:

- V-ECUs developed according to the AUTOSAR Classic Platform, either model-based or code-based

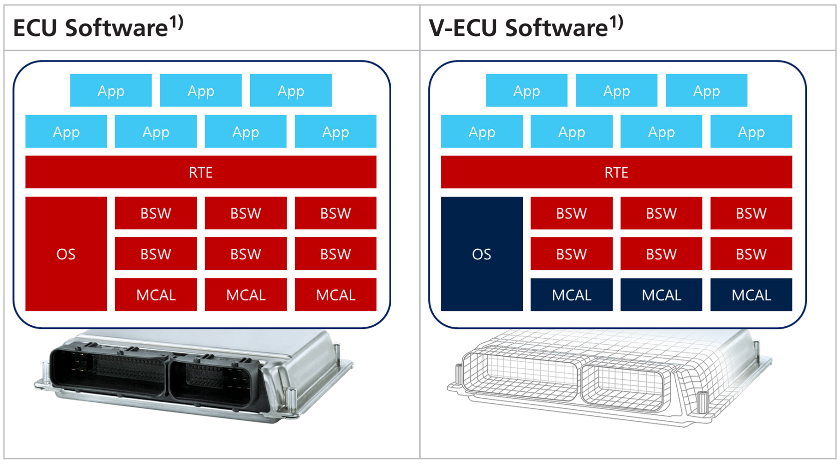
  For more information, refer to Model-Based and Code-Based Classic V-ECUs on page 17.
- V-ECUs based on non-AUTOSAR code

  For more information, refer to Integrating External Code on page 53.

**Example of virtualizing an AUTOSAR classic ECU**

To test an AUTOSAR classic ECU in an early development phase, you can create a classic V-ECU by *virtualizing the ECU*: The hardware-independent software is re-used without change in the V-ECU, whereas all the hardware-dependent software has to be replaced by software that realizes the same behavior on VEOS (PC or cloud).

The following illustrations show the software components of an AUTOSAR classic ECU, and of the related virtualized V-ECU (for an animated graphic, refer to dSPACE Help).

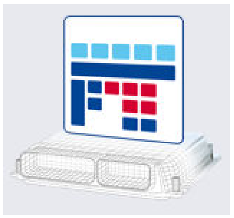
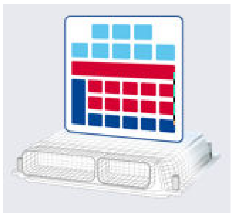


[1] 🟦 Application software 🟦 dSPACE basic software 🟥 Third-party basic software

**Levels of classic V-ECUs**

Depending on the development phase, the software of classic V-ECUs can vary widely according to the contained ECU software parts: The software may consist of only the production code of application software components or even include the entire ECU software above the low-level hardware drivers.

The following table lists the main V-ECU levels:

| V-ECU Level | Contained ECU Software | Example Test Goals |
|---|---|---|
| Application-level V-ECUs[1), 2)] | ▪ (Production-code) application software or subsets of it<br>▪ (Non-production-code) software required to simulate the application software, such as the operating system and code for signal communication.<br>An application-level V-ECU communicates at the physical signal level. It does not use bus communication. | ▪ Test the application software distributed over several components or ECUs, especially at an early development stage of the ECU, or completely independent of a specific ECU development project. |
| Simulation BSW V-ECUs[1)]  | ▪ (Production-code) application software<br>▪ Subset of the (non-production-code) basic software, such as the COM stack, the memory stack (NVRAM), or a Diagnostic Event Manager (Dem) | ▪ Test the application software that is connected to a communication bus. This allows for bus monitoring and connection to restbus models.<br>▪ Test the V-ECU within a V-ECU network interconnected by a simulated communication bus.<br>▪ Diagnostic tests |
| Production BSW V-ECUs[1)]  | ▪ (Production-code) application software<br>▪ (Production-code) basic software | ▪ Test the entire ECU behavior at a late development stage.<br>▪ Test ECU networks.<br>▪ Frontload HIL tests (refer to Bus Communication Features for SIL Testing on page 46). |

1) 🟦 Application software 🟦 dSPACE basic software 🟥 Third-party basic software

2) You can also integrate individual controller models in a simulation system. These models represent control functions of an ECU, and can be used as a first step towards an application-level V-ECU. For more information, refer to Generating Simulink Implementation Containers (Model Interface Package for Simulink - Modeling Guide ).

**Basic software for V-ECUs**

For V-ECU development and SIL testing purposes, you can use either dSPACE basic software or third-party basic software.

- Using *dSPACE basic software* lets you create V-ECUs with basic software functionality and without the need for in-depth AUTOSAR knowledge. dSPACE basic software is typically used in *application-level V-ECUs* and *simulation BSW V-ECUs*.

  For an overview of the dSPACE basic software modules, refer to Basic Software Module Support for Classic V-ECUs on page 101.
- Using *third-party basic software* lets you create V-ECUs that are very close to the production ECU, leading to a more realistic simulation behavior. Third-party basic software is typically used in *production BSW V-ECUs*.

For more information, refer to Basic Software Module Support for Classic V-ECUs on page 101.

**Related topics**

Basics

References

## Model-Based and Code-Based Classic V-ECUs

**Introduction**

The SIL testing tool chain supports model-based and code-based classic V-ECUs:

- Model-based V-ECUs: A model-based V-ECU is based on an AUTOSAR model.

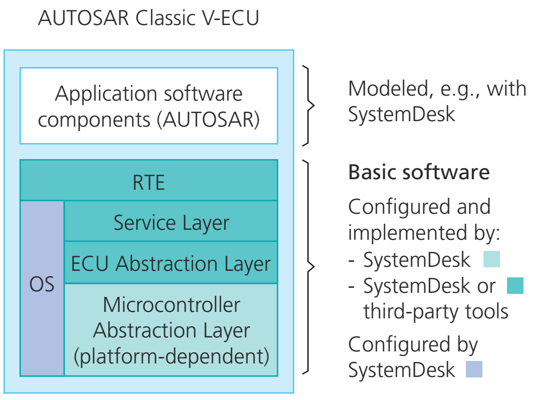
  *Application-level V-ECUs* and *simulation BSW V-ECUs* are typically model-based.
- Code-based V-ECUs: A code-based V-ECU is based on external AUTOSAR or non-AUTOSAR code.

  *Production BSW V-ECUs* are typically code-based.

**Model-based V-ECU implementation**

A *model-based* classic V-ECU implementation is based on an ECU in an AUTOSAR system that is modeled in or imported to SystemDesk.

The following illustration shows a model-based classic V-ECU:


AUTOSAR Classic V-ECU
Application software components (AUTOSAR)
RTE
Service Layer
ECU Abstraction Layer
OS
Microcontroller Abstraction Layer (platform-dependent)
Modeled, e.g., with SystemDesk
Basic software
Configured and implemented by:
- SystemDesk
- SystemDesk or third-party tools
Configured by SystemDesk

**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.

**Basic software** The generic term for the following software that is part of the ECU software according to the AUTOSAR Classic Platform:

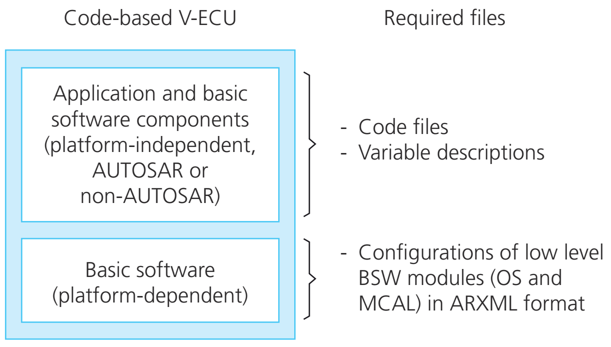
- Operating system (OS)
- AUTOSAR Services
- Communication stack
- I/O stack (ECU abstraction and microcontroller abstraction)
- Memory stack
- Complex device driver

Together with the run-time environment, the basic software is the platform for the ECU application software.

## Code-based V-ECU implementation

A *code-based* classic V-ECU implementation is based on external AUTOSAR or non-AUTOSAR code.

The following illustration shows a code-based classic V-ECU:


Code-based V-ECU
Required files
Application and basic software components (platform-independent, AUTOSAR or non-AUTOSAR)
Basic software (platform-dependent)
- Code files
- Variable descriptions
- Configurations of low level BSW modules (OS and MCAL) in ARXML format

For more information on how to create a code-based virtual ECU that uses the microcontroller abstraction layer (MCAL) as the interface between the production code and the simulator, refer to https://www.dspace.com/go/CodeBasedVECU.

## Related topics

Basics

References

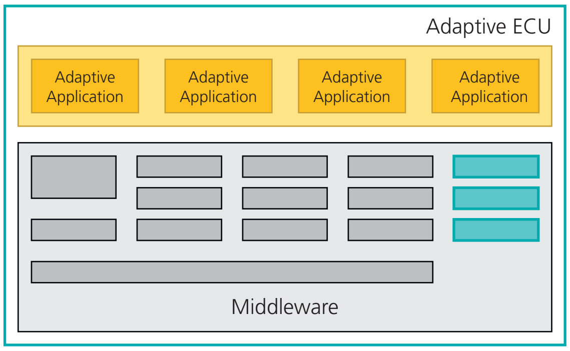
## Basics on Adaptive V-ECUs

**Software of adaptive ECUs**

Adaptive ECU software usually handles complex tasks, such as tasks for autonomous driving. Adaptive ECU software also allows for the *dynamic* integration of new applications and changes to the network communication at run time, which is not possible with *statically* configured classic ECU software. Communication is service-oriented via Ethernet.

An adaptive ECU according to the AUTOSAR Adaptive Platform consists of the *AUTOSAR Runtime for Adaptive Applications (ARA)* (also labeled as the *middleware*), and adaptive applications that run on it. Since the middleware is configured dynamically, it enables flexible updating and exchanging of applications during run time.

The following illustration shows adaptive ECU software according to the AUTOSAR Adaptive Platform.



In general, the middleware is provided as an image to be integrated into the adaptive ECU. The adaptive applications are built with the related middleware software development kit (SDK) and added as independent layers. Because the middleware is independent of the adaptive applications, you can add or exchange adaptive applications as required.

**Adaptive Platform Demonstrator** If you are member of the AUTOSAR Development Partnership, you have access to the Adaptive Platform Demonstrator. The Adaptive Platform Demonstrator provides a reference implementation of the AUTOSAR Adaptive Platform middleware and a related SDK for building adaptive applications.

**Adaptive V-ECUs**

With the dSPACE SIL testing tool chain, you can create and simulate adaptive V-ECUs:

- V-ECUs developed according to the AUTOSAR Adaptive Platform.

  The dSPACE SIL testing tool chain lets you integrate a middleware into an adaptive V-ECU and compile adaptive applications using the SDK related to the middleware.

- Non-AUTOSAR Linux-based V-ECUs

  VEOS executes adaptive V-ECUs based on a Linux kernel provided in a virtual machine. You can use this virtual machine also to execute any other non-AUTOSAR Linux-based V-ECU.

**Creating adaptive V-ECUs** You can create adaptive V-ECUs in the following ways:

- From the source files of the adaptive applications.

  In this case, the Adaptive Platform Demonstrator provided by AUTOSAR can be used as the middleware. SystemDesk lets you integrate and create the adaptive V-ECU.
- From the source files of the adaptive applications, your own middleware, and the related SDK.

  SystemDesk lets you integrate and create the adaptive V-ECU.
- From an image that includes both the adaptive applications and the middleware.
- From separate images of the middleware and the adaptive applications.

For more information, refer to Creating Adaptive V-ECUs (SystemDesk Manual 🕮).

**Simulating adaptive V-ECUs** During offline simulation on VEOS, the adaptive V-ECU runs as a Linux virtual machine using a specific simulation Linux kernel.

For more information, refer to Basics on Simulating Adaptive V-ECUs (VEOS Manual 🕮).

**Note**

The dSPACE SIL testing tool chain does not support white-box testing of adaptive V-ECUs. However, you can use third-party diagnostic tools as an alternative. Contact dSPACE Support for further options.

**Related topics**

Basics

Basics on Simulating Adaptive V-ECUs (VEOS Manual 🕮)
Creating Adaptive V-ECUs (SystemDesk Manual 🕮)

# Tool Chain and Workflows for SIL Testing

**Where to go from here**

Information in this section

## Tool Chain and Workflow for SIL Testing

**Introduction**

dSPACE provides a tool chain and workflow that enables the SIL testing and verification of ECU software and environment models.

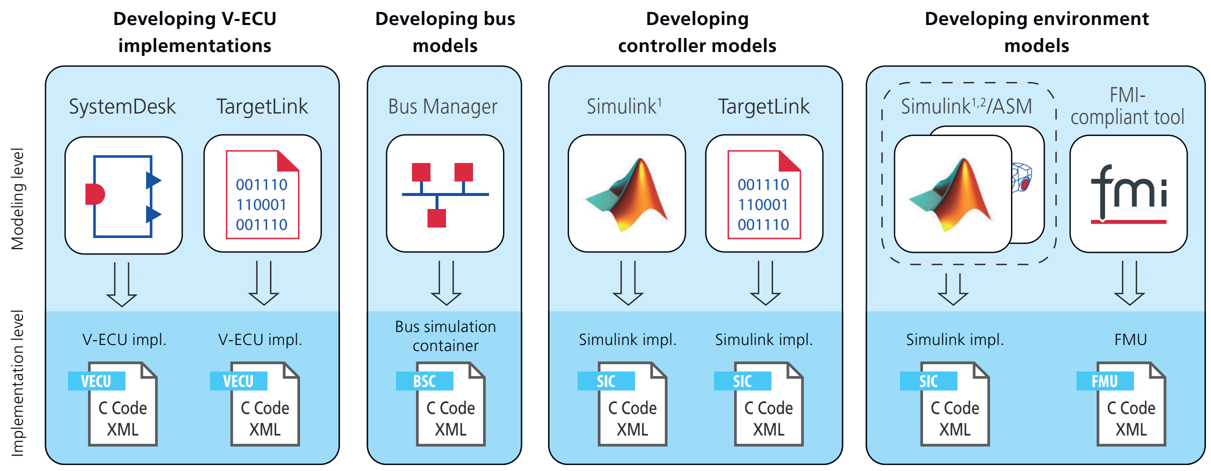
**Overview**

The SIL testing workflow can be divided into the following steps:

1. Developing the simulation system
2. Integrating and building the simulation system
3. Experimenting and testing with the simulation system

## Developing the simulation system

The first step in the tool chain is to develop the simulation system as shown in the following illustration.



[1] With Model Interface Package for Simulink
[2] Simulink models can also be added directly to a ConfigurationDesk application

The following table lists the workflow steps and the software products involved.

| Workflow Step | Involved Software Products |
|---|---|
| Developing V-ECU implementations | ▪ SystemDesk (refer to SystemDesk on page 83)[1)]<br>▪ TargetLink (refer to TargetLink on page 84)[2)] |
| Developing bus models | Bus Manager (refer to Bus Manager on page 65) |
| Developing controller models | ▪ Model Interface Package for Simulink (refer to Model Interface Package for Simulink on page 73)[3)]<br>▪ TargetLink (refer to TargetLink on page 84)[3)] |
| Developing environment models | ▪ Simulink® modeling environment:<br>  ▪ Automotive Simulation Models (refer to Automotive Simulation Models and ModelDesk on page 63)<br>  ▪ Model Interface Package for Simulink (refer to Model Interface Package for Simulink on page 73)[4)]<br>▪ FMI modeling environment:<br>  ▪ FMI-compliant tools |

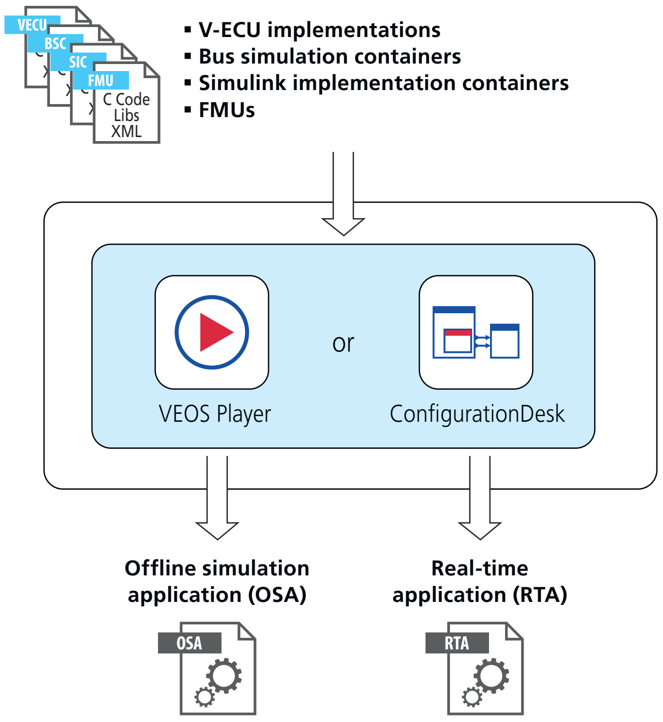
1) For generating V-ECU implementations and V-ECUs.
2) For generating V-ECU implementations and V-ECUs based on an AUTOSAR or non-AUTOSAR TargetLink subsystem.
3) For generating Simulink implementation containers and related A2L variable description files.
4) For generating Simulink implementation containers and related SDF/TRC variable description files.

**Tip**

If the integration of model implementations is not possible, an alternative is to couple VEOS with a third-party simulator, such as an instruction set simulator.
For more information, refer to Co-Simulation with VEOS on page 42.

**Integrating and building the simulation system**

The second step in the tool chain is to integrate and build the simulation system for a specific simulation platform as shown in the following illustration.

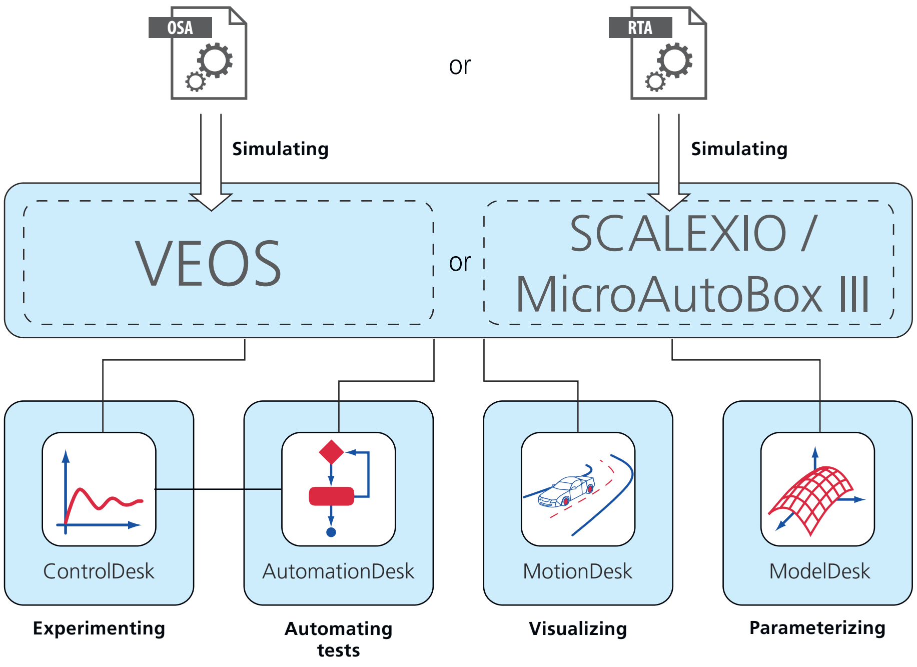

VECU
BSC
SIC
FMU
C Code
Libs
XML
▪ V-ECU implementations
▪ Bus simulation containers
▪ Simulink implementation containers
▪ FMUs
VEOS Player
or
ConfigurationDesk
Offline simulation application (OSA)
Real-time application (RTA)
OSA
RTA

The following table lists the workflow steps and the software products involved.

| Workflow Step | Involved Software Products |
|---|---|
| Integrating the simulation system | |
| For offline simulation on VEOS | VEOS on page 88<br>(optionally: SYNECT (refer to SYNECT on page 81)) |
| For real-time simulation on SCALEXIO or a MicroAutoBox III | ConfigurationDesk on page 67 |

## Experimenting and testing with the simulation system

The third step in the tool chain is to experiment and test with the simulation system on a specific simulation platform as shown in the following illustration.



The following table lists the workflow steps and the software products involved.

| Workflow Step | Involved Software Products |
|---|---|
| Experimenting with ECUs and V-ECUs | ControlDesk (refer to ControlDesk on page 69) |
| Visualizing experiments | MotionDesk (refer to MotionDesk on page 76) |
| Parameterizing Automotive Simulation Models (ASMs) | ModelDesk (refer to Automotive Simulation Models and ModelDesk on page 63) |
| Automating tests | ▪ AutomationDesk (refer to AutomationDesk on page 60)<br>▪ Real-Time Testing (refer to Real-Time Testing on page 77)[1)]<br>▪ dSPACE XIL API Implementations |

1) For measuring and stimulating test data in real-time, i.e., synchronously to the running application.

## Related topics

Basics

## Additional Workflow Steps for Rapid Prototyping Access to V-ECUs

**Introduction**

When you want to get rapid prototyping access to V-ECU-internal variables and functions in a SIL testing scenario, some additional workflow steps come into play with regard to the basic workflow.

**Additional steps when developing the simulation system**

When you develop the simulation system and want to get rapid prototyping access to V-ECU-internal variables and functions, you have to prepare runnables for being accessed by configuring the Rapid Prototyping Access (RptAccess) module.

The following table lists the additional workflow steps and the software products involved.

| Additional Workflow Step | Involved Software Product(s) |
|---|---|
| Configuring the V-ECU basic software using the Rapid Prototyping Access (RptAccess) module | SystemDesk 1) |

1) In addition, the Rapid Prototyping Access (RptAccess) module is involved.

**Additional steps when integrating and building the simulation system**

When you integrate and build the simulation system and want to access V-ECU-internal variables and functions, you have to perform some additional workflow steps that depend on the simulation platform.

The following table lists the additional workflow steps and the software products involved.

| Additional Workflow Step | Involved Software Product(s) |
|---|---|
| Integrating a Simulink or TargetLink model dynamically into a V-ECU:<br>▪ For offline simulation on VEOS<br>▪ For real-time simulation on a SCALEXIO System or on a MicroAutoBox III | RTI Bypass Blockset |

**Additional steps when experimenting and testing with the simulation system**

When experimenting and testing with the simulation system, there are no additional workflow steps for accessing V-ECU-internal variables and functions.

**Related topics**

Basics

# Simulation Platforms for SIL Testing

**Where to go from here**

Information in this section

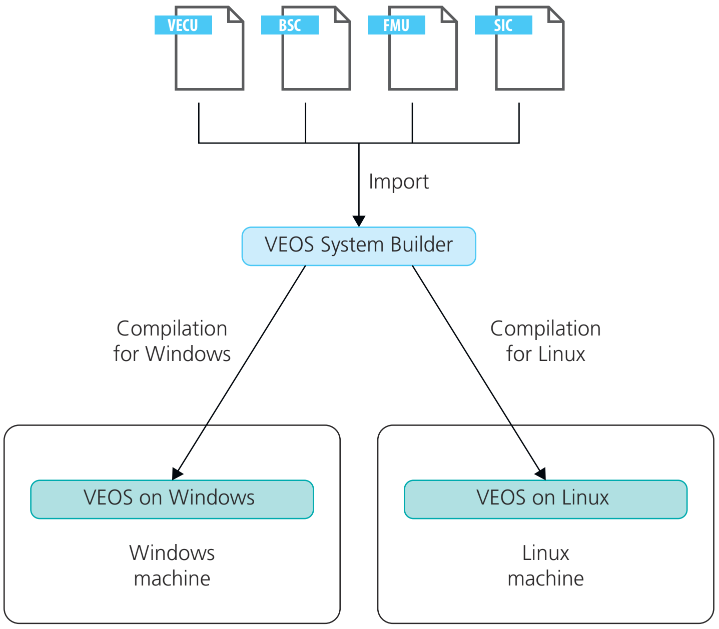
## Basics on Offline Simulation with VEOS

**Introduction**

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

**Running an offline simulation on Windows or Linux**

You can run an offline simulation application with VEOS on Windows or on Linux by selecting the corresponding compiler.



Running an offline simulation on Linux enables you, for example, to perform automated tests on a Linux-based cluster. For more information, refer to Cluster Simulation for Automated Test Drives on page 38.

For more information on running an offline simulation application on VEOS on Linux, refer to Introduction to VEOS on Linux (VEOS Manual 🕮).

**Running an offline simulation on a local or remote simulator**

You can run an offline simulation application on a local simulator or on a remote simulator.

For more information, refer to Simulation on a Local or Remote Simulator (VEOS Manual 🕮).

**Support of 64- and 32-bit applications**

**Building and simulating 64- and 32-bit applications** Depending on the characteristics of your code, VEOS lets you build 64- or 32-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- as well as 32-bit libraries.

**Stimulating, measuring, and calibrating variables of 64- and 32-bit applications** You can stimulate, measure, and calibrate variables of 64- and 32-bit applications executed on VEOS using dSPACE experiment software such as ControlDesk.

**Limitations** Rapid prototyping access to V-ECU internal variables and functions, i.e., virtual bypassing, is not supported for 64-bit V-ECUs.

**Bus communication simulation**

In offline simulation on VEOS, bus communication is simulated on the PC.

For more information, refer to Bus Communication Features for SIL Testing on page 46.

**Blocking and non-blocking communication between VPUs**

VEOS supports *blocking* and *non-blocking* communication between interconnected VPUs.

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.

For details, refer to Specifying Blocking or Non-Blocking Communication (VEOS Manual).

**Code debugging**

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 56.

**Related topics**

Basics

Cluster Simulation for Automated Test Drives........................................................................ 38

Simulation Architecture of VEOS (VEOS Manual 🕮)
Timing in Offline Simulation with VEOS (VEOS Manual 🕮)
VEOS........................................................................................................................................88

HowTos

How to Load and Run an Offline Simulation Application (VEOS Manual 🕮)

## Basics on Real-Time Simulation with SCALEXIO/MicroAutoBox III

**Introduction**

SCALEXIO systems are simulators for hardware-in-the-loop (HIL) simulation. They are used to test electronic control units (ECUs).

The MicroAutoBox III is a rapid control prototyping (RCP) system for in-vehicle use. The MicroAutoBox III can be used in fullpass and bypass scenarios for different RCP applications.

**Note**

One core of the real-time hardware is reserved for executing system services. Therefore, the number of V-ECU implementations and environment models is limited to the number of processor cores minus one. For example, you can use only seven V-ECUs and/or environment models for simulation on a SCALEXIO Processing Unit that has a main processor with eight cores.

**Real I/O connections**

**SCALEXIO** With a hardware-in-the-loop (HIL) simulation on a SCALEXIO system, functions and diagnostics of ECUs can be tested quickly and automatically. One or more real ECUs are usually connected to a model of the plant and the ECUs' restbus for these tests. The plant and the ECUs' restbus are simulated on the SCALEXIO system in real time. Thus, the correct behavior of the ECU can be tested even in critical situations in which the real plant might be damaged.

**MicroAutoBox III** The MicroAutoBox III is a real-time system for performing fast, in-vehicle function prototyping. You can execute controller models or V-ECUs to experience and test control functionalities in a real environment. MicroAutoBox III is ideal for many different rapid control prototyping (RCP) applications.

**Timing effects in real-time simulation with SCALEXIO or MicroAutoBox III**

**Correct real-time timing behavior** Real-time simulation is important for HIL tests and RCP. The computing power required by real-time simulation highly depends on the characteristics of the simulated model: If it contains very demanding calculations, a lot of computing power has to be provided because the correct timing cannot be satisfied otherwise. SCALEXIO and MicroAutoBox III systems fulfill this demand for computing power.

**V-ECUs and environment model application processes: Task scheduling** In real-time simulation on SCALEXIO and a MicroAutoBox III, the clocks of V-ECUs and environment model application processes are not exactly synchronized. The operating system of the dSPACE real-time hardware might start the execution of V-ECU tasks and environment model tasks at slightly different times, depending on the initialization time.

As a consequence, the periodic tasks of V-ECUs and environment model application processes are not triggered synchronously. This effect is similar to real ECUs, which have their own local clocks. The resulting effects can be observed in a ControlDesk experiment when plotting signals from different V-ECUs and/or environment model application processes with a common time axis.

**Blocking and non-blocking communication** Real-time simulations with SCALEXIO or MicroAutoBox III support *blocking* and *non-blocking* communication of signals exchanged between different models. In ConfigurationDesk, signal data is exchanged between models via model port blocks.

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.

For details, refer to Basics on Model Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

**Related topics**

Basics

# Use Cases for SIL Testing

**Where to go from here**

Information in this section

## Early Validation and Verification of Controller Functions

**Introduction**

You can validate and verify controller functions as early as possible.

**Developing single controller functions**

Function developers want early validation and verification of the controller functions they are developing. This is why they are using PC-based simulations as early as possible in the development process.

**Using the entire dSPACE experimenting and testing tool chain** Developers performing the model-based development of control functions with Simulink® and TargetLink can use VEOS, dSPACE's platform for PC-based offline simulation. Simulating on the VEOS platform enables them to validate and verify the function C code using the entire dSPACE experimenting and testing tool chain. For example, they can experiment with functions developed in TargetLink using ControlDesk.

**Controller function integration** The controller function to be validated and verified can be integrated into the overall ECU software.

**Support for controller functions distributed over several ECUs** Controller functions are often distributed over several ECUs. Simulating on the VEOS platform supports this by simulating the signal communication and bus communication between the ECUs.

**Summary**

- The controller function C code to be validated and verified runs on a PC, so no specific simulation hardware is required.
- The entire dSPACE experimenting and testing tool chain can be used for validation and verification.
- Open-loop or closed-loop simulation can be used to find errors earlier in the development process.
- AUTOSAR controller functions developed with TargetLink can be integrated into the software architecture of an ECU network.
- Testers can use the same models, layouts and simulation scenarios as for the HIL simulation later in the development process.

## Simulating on a Virtual Test Bench

**Introduction**

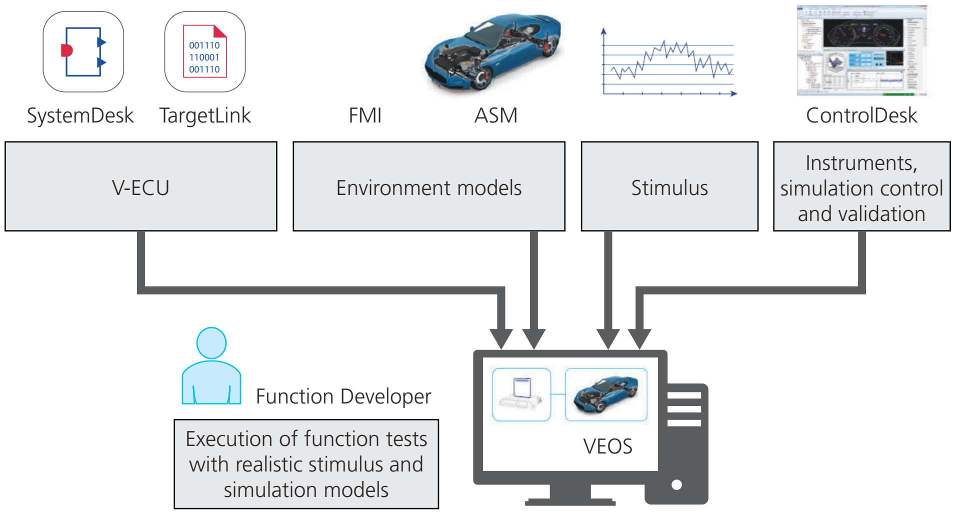
You can design and test a new controller function, for example, for an engine ECU. Because mechanical test benches are expensive, and an engine prototype is often not yet available, a virtual test bench is the ideal solution to test the new engine controller function via SIL testing.

## Virtual test bench simulation with ASM models

dSPACE Automotive Simulation Models (ASMs) offer various fully configurable models that support a wide spectrum of simulations, starting with individual components like combustion engines or electric motors, to vehicle dynamics systems, up to complex virtual traffic scenarios. These can be used as environment models for verifying the controller early in the development process, replacing the mechanical test bench by a virtual one.

The V-ECU containing the controller function and the environment models are interconnected and simulated on VEOS. The simulation is configured and recorded in ControlDesk. Photorealistic layouts create the feeling of a real test bench to make the change to a virtual test bench easy.

The following illustration shows the use case:


001110
110001
001110
SystemDesk
TargetLink
FMI
ASM
ControlDesk
V-ECU
Environment models
Stimulus
Instruments, simulation control and validation
Function Developer
Execution of function tests with realistic stimulus and simulation models
VEOS

## Summary

- The software to be tested runs on a PC, so no hardware, i.e., no real test bench, is required for simulation on a virtual test bench.
- The virtual test bench is always available.
- The controller function and engine model are separated with defined interfaces to enable fast function updates.
- Testers can use the same models, layouts, and simulation scenarios as for the HIL simulation later in the process.
- Multiple virtual test benches can be executed parallel in the cloud.

## Related topics

Basics

Automotive Simulation Models and ModelDesk........................................................................ 63

## Integration and Validation of V-ECUs

**Introduction**

You can integrate AUTOSAR application software components (SWCs) and basic software components (BSWCs) into one or more virtual ECUs (V-ECUs) to validate their functional interaction.

**Validating and testing the overall behavior of ECU software**

The single AUTOSAR application software components were tested and implemented by a function developer at the beginning of an ECU's development process. The task of a software integrator or system architect is now to integrate these components in one or more virtual ECUs so that their functional interaction can be tested in connection with ECU basic software components and network communication. This can be done by means of SystemDesk, dSPACE's system architecture software. SystemDesk is used to model the ECU architectures including their bus communication and to integrate the SWCs and BSWCs that were implemented. When system modeling is finished, SystemDesk lets you generate one or more virtual ECUs to be tested in an offline simulation.

Virtual ECUs can be tested in both open-loop and closed-loop simulation. For closed-loop simulation, the V-ECUs are integrated with environment models representing the ECUs' environment. The environment can be modeled in Simulink® by using dSPACE's Automotive Simulation Models (ASMs), or in an FMI-compliant tool.

For SIL testing, the behavior of the V-ECUs and the environment is then simulated with VEOS, SCALEXIO, or the MicroAutoBox III.

The simulation is controlled and visualized using ControlDesk. V-ECU variables can be parameterized and measured exactly as they would be in simulation scenarios with real ECUs.

**Summary**

- Test the integration of ECU software before the first ECU prototype is available
- Simulate the interaction between application software components and basic software modules
- Simulate bus communication between V-ECUs
- Perform open-loop or closed-loop simulations to find errors earlier in the development process
- Easily integrate environment models
- Handle AUTOSAR systems

## Frontloading Functional HIL Tests

**Introduction**

You can frontload functional ECU tests from hardware-in-the-loop (HIL) simulation to offline simulation and to prepare a HIL simulation as early as possible.

## Using V-ECUs to frontload HIL tests

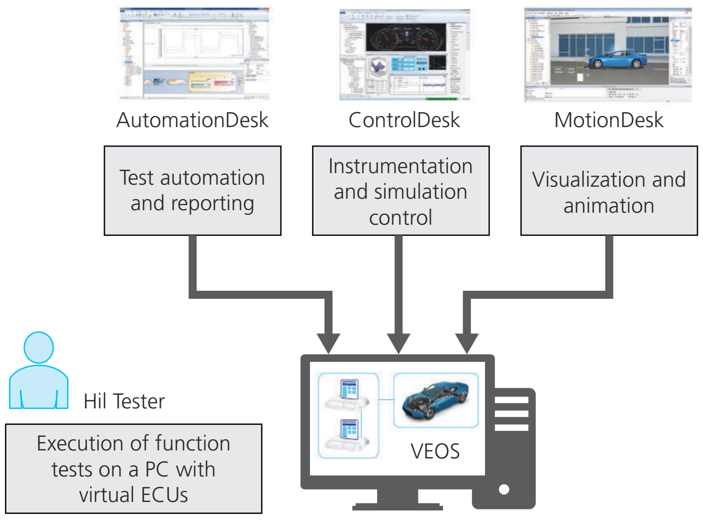
Classic hardware-in-the-loop (HIL) scenarios for testing ECU software require real prototypes of the ECUs as devices under test. That is why HIL simulation takes place late in the development process of ECU software. This also involves the preparation of HIL tests, which cannot start before the simulator hardware is available. Using SIL testing, you can frontload functional HIL tests.

In the first step, all the ECUs are virtual. The virtual ECUs (V-ECUs) can be simulated in conjunction with a real-time model of their environment in a PC-based offline simulation with VEOS. No additional hardware is needed. You can use offline simulation with VEOS to validate your ECU software and set up your tests and environment model at an early stage. SIL testing with VEOS is based on the same dSPACE tools, protocols and standards that are used in a classic HIL scenario. You can reuse instrumentation layouts, data sets, measurement data, and test scripts developed for the V-ECUs later in the HIL simulation when using the related real ECUs.

In a next step, you can change your simulation platform from VEOS to a SCALEXIO system and replace one or more V-ECUs of your (V-)ECU network by real ECUs. The missing ECUs are still represented by V-ECUs and executed together with the ECUs' real-time environment on the SCALEXIO system. The V-ECUs represent their real counterparts far better than the soft ECUs used in classic HIL scenarios and allow more realistic restbus simulation.

Finally, you can switch to a classic HIL scenario by replacing all the V-ECUs with real ECUs. You can (re)use an optimized simulation environment at this stage.

The following illustration shows the use case:


AutomationDesk
ControlDesk
MotionDesk
Test automation and reporting
Instrumentation and simulation control
Visualization and animation
Hil Tester
Execution of function tests on a PC with virtual ECUs
VEOS

## Summary

- Perform early setup and validation of models, layouts, test libraries, and test implementations on a standard PC
- Exactly reproducible test scenarios
- Perform test migration on a standard PC
- Find development errors earlier to save time and costs

- (Re)use V-ECUs to frontload your HIL tests
- Reuse test cases between different development phases

## Cluster Simulation for Automated Test Drives

**Introduction**

You can test automated driving functions or self-driving cars, which involve millions of kilometers of test drives.

**Driving millions of kilometers on PCs**

Test drives for automated driving functions or self-driving cars must cover a broad range of roads, weather conditions, and traffic scenarios. Using only hardware-in-the-loop (HIL) simulators for these tests requires a vast number of HIL systems and real ECU prototypes.

As a PC-based simulation platform, VEOS can handle high numbers of simulation requests.

Tests with VEOS:

- Can be executed completely automated by using the automation interface of the VEOS Player.
- Can be executed faster than in real time (depending on the size of the model to be simulated).

**Cluster simulation** For the parallel execution of virtual test drives, you can set up a cluster with as many PCs on which VEOS is installed as needed. Depending on the number of VEOS installations used, you can complete millions of test drive kilometers each day. Each cluster node hosts one VEOS installation that runs a simulation system with a number of V-ECUs and environment VPUs. The cluster is controlled by one central unit that schedules the execution of the simulation jobs and test cases.

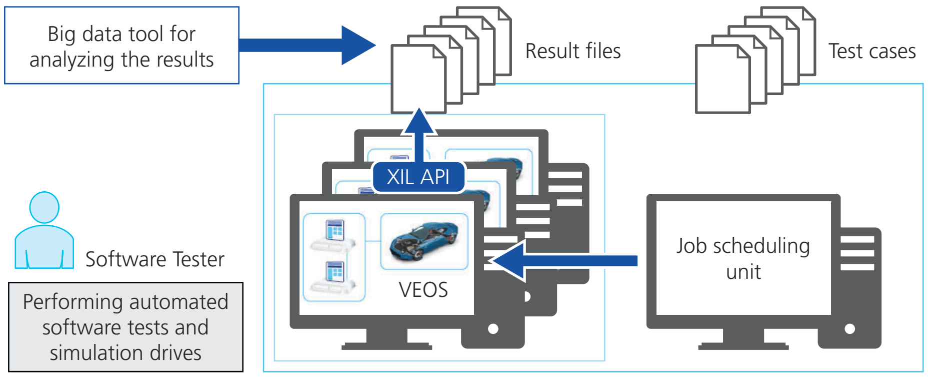
Any test drive that fails can also be reproduced and debugged in detail using VEOS at a developer workplace using VEOS.

**Note**

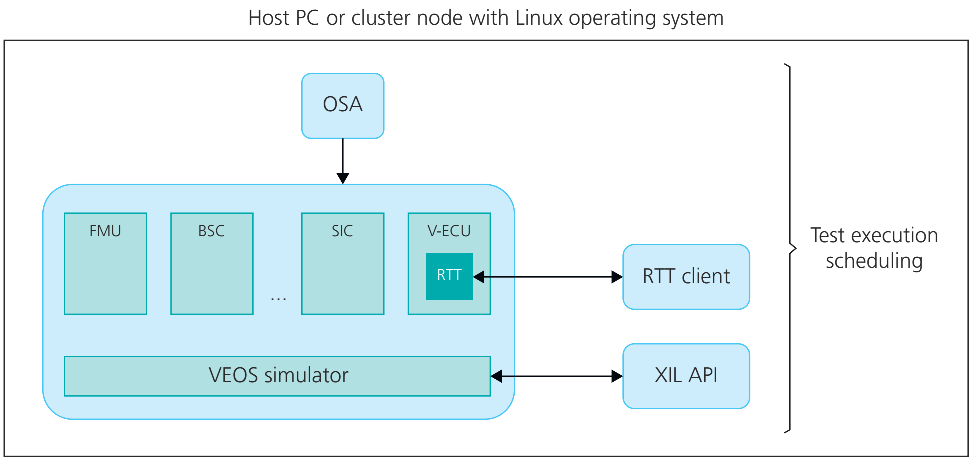
For information on how to schedule offline simulations on VEOS in a PC cluster, contact dSPACE Support.

**Accessing and recording simulation data** Simulation data generated by VEOS can be accessed and recorded using the XIL API. The data can be transferred to a central storage device and can be analyzed later, for example.

The following illustration shows the use case:


Big data tool for analyzing the results
Result files
Test cases
XIL API
Software Tester
VEOS
Job scheduling unit
Performing automated software tests and simulation drives

The following illustration shows the use case when the offline simulation is performed by the VEOS simulator running on a Linux machine, i.e., on a host PC or cluster node with Linux operating system:


Host PC or cluster node with Linux operating system
OSA
FMU
BSC
...
SIC
V-ECU
RTT
RTT client
VEOS simulator
XIL API
Test execution scheduling

**Summary**

- Comprehensive tests at an early development stage
- Highly scalable due to virtual ECUs
- Deterministic and reproducible test execution
- High test throughput through fast test execution

**Related topics**

Basics

Basics on Offline Simulation with VEOS .......... 28

## Rapid Prototyping Access to V-ECU-Internal Variables and Functions

**Introduction**

You can get rapid prototyping access to V-ECU-internal variables and functions.

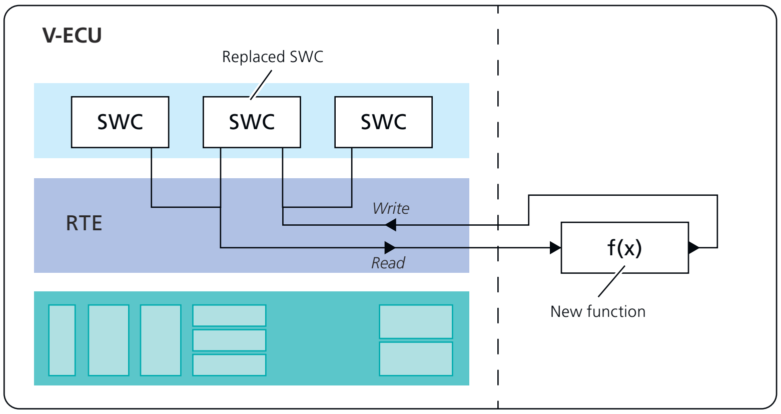
For example, for:

- Replacing an existing ECU function with a new one, and testing the new ECU function as early as possible. See Bypassing an existing ECU function in a virtual environment on page 40.
- Synchronizing V-ECU application tests with the value of a V-ECU-internal status variable. See Synchronizing test functions with V-ECU-internal states on page 41.
- Connecting V-ECU-internal variables to the I/O of MicroAutoBox III. See Connecting V-ECU-internal variables to the hardware of a MicroAutoBox III on page 41.

**Bypassing an existing ECU function in a virtual environment**

You can bypass an existing ECU function, i.e., replace the function with a new one, and test the new ECU function as early as possible.

The following illustration shows an example:



When you have a virtual ECU (V-ECU), you can develop new ECU functions in Simulink® or in TargetLink and integrate them dynamically in the V-ECU for offline or real-time simulation. This allows you to develop and test the new ECU functions using the *internal bypassing method* and without having to modify the V-ECU software architecture.

Depending on your simulation platform, refer to:

- VEOS: Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS (RTI Bypass Blockset Reference 📖)
- SCALEXIO or MicroAutoBox III: Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III (RTI Bypass Blockset Reference 📖)

**Preparation for bypassing with real ECUs** Function bypassing in a virtual environment is a preparatory step towards bypassing with real ECUs.

When you switch from bypassing on a V-ECU to bypassing on a real ECU, you can:

- Execute the recompiled model code directly on the real ECU by using the *internal bypass method*

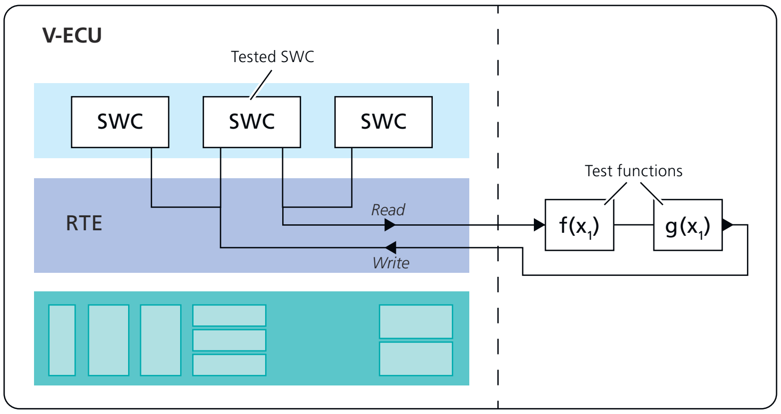
  or:
- Execute the recompiled model code on an external rapid control prototyping (RCP) system, such as MicroAutoBox III, which is connected to the real ECU via ECU interface hardware such as the DCI-GSI2. This is the *external bypassing method*.

**Synchronizing test functions with V-ECU-internal states**

You can synchronize the execution of functions used to test the V-ECU application with V-ECU-internal events (*white-box testing*).

White-box testing is an extension to conventional (black-box) HIL testing in which the ECU is stimulated via its physical I/O. It allows you to test specific functions in the V-ECU application without having to stimulate the V-ECU's I/O.

The following illustration shows an example:



Once you have a virtual ECU (V-ECU), you can develop test functions in Simulink® or in TargetLink and integrate them dynamically in the V-ECU. This allows you to test the ECU functions in a virtual environment, using the *internal bypassing method* and without having to modify the V-ECU software architecture.

**Preparation for testing with real ECU and real HIL simulator** White-box testing in a virtual environment is a preparatory step towards white-box testing with real ECUs: You can replace a V-ECU with a real ECU to be tested, and reuse the Simulink or TargetLink model containing the test function(s) on the HIL simulator. The HIL simulator accesses the ECU via the *external bypassing method*.

**Connecting V-ECU-internal variables to the hardware of a MicroAutoBox III**

You can synchronize the access to V-ECU-internal variables with code executions that involve hardware features of the MicroAutoBox III.

**Summary**

- Access V-ECU-internal variables and functions
- Test functions by reading and writing function input and output values; tests can be synchronized by triggering test executions depending on V-ECU-internal states
- Implement connections to an environment model by overlaying I/O functions of the V-ECU
- Implement and test new control algorithms by overlaying (V-) ECU functions (bypassing)

**Related topics**

Basics

Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III (RTI Bypass Blockset Reference)
Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS (RTI Bypass Blockset Reference)

## Co-Simulation with VEOS

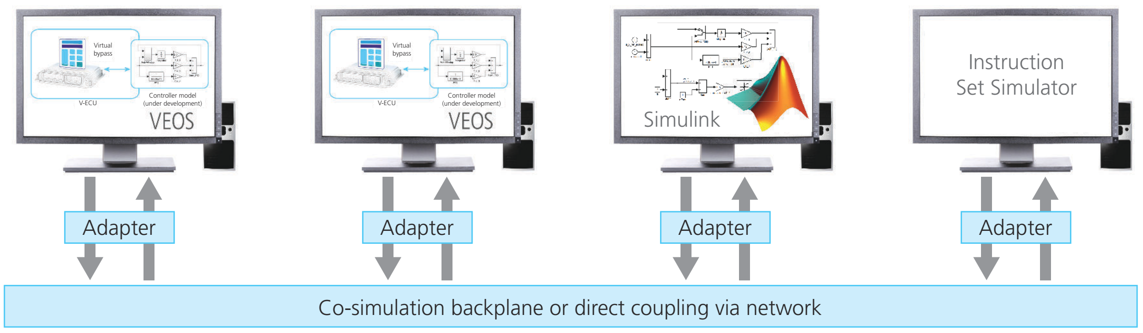
**Introduction**

You can couple the offline simulation runing on VEOS with domain-specific simulators.

**Coupling simulators**

The dSPACE SIL testing tool chain provides several options to integrate models into a system for offline simulation on VEOS. These options are based on the *integration of SIC, BSC, V-ECU, FMU containers with generated C/C++ code*.

However, if the integration of model code is not possible, an alternative is to couple VEOS with a third-party simulator, such as Simulink®, or an instruction set simulator.


Virtual bypass
V-ECU
Controller model (under development)
VEOS
Virtual bypass
V-ECU
Controller model (under development)
VEOS
Simulink
Instruction Set Simulator
Adapter
Adapter
Adapter
Adapter
Co-simulation backplane or direct coupling via network

The different simulators can be coupled either directly via the TCP/IP network, or via a co-simulation backplane that coordinates the simulator interplay.

Contact dSPACE Support for a coupling solution for your specific simulator.

**Summary**

- Perform closed-loop simulations using existing models in domain-specific simulators.
- Increase the computation power by distributing a system model on multiple PCs.
- Couple data replay tools and realistic sensor signal simulation.

# Features for SIL Testing

**Where to go from here**

Information in this section

## Bus Communication Features for SIL Testing

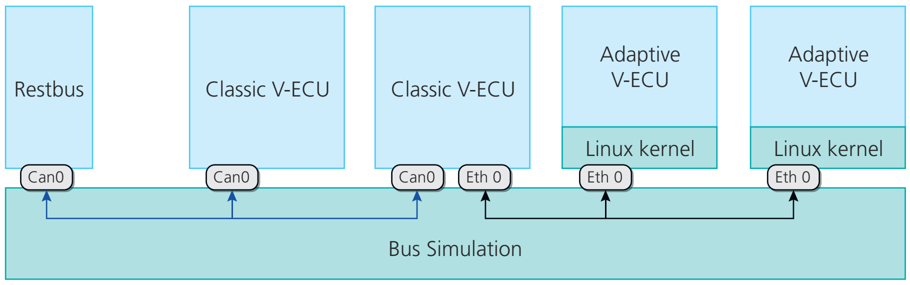
**Introduction**

In offline simulation on VEOS, bus communication is simulated on the PC.

In real-time simulation on SCALEXIO or MicroAutoBox III, the bus hardware of the real-time hardware is used for bus communication.

**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:

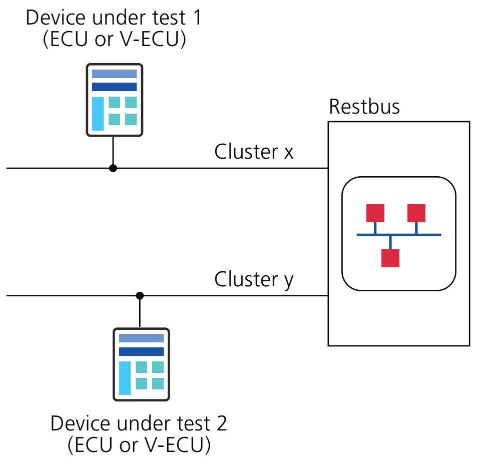
- Support of the AUTOSAR COM and PDU Router behavior for any bus type
- Cyclic and event-controlled transmission of IPDUs. Mixed transmission mode is supported for periods greater than zero.
- Delayed start-up of bus communication if V-ECU start-up is delayed.

- 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*.

  SCALEXIO and MicroAutoBox III: *message transmission times* depend on, for example, the hardware used for bus communication and on the bus configuration (baud rate).
- 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 (VEOS Manual 🕮).

**Related topics**

Basics

Limitations........................................................................................................ 112
Restbus Simulation in VEOS (VEOS Manual 🕮)
Workflow for Simulating Bus Communication (VEOS Manual 🕮)

References

Basic Software Module Support for Classic V-ECUs.................................................. 101

## Real-Time Simulation with SCALEXIO or MicroAutoBox III

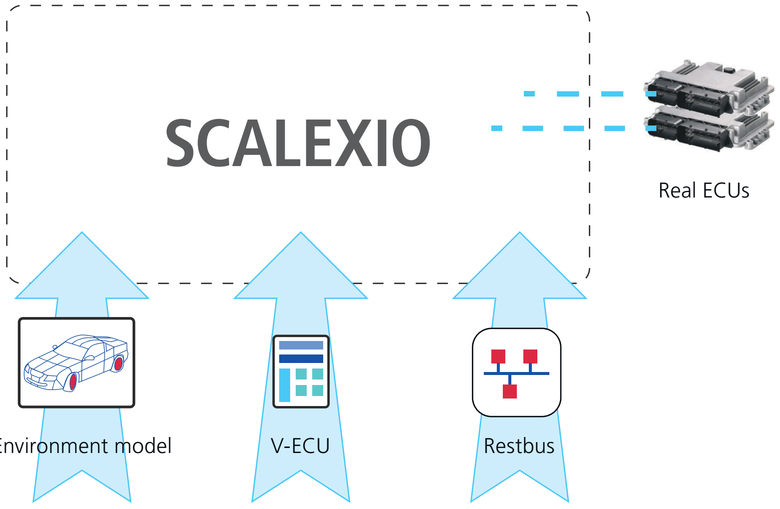
**Introduction**

If an ECU hardware prototype is not yet available, a V-ECU can be simulated on SCALEXIO or a MicroAutoBox III instead. Just like any other behavior model, you can include V-ECUs in a SCALEXIO or MicroAutoBox III real-time application.

**Integration in a network with real ECUs**

The V-ECU can run either alone or together with environment models, other V-ECUs, restbus ECUs, and/or real ECUs.

The following illustration shows this setup for SCALEXIO as an example.



**Related topics**

Basics

## SIL Testing and Sensor Simulation

**Introduction**

Sensor Simulation provides tools and dSPACE product features used to validate camera or laser sensors in simulations.

The SIL testing tool chain supports Sensor Simulation, i.e., you can validate sensors in a completely virtual environment.

**Related topics**

Basics

Simulation with Sensors in a SIL Environment (Sensor Simulation Overview 📖)

## Simulating Software Component Code on the Virtual Functional Bus (VFB) Level

**Introduction**

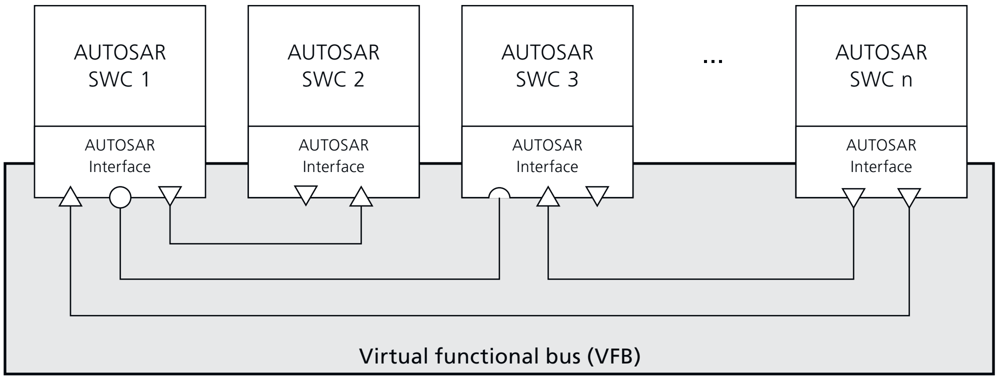
To simulate the application software components (SWCs) of an automotive electrics/electronics system on the *virtual functional bus* (VFB) level, SystemDesk lets you create a *preconfigured ECU*. Simulating such a preconfigured ECU lets you verify the dynamic system behavior of interconnected SWCs at an early design stage if the code files for the SWCs are already available.

**Basics on the VFB concept**

The *virtual functional bus* (VFB) is an AUTOSAR concept to separate the development of an application, which is a composition of interconnected application SWCs, from the development of the infrastructure of the SWCs. As a result, the SWCs implementing the application are largely independent of the communication mechanisms through which one SWC interacts with other SWCs or with sensor or actuator hardware. AUTOSAR application SWCs can therefore be implemented independently of the underlying hardware.

This independence is achieved by providing a *virtual functional bus* (VFB) as a means for standardized basic software and virtual, idealized hardware. The VFB defines standardized services (such as those to be provided later by an operating system or by ECU basic software) and available communication methods between components (sender/receiver, client/server, etc.) for a virtual execution platform.

The illustration below visualizes the VFB concept.



**Creating a preconfigured ECU**

To simulate application SWCs on the VFB level, SystemDesk lets you create a *preconfigured ECU*. All the application SWCs to be simulated are mapped to that single ECU, which SystemDesk preconfigures automatically, and which represents the idealized system according to the VFB concept. Since the effects of the underlying ECU hardware are not interesting in this early design stage, the software components seem to run on a VFB.

**Simulating on the virtual functional bus**

**Mapping all SWCs to one preconfigured ECU** You can simulate on the VFB level by mapping all the SWCs to a single ECU.

**Creating an ECU configuration realizing the VFB behavior** SystemDesk automatically creates an ECU configuration for the preconfigured ECU. The ECU configuration includes RTE and OS configurations and an ECU software composition.

**Automatic scheduling** SystemDesk computes a default scheduling automatically based on the RTE events in the software architecture. SystemDesk maps all runnables to be executed with the same cycle time to the same task. A separate task is generated for each acyclic runnable.

For further information on VFB simulation, refer to Using a Preconfigured ECU and System for Simulation (SystemDesk Manual 📖).

**Related topics**

Basics

Using a Preconfigured ECU and System for Simulation (SystemDesk Manual 📖)

# Integrating and Simulating Classic ECU Basic Software

**Introduction**

SystemDesk supports you with classic basic software modules that define certain basic software functionalities of an ECU.

**AUTOSAR classic ECU software architecture**

The AUTOSAR development partnership has defined a standardized architecture for classic ECU software.

AUTOSAR ECU software is divided into the following parts:

- The application software is the functional part of the ECU software such as the controller code. It consists of AUTOSAR software components.
- The run-time environment (RTE) is a static middle-ware software that allows ECU function development independently of the ECU hardware. Each V-ECU has an RTE of its own.

- The basic software is a standardized software layer providing services to the software components. It is necessary in order to run the functional part of the software.

For a detailed explanation of the AUTOSAR goals and definitions, refer to the technical reports and explanatory documents at:

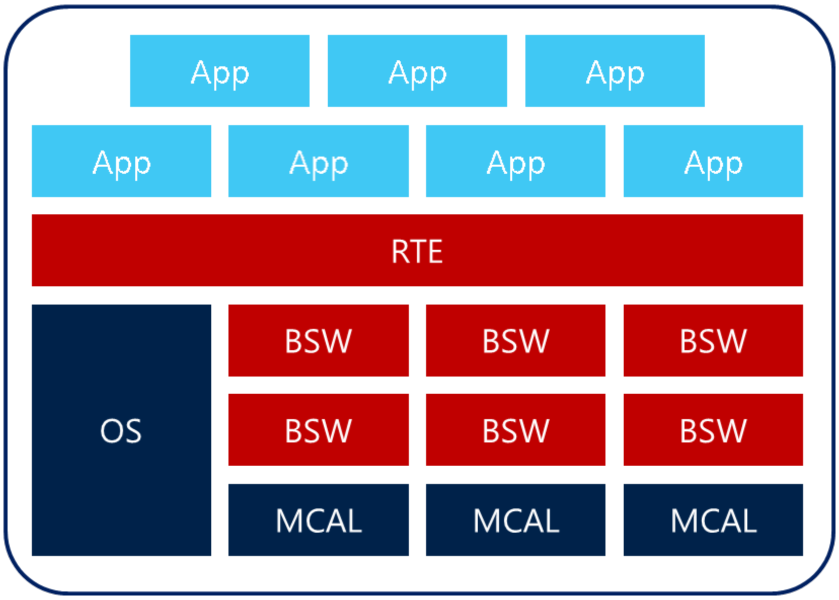
http://www.autosar.org.

**Basic software modules** A basic software module defines a certain basic software functionality on an ECU. A module consists of the following:

- A module configuration that contains all the module's ECU parameters
- A basic software module description and an optional service component if the basic software module has AUTOSAR interfaces
- Code files that are generated for the basic software module by a code generator

For an overview of the basic software module support for AUTOSAR classic V-ECUs and non-AUTOSAR V-ECUs, refer to Basic Software Module Support for Classic V-ECUs on page 101.

**Software architecture of an AUTOSAR classic V-ECU** The following illustration shows an example of a classic V-ECU and its software architecture according to AUTOSAR.



Legend:

🟦 Application software 🟦 dSPACE basic software 🟥 Third-party basic software

**Simulating basic software modules**

For simulating basic software modules, you can use related implementations provided by dSPACE or generated with a basic software configuration tool such as EB tresos studio. SystemDesk's *ECU Configuration Framework* is adaptable for RTE and BSW modules of any vendor. Contact dSPACE Support for available vendor-specific adaptations.

**Related topics**

Basics

References

## Integrating External Code

**Introduction**

SystemDesk lets you integrate external code files in a code-based V-ECU.

SystemDesk lets you create V-ECU implementations based on the following external code:

- AUTOSAR code for software components that are not modeled in SystemDesk
- Non-AUTOSAR legacy code

**Creating code-based V-ECUs**

You can create a code-based virtual ECU that uses the microcontroller abstraction layer (MCAL) as the interface between the production code and the simulator. In case of non-AUTOSAR code, you have to adapt the lower interfaces of your code to the APIs provided by dSPACE for integration in a V-ECU. The dSPACE APIs are based on a subset of the MCAL specified by AUTOSAR.

For more information, refer to https://www.dspace.com/go/CodeBasedVECU.

**Integrating non-AUTOSAR legacy code using the Legacy Code Integrator**

dSPACE provides a Legacy Code Integrator (LCI) to integrate non-AUTOSAR legacy code in a V-ECU. The LCI wraps the code in a software component and creates an appropriate V-ECU for it.

To get the Legacy Code Integrator, contact dSPACE Support.

**Related topics**

Basics

## Stimulating and Manipulating SWCs or V-ECUs via RTE Interventions

**Introduction**

SystemDesk lets you configure RTE interventions to stimulate, manipulate, or observe the communication of software components during simulation run time.

**Basics on RTE interventions**

**Accessing SWC communication** *RTE interventions* are additional insertions in the original RTE code that let you access and overwrite the communication of software components (SWCs) and underlying runnables in a simulation. For example, they let you stimulate SWC ports or inject error states in a simulation to test the behavior of application software components.

SystemDesk lets you create RTE interventions for:

- Read/write access to data elements of connected and unconnected ports of software components
- Read/write access to operation arguments of software components
- Read/write access to interrunnable variables of software components
- Read/write access to status return values of RTE API functions
- Triggering the execution of self-defined actions by executing custom C code

**Connecting stimuli** You can connect internal or external stimuli to software components and their runnables via RTE interventions:

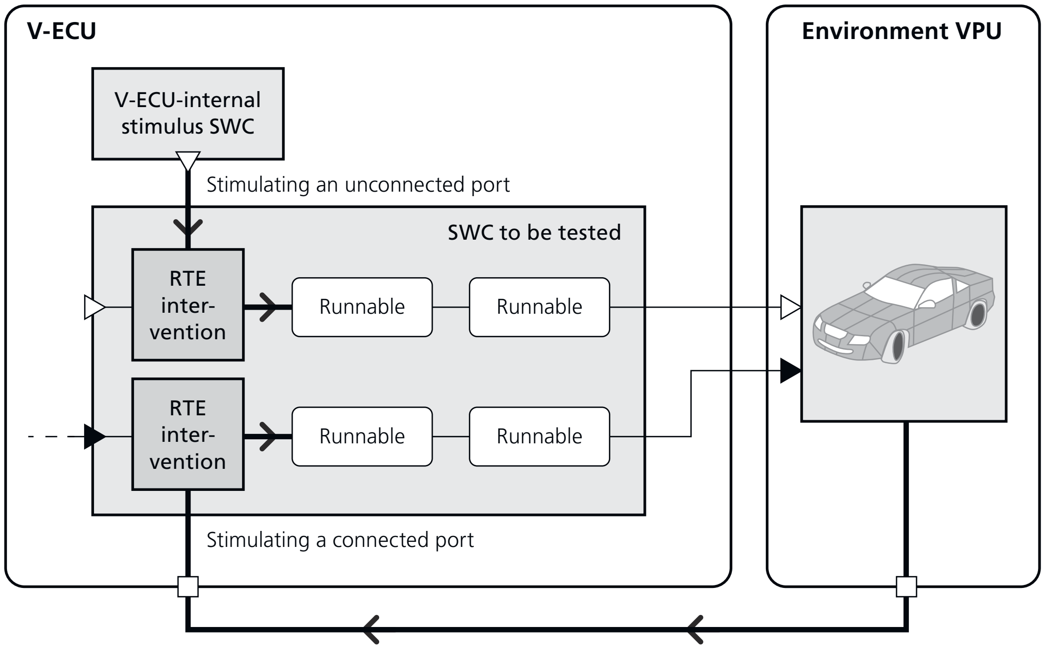
- You can use a specific stimulus SWC integrated on the V-ECU as an internal stimulus, and connect the stimulus SWC to the SWCs to be tested via *RTE service ports*.
- You can use an environment model as a source of external stimulus signals, and connect the model to the V-ECU containing the SWCs to be tested via *data access points*.

You can enable and disable RTE interventions during a simulation.

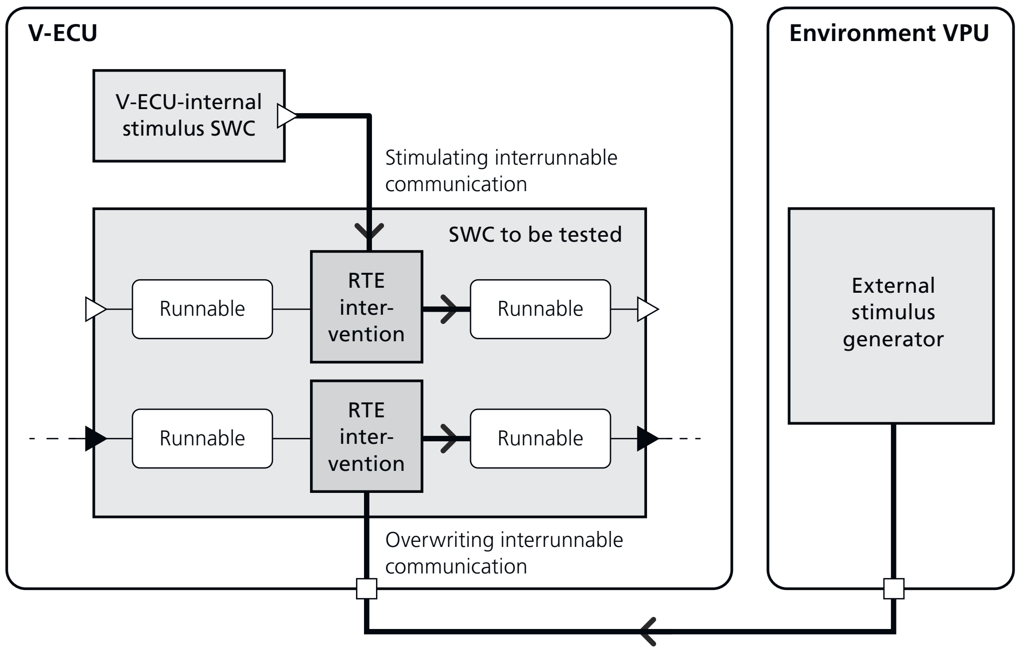
**Examples for using RTE interventions**

**Stimulating connected or unconnected ports** The following illustration shows an example of two RTE interventions that are inserted in the communication of sender-receiver interfaces. Both RTE interventions are used for stimulation:

- The upper RTE intervention stimulates an unconnected inport, for example, by connecting it to a V-ECU-internal stimulus SWC.
- The lower RTE intervention affects a connected inport, for example, by connecting it to an environment VPU. In this case, the original signal from the inport is overwritten by the signal coming from the model executed on the environment VPU.



**Manipulating interrunnable communication** The illustration below shows an example of two RTE interventions that are inserted in the communication between runnables. Both are used to manipulate the communication. The upper RTE intervention gets its input signal from a V-ECU-internal stimulus SWC. The lower RTE intervention gets its input signal from an environment VPU. In both cases, the original signal is overwritten.


V-ECU
Environment VPU
V-ECU-internal stimulus SWC
Stimulating interrunnable communication
SWC to be tested
Runnable
RTE inter-vention
Runnable
Runnable
RTE inter-vention
Runnable
External stimulus generator
Overwriting interrunnable communication

**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. For details on the specific A2L variables for simulation, refer to Specific Variables for Simulation (VEOS Manual 🕮).

For details on RTE interventions, refer to Configuring RTE Interventions (SystemDesk Manual 🕮).

**Related topics**

Basics

SystemDesk........................................................................................................................ 83

## Debugging Source Code During Offline Simulation

**Introduction**

To debug your target C code during offline simulation, VEOS supports you with some debugging features.

**Debugging target code in an offline simulation**

**Basics** Source code debugging is useful, for example, to locate problems in:
- ECU software
- Environment model code

You can set breakpoints, inspect variables, and step through the code.

**Debugging 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.

To debug target code of a specific VPU in a simulation, you can attach the related VPU process to the debugger.

For information on debugging with VEOS, refer to Basics on Debugging Source Code in an Offline Simulation (VEOS Manual 🕮).

**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.

**Related topics**

Basics

VEOS........................................................................ 88

## Memory Segments and Calibration Pages

**Introduction**

*Memory segments* contain the calibratable parameters of an ECU application.

A memory segment can be arranged in *calibration pages*: For the ECU, these pages appear at the same addresses, that is, only one page is visible to the ECU at a time. When you change parameter values on one page, you can make these changes available to the ECU via a single page switch.

**Calibration pages on V-ECUs**

- Memory segments of classic V-ECUs built with VEOS have two calibration pages.

  You can use ControlDesk to switch the active V-ECU calibration page at run time. 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 🕮).
- Memory segments of classic V-ECUs built with ConfigurationDesk have one calibration page.

## OEM–Supplier Workflow Support

**Introduction**

The *VECU* 3.0 format for V-ECU implementation files is designed to improve OEM–supplier workflows.

**VECU 3.0 and CTLGZ format** The VECU 3.0 format introduced with dSPACE Release 2020-B replaces the former CTLGZ 2.x format.

The SIL testing tool chain still supports the CTLGZ 2.x file format. For more compatibility information, refer to Compatibility of VEOS 5.2 (VEOS Manual).

**Improvements to OEM–supplier workflows**

**IP protection and shortened build times** 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.

A VECU file is structured according to *packages*, each of which can contain, for example, source code and platform-dependent object code. A VECU file can be updated *incrementally* by exchanging individual packages.

**Multi-platform support** 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.

**Recognition of platform compatibility** During the build process for a specific simulation platform, the build tool, such as ConfigurationDesk, can recognize whether the platform-dependent content of the VECU file is compatible with that specific simulation platform. The build tool automatically selects the package that is most suitable for the specific simulation platform. For example, a precompiled package is preferred over a package with generic source code.

**Upcoming improvements**

The following improvements will be provided in a future dSPACE Release:
- Creating user-specific packages, for example, for code-based V-ECUs, including the possibility to precompile libraries
- Managing packages in a container
- Managing platform dependencies

Contact dSPACE Support for an intermediate solution.

# Products for SIL Testing

**Introduction**

Gives an overview of the dSPACE products in the tool chain for SIL testing.

**Where to go from here**

Information in this section

## AutomationDesk

**Introduction**

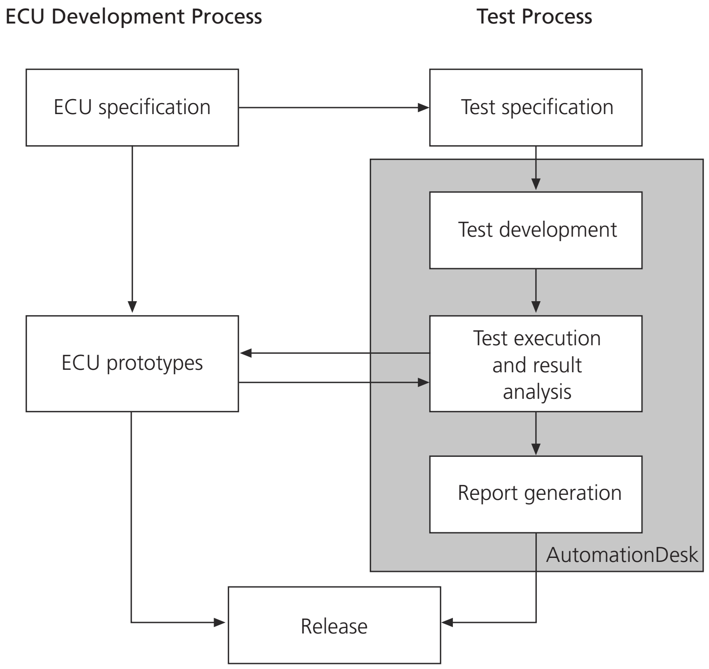
AutomationDesk is the dSPACE test software for automated hardware-in-the-loop (HIL) and software-in-the-loop (SIL) testing of the application software or diagnostic functions of electronic control units (ECUs).

## Field of application and product overview

**Field of application** With AutomationDesk, you can create and specify control flows and test parameters, execute tests and log the results. AutomationDesk's main application field is the automated testing of application software or diagnostic functions of real and virtual ECUs, e.g., with hardware-in-the loop (HIL) simulations or in offline simulations with VEOS. Tests that were originally created for testing virtual ECUs can easily be reused for testing real ECUs.

With AutomationDesk, a series of test cases can be executed at any time, for example, at night or over the weekend. This increases test coverage and the quality of the ECU software while saving time and costs.

The illustration below shows the integration of AutomationDesk into the ECU development process:



**Product overview** AutomationDesk provides libraries containing a large number of predefined test steps, e.g., for easy access to the simulation platform, to a Failure Insertion Unit (FIU), or to calibration or diagnostics software.

AutomationDesk provides the following capabilities:

- Test management and execution
- Graphical test development
- Platform management, for example, for handling applications on simulation platforms
- Automation libraries such as:
  - *XIL API* and *XIL API Convenience* libraries to access simulation platforms via ASAM XIL API standard
  - *ControlDesk Access* library to remote-control ControlDesk
- Management of test cases

- Support of VEOS for PC-based offline simulation; support of SCALEXIO and MicroAutoBox III for real-time simulation

**Integration into the SIL testing tool chain**

**Overview** For an overview of how AutomationDesk is integrated into the SIL testing tool chain, refer to Experimenting and testing with the simulation system on page 24.

**Accessing virtual ECUs** Virtual ECUs can be accessed by AutomationDesk like real ECUs. They are described by A2L files and can be accessed via the *ControlDesk Access* and *Remote Calibration (COM)* libraries.

For more information, refer to Overview of the ControlDesk Access Library Elements (AutomationDesk Accessing ControlDesk 🕮).

**Accessing environment models** Environment models can be accessed by AutomationDesk via the *XIL API* library, which supports the MAPort (model access port) implementation of the XIL API standard to access a simulation platform. Environment models can also be accessed via the *ControlDesk Access* library.

For instructions, refer to How to Prepare a Project for Using the XIL API Library (AutomationDesk Accessing Simulation Platforms 🕮).

**Testing** Testing with V-ECUs and environment models is the same as with real ECUs and other dSPACE simulation platforms.

Using the XIL API library on environment models, you can:

- Read and write to scalar and vector-type variables. Refer to How to Write and Read Variables (AutomationDesk Accessing Simulation Platforms 🕮).
- Capture data by using complex trigger conditions. Refer to How to Capture Data (AutomationDesk Accessing Simulation Platforms 🕮).
- Stimulate variables from a simulation application. Refer to Basics on Stimulating Variables via AutomationDesk (AutomationDesk Accessing Simulation Platforms 🕮).

**Changing the simulation platform and reusing tests** AutomationDesk lets you switch from offline to real-time simulation by changing the simulation platform from VEOS to SCALEXIO or a MicroAutoBox III. Thus, tests that were originally created for VEOS can be reused for SCALEXIO or the MicroAutoBox III with few adaptations.

To reuse tests, you have to change the simulation platform from VEOS to SCALEXIO or a MicroAutoBox III.

- Reusing tests is possible if variable connections can be restored. A variable connection involves connecting a variable to an instrument or a signal from a signal generator, for example.

  This is possible if the paths and names of variables are the same in the variable descriptions for the platforms/devices in the offline and the real-time simulation.
- To do so, specify the MAPortConfiguration data object to reference the application path (SDF) and platform identifier of the changed simulation platform.

Refer to MAPortConfiguration (Data Object) (AutomationDesk Accessing Simulation Platforms 🕮).

- If your automation sequence includes stimulus generation via XIL API, you have to specify an STZ file that references the changed platform.

**Tip**

If you have an STZ file, for example, created for the VEOS simulation platform, the **Signal Editor** lets you change the file to reference the SCALEXIO or MicroAutoBox III simulation platform. Refer to Change Platform (ControlDesk Signal Editor 🕮) and Check Mapping (ControlDesk Signal Editor 🕮).

You can now reuse the test.

**Related topics**

Basics

Basics on Stimulating Variables via AutomationDesk (AutomationDesk Accessing Simulation Platforms 🕮)
Overview of the ControlDesk Access Library Elements (AutomationDesk Accessing ControlDesk 🕮)
Testing the Simulation Application via XIL API Library (SCALEXIO – Hardware and Software Overview 🕮)
Tool Chain and Workflow for SIL Testing........................................................................... 21

HowTos

How to Capture Data (AutomationDesk Accessing Simulation Platforms 🕮)
How to Prepare a Project for Using the XIL API Library (AutomationDesk Accessing Simulation Platforms 🕮)
How to Write and Read Variables (AutomationDesk Accessing Simulation Platforms 🕮)

References

Change Platform (ControlDesk Signal Editor 🕮)
Check Mapping (ControlDesk Signal Editor 🕮)
MAPortConfiguration (Data Object) (AutomationDesk Accessing Simulation Platforms 🕮)

# Automotive Simulation Models and ModelDesk

**Introduction**

dSPACE Automotive Simulation Models can be used as environment models for SIL testing. They can be parameterized using ModelDesk.

## Field of application and product overview

**Field of application** The Automotive Simulation Models (ASM) are open Simulink® models that let you simulate automotive applications: for example, vehicle dynamics or traffic scenarios. These models provide the environment for V-ECUs or real ECUs in a simulation scenario.

**Product overview**

- Automotive Simulation Models (ASMs) for the simulation of combustion engines and electric motors, vehicle dynamics systems, and virtual traffic scenarios
- Predefined layouts for the ASMs to be used in experiments with ControlDesk
- ModelDesk for managing the parameter values of ASMs and downloading the values to a simulation platform
- For use with ASMs, ModelDesk also provides:
  - A *Road Generator* to specify roads
  - A *Scenario Editor* to specify the movement of the ASM vehicle and traffic objects in traffic scenarios

## Integration into the SIL testing tool chain

**Overview** For an overview of how the Automotive Simulation Models are integrated into the SIL testing tool chain, refer to Developing the simulation system on page 22.

For an overview of how ModelDesk is integrated into the SIL testing tool chain, refer to Experimenting and testing with the simulation system on page 24.

**Developing environment models** The Automotive Simulation Models can be used as environment models (SICs) in a simulation system.

For more information, refer to Introduction to Automotive Simulation Models (ASM User Guide ).

**Parameterizing the environment model** When performing simulations, ModelDesk lets you parameterize the environment model without rebuilding it.

For more information, refer to Features of ModelDesk (ModelDesk Basics ).

**Reusing parameter sets when switching from offline to real-time simulation** ModelDesk lets you reuse parameter sets when you switch from offline to real-time simulation. This requires that the SDF files for offline to real-time simulation contain the same paths and variables names.

For more information, refer to How to Choose a Model and Initialize the Consistency Check (ModelDesk Parameterizing 🕮).

**Related topics**

Basics

Features of ModelDesk (ModelDesk Basics 🕮)
Introduction to Automotive Simulation Models (ASM User Guide 🕮)
Tool Chain and Workflow for SIL Testing....................21

HowTos

How to Choose a Model and Initialize the Consistency Check (ModelDesk Parameterizing 🕮)

## Bus Manager

**Introduction**

The Bus Manager lets you configure bus communication for simulation purposes (e.g., restbus simulation). You can work with multiple communication matrix files of various file formats and configure the communication of multiple communication clusters at the same time.

There are two versions of the Bus Manager:

- Bus Manager in ConfigurationDesk

  The Bus Manager in ConfigurationDesk lets you configure bus communication and either implement it in a real-time application for SCALEXIO or a MicroAutoBox III, or generate bus simulation containers for integration in an offline simulation application for VEOS.
- Bus Manager (stand-alone)

  The Bus Manager (stand-alone) lets you configure bus communication and generate bus simulation containers for integration in an offline simulation application for VEOS.

**Field of application and product overview**

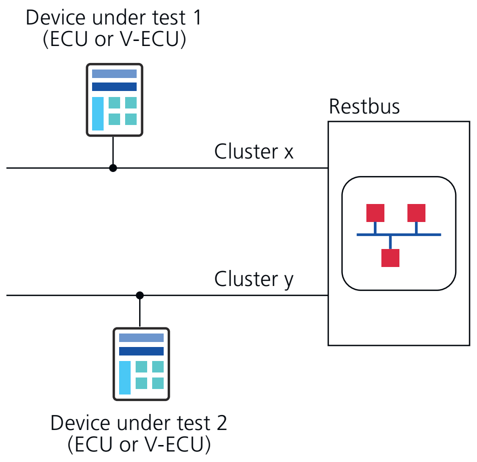
**Field of application** The Bus Manager lets you simulate the bus communication between ECUs of the same or different bus systems, e.g., CAN or LIN communication clusters.

The Bus Manager lets you configure the simulation of individual ECUs, and the simulation of a restbus.

- *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.

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.



- *ECU simulation*

  Simulating individual ECUs lets you, for example, test the communication of an ECU under development before the real ECU is available.

**Product overview** The Bus Manager provides the following capabilities:

- Support of CAN and LIN
- Supported communication matrix file formats:
  - AUTOSAR system description
  - DBC
  - FIBEX
  - LDF
- Support of *static* and *dynamic* bus simulation:
  - In *static bus simulation*, *constant* signal values are transmitted and received by the bus. This is sufficient, for example, when you perform a restbus simulation and the ECU to be tested expects a cabin temperature within a specific temperature range from an ECU that is part of the simulated restbus.
  - In *dynamic bus simulation*, bus communication is coupled to a *behavior model*, which is modeled in MATLAB/Simulink, for example. The control algorithms of the behavior model calculate the dynamic signal values of the simulated ECUs during simulation.

    Coupling a behavior model is necessary if the signals of a simulated ECU must change dynamically during run time (e.g., cabin temperature values that are regulated by the ECU to be tested and exchanged with the simulated ECU).

  Depending on the bus configuration, you can manipulate the bus communication via ControlDesk both in static and dynamic bus simulations.

**Integration into the SIL testing tool chain**

**Overview** For an overview of how the Bus Manager is integrated in the SIL testing tool chain, refer to Developing the simulation system on page 22.

**Implementing bus communication in offline simulation applications for VEOS** The Bus Manager lets you configure bus communication and generate bus simulation container (BSC) files that can be integrated in an offline simulation application for VEOS.

For more information, refer to Typical Workflows for Using the Bus Manager and Generating Bus Simulation Containers (Bus Manager (Stand-Alone) Implementation Guide 📖).

**Implementing bus communication in real-time applications for SCALEXIO or a MicroAutoBox III** The Bus Manager in ConfigurationDesk lets you configure bus communication and implement it in a real-time application for SCALEXIO or a MicroAutoBox III.

As an alternative, BSC files generated with the Bus Manager can be integrated in a real-time application for SCALEXIO or a MicroAutoBox III.

For more information, refer to Workflow for Using the Bus Manager and Configuring Bus Communication for Real-Time Applications (ConfigurationDesk Bus Manager Implementation Guide 📖).

**Related topics**

Basics

Tool Chain and Workflow for SIL Testing........................................................................................21
Typical Workflows for Using the Bus Manager and Generating Bus Simulation Containers (Bus Manager (Stand-Alone) Implementation Guide 📖)
Workflow for Using the Bus Manager and Configuring Bus Communication for Real-Time Applications (ConfigurationDesk Bus Manager Implementation Guide 📖)

## ConfigurationDesk

**Introduction**

ConfigurationDesk is a software tool for implementing a simulation system on SCALEXIO or a MicroAutoBox III.

**Field of application and product overview**

**Field of application** ConfigurationDesk lets you configure and implement real-time applications for SCALEXIO and a MicroAutoBox III.

**Product overview** ConfigurationDesk provides the following capabilities:

- Integration of an environment model, I/O functionality of a SCALEXIO/MicroAutoBox III system, and devices externally connected to the SCALEXIO/MicroAutoBox III system (e.g., ECUs, sensors, actuators)
- Separation of environment modeling from I/O configuration and assignment to the hardware resources of a specific SCALEXIO/MicroAutoBox III system

- Build environment for building and downloading a real-time application for a SCALEXIO/MicroAutoBox III system

For detailed information on ConfigurationDesk, refer to ConfigurationDesk Real-Time Implementation Guide 🕮.

**Integration into the SIL testing tool chain**

**Overview** For an overview of how ConfigurationDesk is integrated in the SIL testing tool chain, refer to Integrating and building the simulation system on page 23.

**Importing V-ECU implementations and environment models, and connecting them with real ECUs** ConfigurationDesk lets you import:

- V-ECU implementations
- Bus simulation container (BSC) files
- Environment models:
  - Simulink® models - either directly or as Simulink implementation containers

    For details, refer to Creating the Interface of Behavior Models (Model Interface Package for Simulink - Modeling Guide 🕮) and Workflow for Integrating Simulink Implementation Containers in Executable Applications (ConfigurationDesk Getting Started 🕮).
  - Functional Mock-up Units (FMUs)

ConfigurationDesk lets you integrate these elements in a simulation system to be used on a SCALEXIO/MicroAutoBox III system.

For instructions, refer to How to Import a Model Topology (ConfigurationDesk Real-Time Implementation Guide 🕮).

**Configuring and building the real-time application for the SCALEXIO/MicroAutoBox III system** ConfigurationDesk lets you:

- Configure the real-time application to be executed on a SCALEXIO/MicroAutoBox III system. This includes:
  - Specifying the model interface. Refer to Specifying the Model Interface (ConfigurationDesk Real-Time Implementation Guide 🕮).
  - Modeling tasks. Refer to Modeling Executable Applications and Tasks (ConfigurationDesk Real-Time Implementation Guide 🕮).
  - Specifying the access to the real-time hardware
  - Configuring the build process. Refer to Configuring the Build Process (ConfigurationDesk Real-Time Implementation Guide 🕮).
- Build the real-time application for the SCALEXIO/MicroAutoBox III system. Refer to Starting the Build Process (ConfigurationDesk Real-Time Implementation Guide 🕮).

**Related topics**

Basics

Configuring the Build Process (ConfigurationDesk Real-Time Implementation Guide 🕮)
Creating the Interface of Behavior Models (Model Interface Package for Simulink - Modeling Guide 🕮)
Modeling Executable Applications and Tasks (ConfigurationDesk Real-Time Implementation Guide 🕮)
Specifying the Model Interface (ConfigurationDesk Real-Time Implementation Guide 🕮)
Starting the Build Process (ConfigurationDesk Real-Time Implementation Guide 🕮)
Tool Chain and Workflow for SIL Testing........................................................................... 21
Workflow for Integrating Simulink Implementation Containers in Executable Applications (ConfigurationDesk Getting Started 🕮)

HowTos

How to Import a Model Topology (ConfigurationDesk Real-Time Implementation Guide 🕮)

# ControlDesk

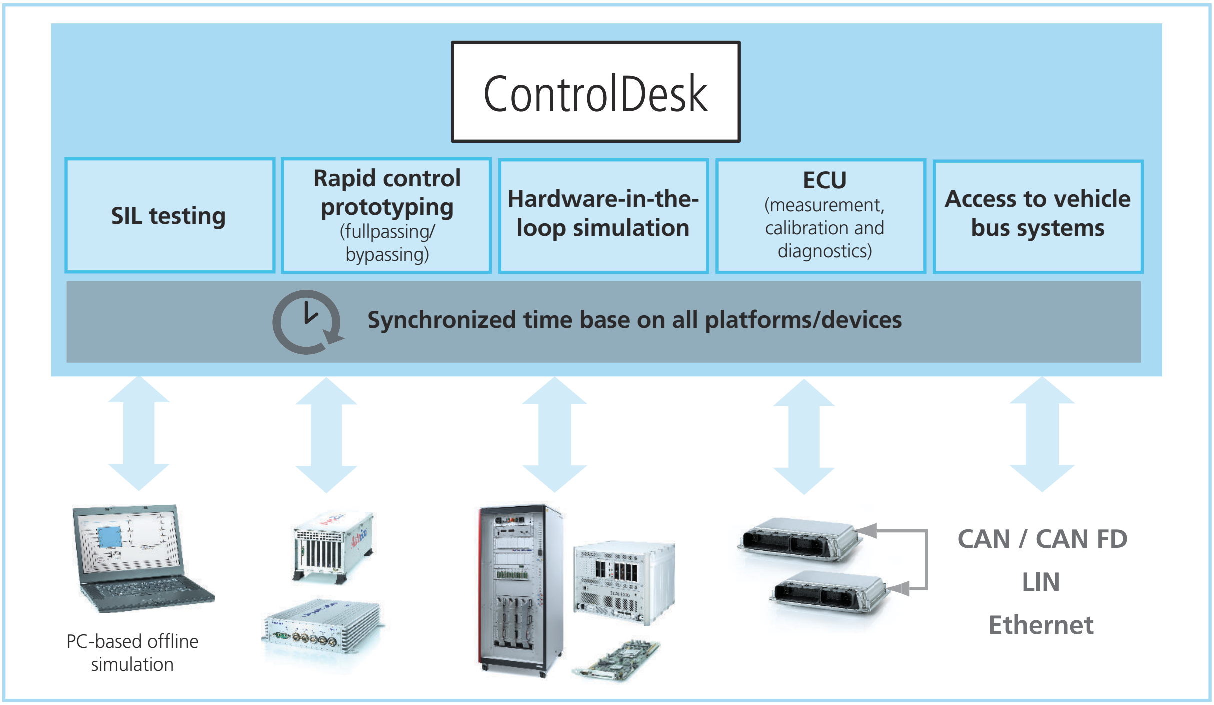
**Introduction**

ControlDesk is the dSPACE experiment software for ECU development.

**Field of application and product overview**

**Field of application** ControlDesk lets you perform all experiment tasks in ECU development within a single working environment. Below is a list of the most important fields of application ControlDesk can be used for:

- SIL testing
- Rapid control prototyping (fullpass, bypass)
- Hardware-in-the-loop (HIL) simulation
- ECU measurement, calibration, and diagnostics
- Access to (real) vehicle bus systems (CAN, Ethernet, FlexRay, LIN)


ControlDesk
SIL testing
Rapid control prototyping (fullpassing/ bypassing)
Hardware-in-the-loop simulation
ECU (measurement, calibration and diagnostics)
Access to vehicle bus systems
Synchronized time base on all platforms/devices
PC-based offline simulation
CAN / CAN FD
LIN
Ethernet

**Product overview** ControlDesk provides the following capabilities:

- Project management
- Platform management, e.g. for handling applications on simulation platforms
- Creation of layouts and instruments, for example, for modifying parameter values interactively
- Synchronous measurement on all data sources
- Support of VEOS for PC-based offline simulation; support of SCALEXIO/MicroAutoBox III systems for real-time simulation
- Monitoring and replaying bus communication during a running simulation, and displaying bus statistics
- Stimulus generation on dSPACE simulation platforms such as VEOS and SCALEXIO/MicroAutoBox III systems
- Tool automation

## Integration into the SIL testing tool chain

**Overview** For an overview of how ControlDesk is integrated into the SIL testing tool chain, refer to Experimenting and testing with the simulation system on page 24.

**Loading a simulation application** Loading a simulation application to VEOS and to a SCALEXIO/MicroAutoBox III system is supported.

Refer to Handling Real-Time and Offline Simulation Applications (ControlDesk Platform Management 🕮).

**Accessing virtual ECUs** Classic V-ECUs can be accessed by ControlDesk like real ECUs. They are described by A2L files and can be accessed via XCP on Ethernet.

Refer to Basics on Accessing an ECU (ControlDesk Platform Management 🕮).

**Accessing environment models** In an offline simulation, an environment model can be accessed by the *VEOS platform*. In a real-time simulation, an environment model can be accessed by a *SCALEXIO* or *MicroAutoBox III platform*. In both cases, the available variables are described by an SDF file.

**Experimenting** Experimenting with V-ECUs and environment models is the same as experimenting with real ECUs and other dSPACE simulation platforms.

For example, you can:

- Perform triggered measurements on environment models in offline and in real-time simulation. To configure triggered measurements, refer to Configuring Triggered Measurement on dSPACE Platforms (ControlDesk Measurement and Recording 🕮).
- Define stimulus signals graphically for time-synchronous stimulus generation on VEOS, SCALEXIO, and a MicroAutoBox III, including replaying measured data via ControlDesk's Signal Editor. Refer to Introduction to the Signal Editor (ControlDesk Signal Editor 🕮).

**Changing the simulation platform and reusing experiments** For SIL testing, ControlDesk lets you migrate from offline to real-time simulation by changing the simulation platform from VEOS to SCALEXIO or a MicroAutoBox III and reconfiguring the XCP devices. This allows you to modify experiments originally created for VEOS and reuse them for SCALEXIO or a MicroAutoBox III. For more information, refer to Switching the Simulation Platform and Reusing Experiment Parts (ControlDesk Platform Management 🕮).

**Related topics**

Basics

Basics on Accessing an ECU (ControlDesk Platform Management 🕮)
Configuring Triggered Measurement on dSPACE Platforms (ControlDesk Measurement and Recording 🕮)
Handling Real-Time and Offline Simulation Applications (ControlDesk Platform Management 🕮)
Introduction to the Signal Editor (ControlDesk Signal Editor 🕮)
Switching the Simulation Platform and Reusing Experiment Parts (ControlDesk Platform Management 🕮)
Tool Chain and Workflow for SIL Testing.......... 21

## dSPACE AUTOSAR Compare

**Introduction**

dSPACE AUTOSAR Compare supports scenarios where you have to compare and merge AUTOSAR files.

**Field of application and product overview**

**Field of application** dSPACE AUTOSAR Compare provides the following:

- Comparison and merging of AUTOSAR ARXML files
- Possible integration in a version control system as a comparison tool for different versions of AUTOSAR ARXML files
- Support of SystemDesk-TargetLink roundtrips

**Product overview** dSPACE AUTOSAR Compare provides

- A user interface to inspect differences in ARXML file versions, to copy elements, and to merge files
- A command line interface for integration in a tool chain

**Integration into the SIL testing tool chain**

**Overview** dSPACE AUTOSAR Compare can be used whereever ARXML files are modified and exchanged between tools. It lets you detect and resolve differences.

**SystemDesk-TargetLink roundtrips** dSPACE AUTOSAR Compare can be used in SystemDesk-TargetLink roundtrips. dSPACE AUTOSAR Compare helps you to integrate SystemDesk and TargetLink into a tool chain by intelligent merging of AUTOSAR files.

For example, dSPACE AUTOSAR Compare lets you copy only the elements modeled with TargetLink such as internal behaviors and implementations of software components, and merge them to architecture ARXML files. The AUTOSAR elements and properties that are not supported by TargetLink are preserved. These elements are not copied to TargetLink files but intelligent merging keeps the elements in architecture files.

For more information, refer to Basics on dSPACE AUTOSAR Compare (dSPACE AUTOSAR Compare Manual 🕮).

**Related topics**

Basics

Basics on dSPACE AUTOSAR Compare (dSPACE AUTOSAR Compare Manual 🕮)
SystemDesk........ 83
TargetLink........ 84
Tool Chain and Workflow for SIL Testing........ 21

## MicroAutoBox III

**Introduction**

The MicroAutoBox III is a dSPACE real-time system for performing rapid control prototyping (RCP). It can operate without user intervention, just like an ECU, and can be installed virtually anywhere in the vehicle.

**Field of application and product overview**

In rapid control prototyping, the MicroAutoBox III is used to flexibly develop and optimize control designs without manual programming.

The MicroAutoBox III runs simulations *in real time*. For more information on real-time simulation, refer to Basics on Real-Time Simulation with SCALEXIO/MicroAutoBox III on page 30.

**Integration into the SIL testing tool chain**

**Overview** For an overview of how the MicroAutoBox III is integrated into the SIL testing tool chain, refer to Tool Chain and Workflow for SIL Testing on page 21.

**Platform for real-time simulation** The MicroAutoBox III supports the real-time simulation of simulation systems consisting of production-code V-ECUs or controller behavior models (SIC, BSC, FMU).

**Rapid prototyping access to real ECUs** The MicroAutoBox III also supports ECU interfacing scenarios where the MicroAutoBox III provides the physical interface for rapid prototyping access to real ECUs. Thus, a legacy function on the real ECU can be bypassed with a new function executed on the MicroAutoBox III.

**Related topics**

Basics

## Model Interface Package for Simulink

**Introduction**

The Model Interface Package for Simulink is a software tool for generating code for Simulink models.

**Field of application and product overview**

**Field of application** The Model Interface Package for Simulink lets you prepare Simulink models for being simulated on VEOS, SCALEXIO, and a MicroAutoBox III.

**Product overview** The Model Interface Package for Simulink consists of the following components:

| Component | Description |
|---|---|
| Model Interface Blockset | The blockset lets you define the interface of a Simulink model abstractly.<br>The blockset lets you do the following: |

| Component | Description |
|---|---|
| | ▪ Establish model communication between the Simulink model and units such as a V-ECU using VEOS Player or ConfigurationDesk<br>▪ Connect the Simulink model to the I/O functions of ConfigurationDesk<br>▪ Separate individual models from an overall model in MATLAB/Simulink. The models can be executed in separate application processes on the simulation platform. |
| dSPACE Run-Time Target (`dsrt` system target) | The target for Simulink® Coder™ that lets you generate Simulink model code that can be used to build executable applications for SCALEXIO, the MicroAutoBox III, and VEOS. |

## Integration into the SIL testing tool chain

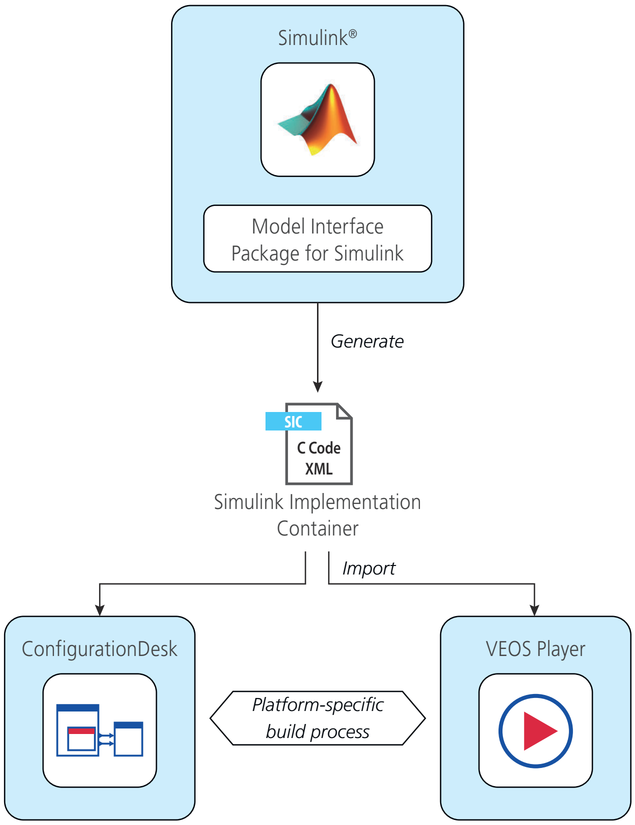
**Overview** For an overview of how the Model Interface Package for Simulink is integrated into the SIL testing tool chain, refer to Developing the simulation system on page 22.

**Generating Simulink implementation container (SIC) files** The Model Interface Package for Simulink lets you generate a Simulink implementation container (SIC) file based on a Simulink® model.

For details on generating an SIC file with Model Interface Package for Simulink, refer to Generating Simulink Implementation Containers (Model Interface Package for Simulink - Modeling Guide).

**Importing Simulink implementation container (SIC) files** You can:

- Import the SIC file to VEOS Player to build the model code for an offline simulation on VEOS. Refer to Importing Simulink Implementation Containers (SICs) (VEOS Manual).
- Import the SIC file to ConfigurationDesk to build the model code for in a real-time simulation on SCALEXIO or a MicroAutoBox III. Refer to Working with Simulink Implementation Containers (ConfigurationDesk Real-Time Implementation Guide).


Simulink®
Model Interface Package for Simulink
Generate
SIC
C Code
XML
Simulink Implementation Container
Import
ConfigurationDesk
Platform-specific build process
VEOS Player

**Tip**

When you integrate SIC files containing generated code for a Simulink model:

- No model code generation is needed for the build process in VEOS Player and in ConfigurationDesk.
- No MATLAB installation is needed for the build process.

**Related topics**

Basics

## MotionDesk

**Introduction**

MotionDesk is the dSPACE software for visualizing movements of objects such as a car in the 3-D world. It is used, for example, in vehicle dynamics applications.

**Field of application and product overview**

**Field of application** MotionDesk lets you visualize the movement of objects in the 3-D world. MotionDesk can visualize parts like vehicles or robotic arms that move in a 3-D world and are simulated on a dSPACE simulation platform. If your PC is fast enough, the latency between simulation and visualization is low, which makes the system capable of "man-in-the-loop" simulation. The simulation data can be recorded and replayed afterwards for a presentation of simulation results.

MotionDesk lets you create a 3-D world from a wide range of 3-D objects for vehicle simulation provided in a 3-D object library. The objects are assembled to create a scene in the MotionDesk 3-D View. Scenes look more realistic through the use of modern rendering techniques such as texture mapping.

Once created, the 3-D scene comes to life in MotionDesk. During simulation, MotionDesk gets motion data from a simulation platform and visualizes the movable objects in accordance with the data. To get the right view of the scene, observers can be defined with varied behaviors, for example, they can be static in the scene or follow a moving object.

**Product overview** MotionDesk provides the following capabilities:

- 3-D scene creation via the 3-D object library
- Calculating and acquiring motion data on dSPACE simulation platforms (including VEOS, SCALEXIO, and the MicroAutoBox III) via MotionDesk Blockset
- Recording and analyzing motion data
- Synchronized replay of different simulations in multiple tracks
- Support of VEOS for PC-based offline simulation; support of SCALEXIO/MicroAutoBox III systems for real-time simulation

**Integration into the SIL testing tool chain**

**Overview** For an overview of how MotionDesk is integrated into the SIL testing tool chain, refer to Experimenting and testing with the simulation system on page 24.

**Visualizing movements of objects in a 3-D world** You can model the environment for V-ECUs in Simulink® and visualize the movement of objects in the 3-D world using MotionDesk for SIL testing purposes. The Simulink model being used as an environment model must be prepared using the MotionDesk Blockset.

For more information, refer to Adapting Simulink Models (MotionDesk Calculating and Streaming Motion Data 📖).

**Changing the simulation platform** MotionDesk, like all other dSPACE software used for HIL simulation, interacts with VEOS in the same way as with a HIL simulator such as SCALEXIO.

For SIL testing, MotionDesk lets you switch from offline to real-time simulation by changing the motion data source from VEOS to SCALEXIO or the MicroAutoBox III.

For more information, refer to How to Select the Data Source (MotionDesk Scene Animation 📖).

**Related topics**

Basics

Adapting Simulink Models (MotionDesk Calculating and Streaming Motion Data 📖)
Tool Chain and Workflow for SIL Testing........................................................................ 21

HowTos

How to Select the Data Source (MotionDesk Scene Animation 📖)

## Real-Time Testing

**Introduction**

Real-Time Testing (RTT) lets you perform tests synchronously to the simulation application running on a simulation platform.

**Field of application and product overview**

**Field of application** Automated testing usually is performed by executing tests on a PC connected to a simulation platform such as a hardware-in-the-loop (HIL) or software-in-the-loop (SIL) system. However, this method often cannot cope where greater timing precision is required, for example, if ECU interaction has to be captured and responded to in a range of milliseconds.

Real-Time Testing runs on the simulation platform synchronously with the simulation application, so all test actions are performed on the basis of the simulation platform clock. Reactive tests which respond to changes in model variables within the same simulation step can therefore be implemented. Time measurements in tests are also far more precise, as there are no latencies in communication. Simulation step size is now the only limit to the maximum time resolution of measurements.

**Product overview** Real-Time Testing provides the following capabilities:

- Real-time test management via scripting (in Python) and graphical user interface
- Test execution synchronously with the simulation application: Variables of the simulation application can be measured and changed in every simulation step.
- No modification of the environment model necessary for real-time testing
- Test development based on standard Python modules, modules provided by dSPACE for real-time testing, and user Python modules

- Support of VEOS for PC-based offline simulation; support of SCALEXIO and the MicroAutoBox III for real-time simulation

**Integration into the SIL testing tool chain**

**Overview** For an overview of how Real-Time Testing is integrated into the SIL testing tool chain, refer to Experimenting and testing with the simulation system on page 24.

**Performing automated tests synchronously with the simulation application** Real-Time Testing (RTT) lets you perform automated tests synchronously with the simulation application. You can test on the VEOS, SCALEXIO, and MicroAutoBox III simulation platforms.

For more information, refer to Implementing RTT Sequences (Real-Time Testing Guide 🕮).

**Changing the simulation platform and reusing experiments** RTT sequences developed with Real-Time Testing are independent of the simulation platform onto which they are downloaded. Thus, RTT test sequences originally created for VEOS can be reused for SCALEXIO or a MicroAutoBox III.

To change the simulation platform, you only have to change the simulation platform access only.

**Related topics**

Basics

Implementing RTT Sequences (Real-Time Testing Guide 🕮)
Tool Chain and Workflow for SIL Testing........................................ 21

## RTI Bypass Blockset

**Introduction**

RTI Bypass Blockset is a Simulink® blockset for dialog-based configuration of ECU interfaces and ECU functions: e.g., for bypassing.

**Field of application and product overview**

**Field of application** RTI Bypass Blockset lets you configure ECU interfaces and implement accesses to and synchronizations with ECU-internal variables and function calls, for example, to implement new ECU functions.

**Product overview** RTI Bypass Blockset provides the following capabilities:

- Support of several methods for accessing and manipulating (V-)ECU applications: e.g., by accessing and overwriting (V-)ECU internal variables
- Support for synchronizing model code execution with (V-)ECU-internal events such as (V-)ECU-internal function executions
- Support of several ECU access types such as CCP, XCP and on-chip debug ports

## Integration into the SIL testing tool chain

**Overview** For an overview of how RTI Bypass Blockset is integrated into the SIL testing tool chain, refer to Additional steps when integrating and building the simulation system on page 25.

**Accessing V-ECU-internal variables and functions** 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.

The RTI Bypass Blockset supports the VEOS, MicroAutoBox III, and the SCALEXIO simulation platforms.

The ability to access V-ECU internal variables and functions from a dynamically integrated Simulink or TargetLink model provides the following advantages:

- You can perform tasks such as rapid control prototyping by implementing a function bypass .
- You can perform these tasks even if the V-ECU software architecture must not be modified and if the V-ECU is available only as a binary file.

For more information, refer to:

- Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS (RTI Bypass Blockset Reference )
- Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III (RTI Bypass Blockset Reference )

## Related topics

Basics

Additional Workflow Steps for Rapid Prototyping Access to V-ECUs.......................................... 25
Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III (RTI Bypass Blockset Reference )
Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS (RTI Bypass Blockset Reference )
Rapid Prototyping Access to V-ECU-Internal Variables and Functions....................................... 40

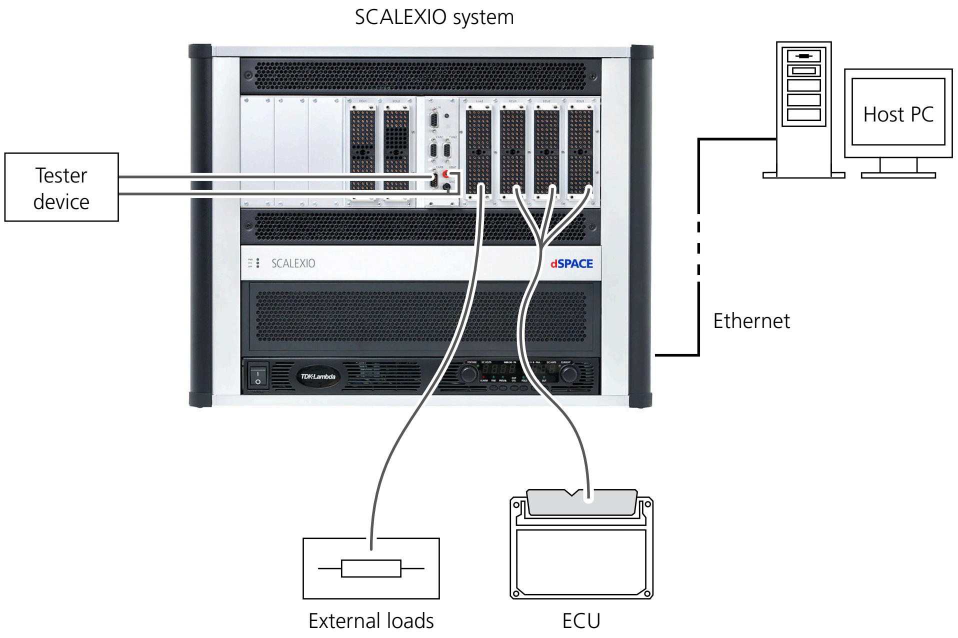
## SCALEXIO

**Introduction**

SCALEXIO systems are simulators for hardware-in-the-loop (HIL) simulation. They are used to test electronic control units (ECUs).

**Field of application and product overview**

**Field of application** SCALEXIO systems are designed for testing ECUs in a hardware-in-the-loop simulation. The following illustration shows an example of a SCALEXIO system that is connected to the ECU, external loads, a tester device, and the host PC.



SCALEXIO runs simulations *in real time*. For details on real-time simulation, refer to Basics on Real-Time Simulation with SCALEXIO/MicroAutoBox III on page 30.

**Product overview** A SCALEXIO system simulates the ECU-controlled system. The SCALEXIO system can:

- Generate the sensor signals which are required by the ECU
- Measure the signals which are generated by the ECU for actuator control
- Connect the output signals of the ECU to loads (internally in the rack or externally) to simulate the actuators
- Receive bus signals which are sent by the ECU and send bus signals to the ECU
- Provide the battery voltage to the ECU
- Simulate failures in the wiring of the ECU

The behavior of the controlled system is specified by a real-time application which runs on the SCALEXIO Processing Unit. The real-time application is implemented on the host PC and downloaded to the SCALEXIO system via Ethernet.

**Integration into the SIL testing tool chain**

**Overview** For an overview of how SCALEXIO is integrated in the SIL testing tool chain, refer to Experimenting and testing with the simulation system on page 24.

**Platform for real-time simulation** SCALEXIO is a system that supports the real-time simulation of simulation systems consisting of V-ECUs and environment models. It also provides the physical interface for connecting V-ECUs and environment models to real ECUs. The V-ECUs are used to mimic the control functions and restbus behavior in the environment of the real ECU(s) to be tested.

**Related topics**

Basics

Additional Workflow Steps for Rapid Prototyping Access to V-ECUs........................................ 25
Basics on Real-Time Simulation with SCALEXIO/MicroAutoBox III.............................................. 30
Tool Chain and Workflow for SIL Testing.................................................................................. 21

## SYNECT

**Introduction**

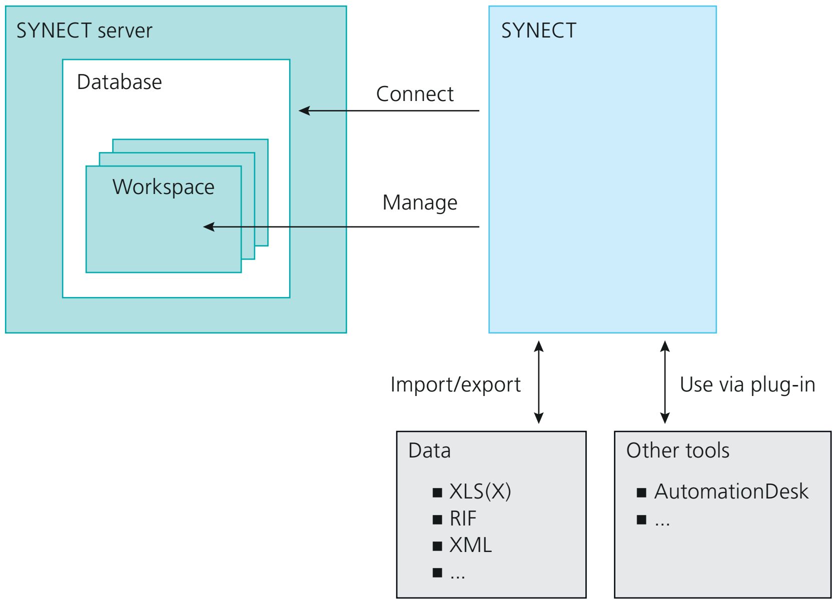
SYNECT is the dSPACE data management and collaboration software with a special focus on model-based development and ECU testing.

**Field of application and product overview**

**Field of application** SYNECT lets you manage data throughout the entire process of model-based development and ECU testing. This data can include models, signals, parameters, tests, test results, and more. SYNECT also handles data dependencies, versions, and variants, as well as links to the underlying requirements.

SYNECT is directly connected to engineering tools, such as MATLAB®, Simulink®, TargetLink, or AutomationDesk, and application/product life cycle management systems (ALM/PLM) so you can work with your preferred tools and seamlessly exchange data.

**Product overview** SYNECT provides a server for working with a central database. With SYNECT's client, you can connect to the server database and import, manage, and export database items in a workspace.



SYNECT provides various modules such as the Test Management, Signal and Parameter Management, and Model Management modules, each of which applies to a different step in model-based development and ECU testing.

## Integration into the SIL testing tool chain

**Overview** SYNECT lets you integrate system models.

**Integrating system models** With SYNECT's Model Management module, you can integrate system models, i.e., models of an ECU network and its environment.

To integrate the individual system model components, SYNECT lets you perform the following steps:

- Import the container files such as FMU files that implement the related system model components

  **Note**

  The SYNECT add-on for System Model Integration lets you import FMU 2.0 containers and SIC, BSC, V-ECU containers from dSPACE Release 2020-A and earlier. To import model containers from later releases, contact dSPACE Support.

- Connect model components
- Export the system model to exchange it with other SYNECT users or to build it using VEOS Player
- Build the system model in conjunction with VEOS for offline simulation

For more information, refer to Basics on Creating System Models (SYNECT Guide 🕮) and Building System Models for VEOS Simulation (SYNECT Guide 🕮).

**Related topics**

Basics

Basics on Creating System Models (SYNECT Guide 🕮)
Building System Models for VEOS Simulation (SYNECT Guide 🕮)
Tool Chain and Workflow for SIL Testing.................................................................. 21

## SystemDesk

**Introduction**

SystemDesk is a tool that lets you model software architectures according to AUTOSAR and create individual V-ECU implementation containers to be used in offline and real-time simulations for SIL testing.

**Product overview**

**Product overview** SystemDesk lets you perform the following:

- AUTOSAR modeling:
  - Software architecture modeling
  - System modeling
  - ECU configuration
- Creation of V-ECU implementations to be used in offline and real-time simulations

**Integration into the SIL testing tool chain**

**Overview** For an overview of how SystemDesk is integrated into the SIL testing tool chain, refer to Developing the simulation system on page 22.

**Interoperability of SystemDesk and TargetLink** SystemDesk lets you integrate software components that were implemented by behavior modeling tools, such as TargetLink.

For more information, refer to Exchanging SWC Containers Between SystemDesk and TargetLink (SystemDesk Manual 🕮).

**Creating V-ECU implementations** SystemDesk lets you create V-ECU implementations in the following ways:

- Classic V-ECUs

  You can create the following kinds of classic V-ECUs:
  - Model-based V-ECUs. A model-based V-ECU is based on an AUTOSAR model.
  - Code-based V-ECUs. A code-based V-ECU is based on externally created code.

You can integrate external code created with your own AUTOSAR tool chain, or non-AUTOSAR legacy code. Legacy code can be integrated in a V-ECU by using the dSPACE Legacy Code Intergrator (LCI), or by using C APIs directly.

- Adaptive V-ECUs developed according to the AUTOSAR Adaptive Platform

For more information, refer to Creating V-ECUs for Virtual Validation (SystemDesk Manual) and Creating Adaptive V-ECUs (SystemDesk Manual).

**Related topics**

Basics

## TargetLink

**Introduction**

TargetLink is a production code generator which produces highly efficient C code from Simulink/Stateflow models. Moreover, with the TargetLink AUTOSAR Module, you can generate software components according to AUTOSAR and exchange them, e.g., with SystemDesk. For SIL testing, TargetLink supports the generation of V-ECU implementations/V-ECUS directly from TargetLink models.

**Field of application and product overview**

**Field of application** TargetLink serves to implement graphically modeled control algorithms in the form of highly efficient C code for deployment on production ECUs. Following this paradigm of model-based design and automatic production code generation with Simulink/TargetLink, you can design, implement and test your control algorithms more efficiently, rapidly and at higher quality.

TargetLink meets the stringent requirements imposed on production code like efficiency, readability and configurability to produce the kind of code that is required.

TargetLink also lets you implement Simulink/TargetLink models in the form of AUTOSAR software components or adaptive applications, and test them in model-in-the-loop (MIL), software-in-the-loop (SIL) and processor-in-the-loop (PIL) simulations. On top of that, TargetLink supports the generation of V-ECU implementations from TargetLink models to run and simulate code generated by TargetLink on VEOS, SCALEXIO, and a MicroAutoBox III.

**Product overview** TargetLink provides the following core capabilities and features:

- Efficient fixed-point or floating-point production code generation directly from Simulink®/Stateflow®
- Comprehensive fixed-point support including autoscaling to simplify the transfer of floating-point algorithms into fixed-point code
- Direct verification of automatically generated code via MIL/SIL/PIL simulation with integrated data logging and plotting
- Support for modular, component-based development
- Efficient data management with a Data Dictionary enabling data exchange with other tools
- Compliance with standards like AUTOSAR, MISRA, OSEK, ISO 26262, ASAM
- Powerful integrated AUTOSAR support for the development of AUTOSAR software components and adaptive applications
- Certified code generation for ISO 26262, IEC 61508 and derived standards

For detailed information on TargetLink, refer to the TargetLink Orientation and Overview Guide .

## Integration into the SIL testing tool chain

**Overview** For an overview of TargetLink is integrated into the SIL testing tool chain, refer to Developing the simulation system on page 22.

**Classic AUTOSAR: Implementing software components** In connection with the TargetLink AUTOSAR Module, TargetLink lets you implement software components according to the AUTOSAR Classic Platform: TargetLink generates the AUTOSAR-compliant production C code and the AUTOSAR software component description.

For information on approaches to modeling software components using SystemDesk in combination with the TargetLink AUTOSAR Module, refer to Development Approaches with the TargetLink Classic AUTOSAR Module (TargetLink Classic AUTOSAR Modeling Guide ).

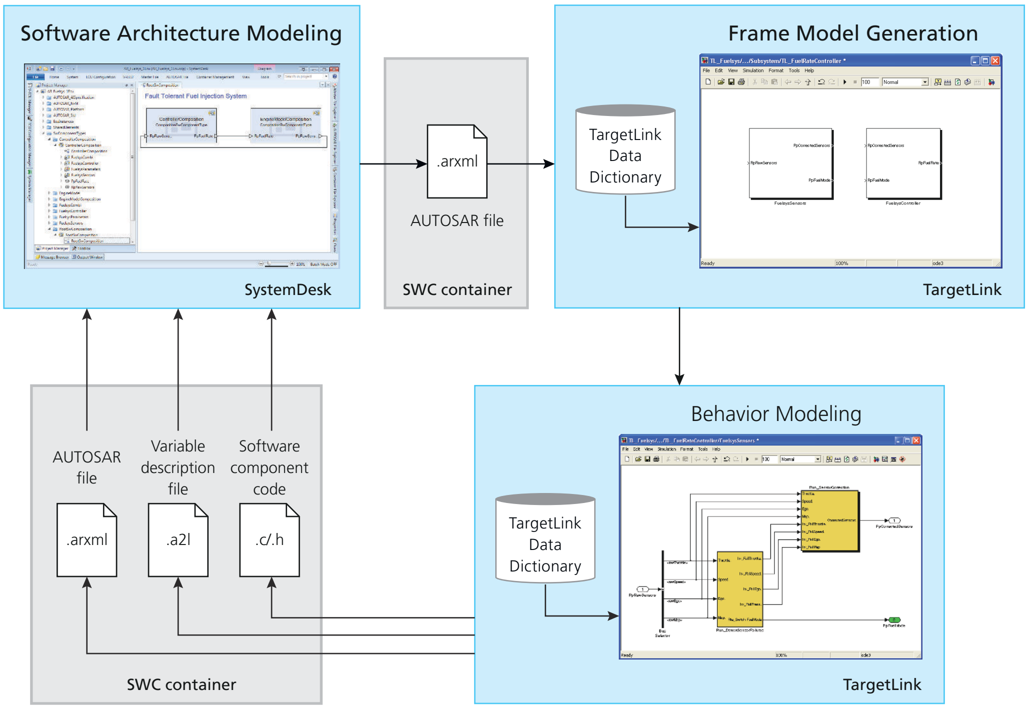
**Classic AUTOSAR: Integrating a software component as an SWC internal behavior in SystemDesk** You can integrate a software component implemented with TargetLink as an SWC internal behavior in SystemDesk. To integrate an SWC internal behavior, you can use *SWC containers*, which allow you to exchange data between SystemDesk and TargetLink. SWC containers are bundles of software-component-related files such as AUTOSAR files (ARXML), code files (H/C), and variable description files (A2L).

**Tip**

In SystemDesk-TargetLink roundtrips, you can use dSPACE AUTOSAR Compare to compare and merge AUTOSAR ARXML files.

After integrating the software component into an AUTOSAR system, you can use SystemDesk to generate the V-ECU implementation.

The illustration below shows an example of the interoperation of SystemDesk and TargetLink:



For information on exchanging SWC containers, refer to Exchanging SWC Containers Between SystemDesk and TargetLink (SystemDesk Manual 📖).

**Classic AUTOSAR: Creating V-ECU implementations** TargetLink lets you wrap TargetLink-generated code in a V-ECU implementation (VECU). This lets you test TargetLink-generated code not only by using the built-in MIL/SIL/PIL simulation, but also by running TargetLink-generated code in the SIL testing tool chain.

You can export a V-ECU implementation from TargetLink, and:

- Import it into VEOS Player to run TargetLink-generated code on VEOS.
- Import it into ConfigurationDesk to run TargetLink-generated code on SCALEXIO or a MicroAutoBox III.

**Note**

You can generate V-ECU implementations from TargetLink for both the non-AUTOSAR and the AUTOSAR use case. If you develop ECU functions consisting of multiple AUTOSAR software components, it is recommended that you integrate them into an AUTOSAR software architecture in SystemDesk, because the Simulink environment does not provide all of the capabilities needed to interconnect multiple software components to an AUTOSAR system.

For more information, refer to Basics on Interoperating with Other dSPACE Tools for Virtual Validation (TargetLink Interoperation and Exchange Guide 🕮).

**Adaptive AUTOSAR: Developing functional parts of adaptive applications** TargetLink lets you develop functional parts of adaptive applications according to the AUTOSAR Adaptive Platform with a model-based approach.

For more information, refer to Introduction to Using TargetLink With Adaptive AUTOSAR (TargetLink Adaptive AUTOSAR Modeling Guide 🕮).

**Creating Simulink implementation containers** TargetLink also lets you generate Simulink implementation container (SIC) files. This lets you execute the production code generated by TargetLink on VEOS for offline simulation and on SCALEXIO or a MicroAutoBox III for real-time simulation.

For details on generating an SIC file with TargetLink, refer to Basics on Interoperating with ConfigurationDesk via SIC Files (TargetLink Interoperation and Exchange Guide 🕮).

**Creating FMUs** TargetLink lets you export the generated code as a Functional Mock-up Unit (FMU) for usage with third-party simulators.

For details on exporting an FMU file with TargetLink, refer to Basics on Exporting FMUs from TargetLink (TargetLink Interoperation and Exchange Guide 🕮).

**Related topics**

Basics

Basics on Exporting FMUs from TargetLink (TargetLink Interoperation and Exchange Guide 🕮)
Basics on Interoperating with ConfigurationDesk via SIC Files (TargetLink Interoperation and Exchange Guide 🕮)
Basics on Interoperating with Other dSPACE Tools for Virtual Validation (TargetLink Interoperation and Exchange Guide 🕮)
Development Approaches with the TargetLink Classic AUTOSAR Module (TargetLink Classic AUTOSAR Modeling Guide 🕮)
Exchanging SWC Containers Between SystemDesk and TargetLink (SystemDesk Manual 🕮)
Introduction to Using TargetLink With Adaptive AUTOSAR (TargetLink Adaptive AUTOSAR Modeling Guide 🕮)
Tool Chain and Workflow for SIL Testing........................................................................... 21

## VEOS

**Introduction**

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

**Field of application and product overview**

**Field of application** VEOS supports testing ECU software in closed loop with environment models during PC-based *offline simulations* of simulation systems. For details on offline simulation, refer to Basics on Offline Simulation with VEOS on page 28.

**Product overview** VEOS consists of the following components:

| **Component** | **Description** |
|---|---|
| VEOS System Builder | The VEOS component for:<br>▪ Building a model implementation for offline simulation on VEOS.<br>▪ Integrating these model implementations into a simulation system. |
| VEOS Simulator | The VEOS component for:<br>▪ Controlling the execution of offline simulations.<br>▪ Interconnecting all the VPUs of a simulation system during offline simulation.<br>VEOS Simulator is executed in a separate task of the PC's operating system. |
| VEOS Player | The graphical user interface for VEOS. |

**Integration into the SIL testing tool chain**

**Overview** For an overview of how VEOS is integrated into the SIL testing tool chain, refer to Integrating and building the simulation system on page 23 and Experimenting and testing with the simulation system on page 24.

**Integrating V-ECUs, environment models, and bus models** VEOS lets you integrate V-ECUs, environment models, and bus models into a simulation system for offline simulation.

For more information, refer to Basics on Integrating the Simulation System (VEOS Manual).

**Platform for offline simulation** VEOS serves as a platform for offline simulation on Windows and on Linux. VEOS offers several services to access variables during simulations, e.g., for measurement, calibration, and stimulation.

For more information, refer to Loading and Running an Offline Simulation Application (VEOS Manual).

VEOS also provides a specific Linux kernel for running Linux-based applications, such as adaptive V-ECUs. They are executed in a hypervisor, which is integrated in the VEOS simulation system.

For more information, refer to Basics on Simulating Adaptive V-ECUs (VEOS Manual).

**Related topics**

Basics

Additional Workflow Steps for Rapid Prototyping Access to V-ECUs.......................... 25
Basics on Integrating the Simulation System (VEOS Manual 🕮)
Basics on Offline Simulation with VEOS.......................... 28
Basics on Simulating Adaptive V-ECUs (VEOS Manual 🕮)
Loading and Running an Offline Simulation Application (VEOS Manual 🕮)
Tool Chain and Workflow for SIL Testing.......................... 21

# Glossary

**Introduction**

Briefly explains the most important expressions and naming conventions used in the context of SIL testing.

**Where to go from here**

**Information in this section**

## 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 process** A component of an executable application for a dSPACE simulator, i.e., a real-time application (RTA) for SCALEXIO or a MicroAutoBox III, or an offline simulation application (OSA) for VEOS.

**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.

**Automation** A communication mechanism that can be used by various programming languages. A client can use it to control a server by calling methods and properties of the server's automation interface.

**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.

**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 VPU** The executable of a bus model built for the VEOS platform. A bus VPU is part of an offline simulation application (OSA).

**Bypassing** A method for replacing an existing ECU function by running a new function.

## 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.

**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).

## 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.

## E

**ECU** Abbreviation of *electronic control unit*.

**ECU interfacing** Methods and tools to synchronously read and/or write individual variables of an ECU application.

**Environment model** A model that represents a part or all of the ECU's environment in a simulation scenario.

**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

**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

## M

**MicroAutoBox III** A system that is used in real-time simulation for fast in-vehicle function prototyping.

**Microcontroller abstraction layer (MCAL)** The lowest layer of the ECU software according to the AUTOSAR Classic Platform standard. It is microcontroller-dependent and contains drivers for enabling the access of on-chip peripheral devices of a microcontroller and off-chip memory-mapped peripheral devices by a defined API. The purpose of the MCAL is to make higher software layers independent of the microcontroller.

**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.

## 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.

**OSA** Abbreviation of offline simulation application (OSA)

## P

**Platform** A software component representing a simulator where a simulation application is computed in real-time (on dSPACE real-time hardware) or in non-real-time (on VEOS).

## R

**Real-time application (RTA)** A real-time application (RTA) file is an executable file for the processing units of dSPACE real-time hardware. After the build process with ConfigurationDesk, the RTA file can be downloaded to the hardware.

An RTA can contain multiple application processes, each of which contains either a V-ECU or environment model. Each application process runs on a separate core of the real-time hardware.

**Real-time simulation** A simulation where the simulation system is executed synchronously to the world time with guaranteed deadlines.

**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.

**RTA** Abbreviation of real-time application (RTA)

**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

**SCALEXIO system** A system that is used in real-time simulation for developing and testing electronic control units (ECUs).

**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.

**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).

**SIL testing** Abbreviation of *software-in-the-loop testing*.

Simulation and testing of individual software functions, complete virtual ECUs (V-ECUs), or even V-ECU networks on a local PC or highly parallel in the cloud independently of real-time constraints and real hardware.

**Simulation application** The generic term for offline simulation application (OSA) and real-time application (RTA).

**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 ECU** A soft ECU realizes a simplified function model, which is often used as a part of an environment model. Soft ECUs are not intended for production code generation.

**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

**TRC file** A variable description file with information on the variables available in an environment model running on a dSPACE platform.

## V

**Variable description** A file describing the variables in a simulation application, which are available for measurement, calibration, and stimulation.

**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.

**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 Player is the graphical user interface for VEOS.

**VEOS Player** An application running on the host PC for editing, configuring and controlling an offline simulation application (OSA) running in VEOS.

**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.

# Appendix

**Where to go from here**

Information in this section

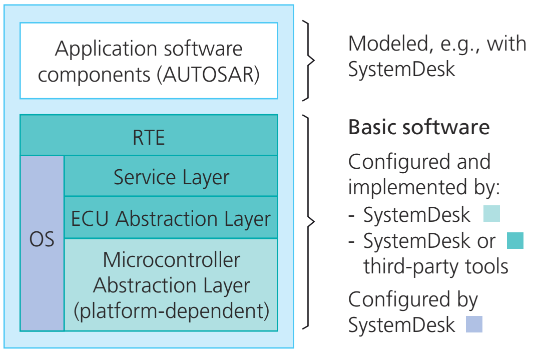
## Basic Software Module Support for Classic V-ECUs

**Introduction**

Provides an overview of the basic software module support for AUTOSAR and non-AUTOSAR classic V-ECUs.

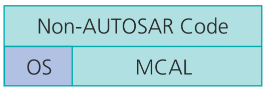
**Software layers of an AUTOSAR classic V-ECU** The following illustration shows the software layers of an AUTOSAR classic V-ECU:

AUTOSAR Classic V-ECU


Application software components (AUTOSAR)
RTE
Service Layer
ECU Abstraction Layer
OS
Microcontroller Abstraction Layer (platform-dependent)
Modeled, e.g., with SystemDesk
Basic software
Configured and implemented by:
- SystemDesk
- SystemDesk or third-party tools
Configured by SystemDesk

**Software layers of a non-AUTOSAR classic V-ECU** You can also build non-AUTOSAR classic V-ECUs by integrating generic C code on top of the provided OS and microcontroller abstraction layer (MCAL).

The following illustration shows the software layers of a non-AUTOSAR classic V-ECU:



**Supported basic software modules**

The following table provides an overview of the supported basic software modules, depending on the simulation platform:

| Basic Software Module | | Configuration (C) and Implementation (I) with SystemDesk | Implementation (I), Build (B), and Simulation (S) on ... ... VEOS | ... SCALEXIO and MicroAutoBox III[1] |
|---|---|---|---|---|
| **Service Layer and ECU Abstraction Layer, i.e., above Microcontroller Abstraction Layer (MCAL)** | | | | |
| **AUTOSAR Modules Implemented by dSPACE[2]** | | | | |
| `CanIf`[3), 4)] | Performs hardware access to the CAN bus.<br>For more information, refer to CAN Interface (CanIf) (SystemDesk Manual 📖). | C | I, B, S | I, B, S |
| `Com` | Manages network communication between ECUs.<br>For more information, refer to Communication (Com) (SystemDesk Manual 📖). | C, I | B, S | B, S |
| `Det` | Traces and logs errors during development.<br>For more information, refer to Default Error Tracer (Det) (SystemDesk Manual 📖). | C, I | B, S | B, S |
| `EcuC` | Contains the configuration parameters and subcontainers of the global PDU collection.<br>For more information, refer to ECU Configuration (EcuC) (SystemDesk Manual 📖). | C, I | B, S | B, S |
| `EcuM` | Handles the power and mode management of the ECU.<br>For more information, refer to ECU State Manager (EcuM) (SystemDesk Manual 📖). | C, I | B, S | B, S |
| `IPduM` | Manages network communication between ECUs.<br>For more information, refer to IPDU Multiplexer (IPduM) (SystemDesk Manual 📖). | C, I | B, S | B, S |
| `LinIf`[3), 4)] | Performs hardware access to the LIN bus.<br>For more information, refer to LIN Interface (LinIf) (SystemDesk Manual 📖). | C | I, B, S | I, B, S |
| `MemMap` | Maps code and variables to ECU memory to avoid RAM wastage, use specific RAM properties, or protect memory.<br>For more information, refer to Memory Mapping (MemMap) (SystemDesk Manual 📖). | C, I[5] | (B, S) | (B, S) |
| `NvM` | Lets application software components and basic software components access nonvolatile memory such as EEPROM or flash.<br>For more information, refer to NVRAM Manager (NvM) (SystemDesk Manual 📖). | C, I | B, S[6] | B, S[6] |

| Basic Software Module | | Configuration (C) and Implementation (I) with SystemDesk | Implementation (I), Build (B), and Simulation (S) on ... ... VEOS | ... SCALEXIO and MicroAutoBox III[1] |
|---|---|---|---|---|
| `OS` | Manages the operating system resources of a V-ECU such as tasks, alarms, counters, and events.<br>For more information, refer to Operating System (Os) (SystemDesk Manual 🕮). | C[7] | I, B, S | I, B, S |
| `PduR` | Manages network communication between ECUs.<br>For more information, refer to PDU Router (PduR) (SystemDesk Manual 🕮). | C, I | B, S | B, S |
| `Rte` | Connects the ECU application software to the basic software, and also interconnects the different application software components.<br>For more information, refer to Run-Time Environment (Rte) (SystemDesk Manual 🕮). | C, I | B, S | B, S |
| **AUTOSAR Modules Implemented by Third-Party Tools** | | | | |
| Various modules such as `Dem`, `Dcm` | | C, I[8] | B, S[9] | - |
| **dSPACE-Specific Modules (Non-AUTOSAR)** | | | | |
| `Apu` | For emulating the driver hardware used for the detection of crankshaft/camshaft signals.<br>For more information, refer to Angular Processing Unit (Apu) (SystemDesk Manual 🕮). | C, I | B, S | - |
| `Dap` | For connecting V-ECUs with other V-ECUs or the I/O hardware via data access points.<br>For more information, refer to Data Access Points (Dap) (SystemDesk Manual 🕮). | C | I, B, S | I, B, S |
| `dSIdBusIf`[4] | For configuring generic bus simulation with idealized behavior.<br>For more information, refer to dSPACE Idealized Bus Interface (dSIdBusIf) (SystemDesk Manual 🕮). | C | I, B, S | - |
| `Edge` | For emulating the driver hardware used for the detection of rising/falling signal edges.<br>For more information, refer to Edge (Edge) (SystemDesk Manual 🕮). | C, I | B, S | - |
| ▪ `RptAccess`<br>▪ `RptSi` | For configuring the V-ECU application for rapid prototyping purposes.<br>▪ The Rapid Prototyping Access (RptAccess) module lets you prepare individual runnables for access from outside the V-ECU for rapid prototyping purposes. To provide access to these | C | I, B, S[10] | I, B, S[11] |

| Basic Software Module | | Configuration (C) and Implementation (I) with SystemDesk | Implementation (I), Build (B), and Simulation (S) on ... ... VEOS | ... SCALEXIO and MicroAutoBox III[1] |
|---|---|---|---|---|
| | runnables, corresponding calls to the dSPACE Internal Bypassing Service are integrated in the V-ECU application. These service calls can be used in bypass models based on the RTI Bypass Blockset.<br>▪ The Rapid Prototyping Service Interface (RptSi) module lets you configure the instances of the dSPACE Internal Bypassing Service to be integrated in the V-ECU application.<br>For more information, refer to Basics on Rapid Prototyping (SystemDesk Manual). | | | |
| `Sab` | For configuring the simulator and starting the ECU state manager in simulations.<br>For more information, refer to Simulator Abstraction (Sab) (SystemDesk Manual). | C | I, B, S | I, B, S |
| `Sfc` | For connecting 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 such as FlexRay.<br>For more information, refer to Signal-to-Frame Converter (Sfc) (SystemDesk Manual). | C, I | B, S | - |
| `Shu` | For emulating the reading of signals at specific times.<br>For more information, refer to Sample and hold unit (Shu) (SystemDesk Manual). | C, I | B, S | - |
| `SpiDev` | For emulating a serial peripheral interface (SPI) device.<br>For more information, refer to Serial Peripheral Device (SpiDev) (SystemDesk Manual). | C[12] | B, S | - |
| **Microcontroller abstraction layer (MCAL)** | | | | |
| **Microcontroller Drivers** | | | | |
| `Gpt` | Provides functions to initialize and control the ECU-internal general-purpose timer. The microcontroller unit driver (`Mcu`) is referenced to access hardware clock settings. Modules such as the operating | C, I | B, S | - |

| **Basic Software Module** | | **Configuration (C) and Implementation (I) with SystemDesk** | **Implementation (I), Build (B), and Simulation (S) on ...** | |
|---|---|---|---|---|
| | | | **... VEOS** | **... SCALEXIO and MicroAutoBox III**[1] |
| | system (`Os`) reference channels of the general-purpose timer.<br>For more information, refer to General-Purpose Timer (Gpt) (SystemDesk Manual 🕮). | | | |
| `Mcu` | Provides functions for microcontroller initialization, power down functionality, reset, and microcontroller-specific functions that are required by other basic software modules of the microcontroller abstraction layer.<br>For more information, refer to Microcontroller Unit (Mcu) (SystemDesk Manual 🕮). | C, (I[5]) | (B, S) | - |
| `Wdg` | Provides functions to control the watchdog, which controls hardware timings of microcontoller initialization and mode switches.<br>For more information, refer to Watchdog Driver (Wdg) (SystemDesk Manual 🕮). | C, I | B, S | - |
| **Memory Drivers** | | | | |
| `Eep`[13] | Provides functions to initialize, read, write, and erase the internal EEPROM (electrically erasable programmable read-only memory).<br>For more information, refer to EEPROM Driver (Eep) (SystemDesk Manual 🕮). | C, I | B, S | - |
| `Fls`[13] | Provides functions to initialize, read, write, and erase flash memory.<br>For more information, refer to Flash Driver (Fls) (SystemDesk Manual 🕮). | C, I | B, S | - |
| **Communication Drivers**[4] | | | | |
| `Can` | Performs hardware access to the CAN bus.<br>For more information, refer to CAN Bus Driver (Can) (SystemDesk Manual 🕮). | C, I | B, S | B, S |
| `Eth` | Performs hardware access to the Ethernet.<br>For more information, refer to Ethernet Driver (Eth) (SystemDesk Manual 🕮). | C, I | B, S | - |
| `Lin` | Performs hardware access to the LIN bus.<br>For more information, refer to LIN Driver (Lin) (SystemDesk Manual 🕮). | C, I | B, S | B, S |
| `Spi` | Performs hardware access to the serial peripheral interface.<br>For more information, refer to Serial Peripheral Interface Handler/Driver (Spi) (SystemDesk Manual 🕮). | C, I | B,S | - |

| Basic Software Module | | Configuration (C) and Implementation (I) with SystemDesk | Implementation (I), Build (B), and Simulation (S) on ... ... VEOS | ... SCALEXIO and MicroAutoBox III[1] |
|---|---|---|---|---|
| **I/O Drivers** | | | | |
| Adc[14] | Provides functions to access analog/digital converter channels of the microcontroller. For more information, refer to Analog-to-Digital Converter Driver (Adc) (SystemDesk Manual 📖). | C, I | B, S | - |
| Dio[14] | Provides functions to read/write the digital I/O of the microcontroller, i.e., the pins of a digital I/O port. The digital I/O driver references port pins that are configured and initialized by the Port driver. For more information, refer to Digital I/O (Dio) (SystemDesk Manual 📖). | C, I | B, S | - |
| Icu[14] | Provides functions to capture and demodulate pulse width modulation (PWM) signals, count pulses, measure the frequency and duty cycle, and generate simple and wake-up interrupts. For more information, refer to Input Capture Unit Driver (Icu) (SystemDesk Manual 📖). | C, I | B, S | - |
| Ocu[14] | Provides functions to compare the value of a counter with a defined threshold, and respond automatically when the threshold is reached. For more information, refer to Output Compare Unit Driver (Ocu) (SystemDesk Manual 📖). | C, I | B, S | - |
| Port | Provides functions to initialize and configure the ports and pins of the microcontroller. For more information, refer to Port Driver (Port) (SystemDesk Manual 📖). | C, I | B, S | - |
| Pwm[14] | Provides functions to initialize and control microcontroller PWM (pulse width modulation) channels. For more information, refer to Pulse Width Modulation Driver (Pwm) (SystemDesk Manual 📖). | C, I | B, S | - |

1) See SCALEXIO and MicroAutoBox III: limited support for simulation of third-party basic software on page 108.
2) The modules are limited with regard to the AUTOSAR module specifications.
3) Partially based on AUTOSAR 3.
4) Only specific combinations of the basic software modules for the communication of a V-ECU are supported. For more information, refer to Basic Software Modules for Virtual Validation (SystemDesk Manual 📖).
5) Stub implementation, not relevant for simulation behavior
6) Data storage in RAM
7) Single-core OS module

| Basic Software Module | Configuration (C) and Implementation (I) with SystemDesk | Implementation (I), Build (B), and Simulation (S) on ... | |
|---|---|---|---|
| | | ... VEOS | ... SCALEXIO and MicroAutoBox III[1)] |

[8)] Support via SystemDesk's ECU Configuration Framework.
[9)] Modules are supported if they fit on top of the provided microcontroller abstraction layer (MCAL)
[10)] For more information, refer to Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on VEOS (RTI Bypass Blockset Reference 📖).
[11)] For more information, refer to Dynamically Integrating a Simulink or TargetLink Model into a V-ECU Running on a SCALEXIO System or on a MicroAutoBox III (RTI Bypass Blockset Reference 📖).
[12)] You have to provide an SPI device implementation for simulation. For more information, refer to https://www.dspace.com/go/CodeBasedVECU.
[13)] Implemented by using the V-ECU RAM
[14)] Mapped internally to the `Dap` module

**SCALEXIO and MicroAutoBox III: limited support for simulation of third-party basic software**

V-ECU basic software usually depends on the microcontroller abstraction layer (MCAL). However, as shown in the table above, SCALEXIO and the MicroAutoBox III support only few modules of the microcontroller abstraction layer (MCAL). As a consequence, the simulation of V-ECUs *that contain third-party basic software* is limited on SCALEXIO and the MicroAutoBox III.

**Related topics**

Basics

Basics on Classic V-ECUs........................................................................................................ 14
Configuring ECUs (SystemDesk Manual 📖)
Using dSPACE Basic Software Modules for Virtual Validation (SystemDesk Manual 📖)

## Avoiding Naming Conflicts when Integrating Custom C Code

**Introduction**

You can integrate custom C code in a model implementation such as a V-ECU implementation. For example, to avoid naming conflicts during the build process of a V-ECU implementation, you must use symbol prefixes and names in your custom C code that are different from the prefixes and names used by SystemDesk's V-ECU generation.

The symbol prefixes and names to avoid depend on the model implementation type and the simulation target. Whether naming conflicts occur also depends on the header files that you actually include in your custom C code.

**Note**

The following lists cover the most important symbol prefixes and names that you must avoid in custom C code. The lists are not exhaustive.

**Prefixes and names to avoid in all model implementations**

There is a number of C symbol prefixes and names that you must not use for user-defined C symbols in all model implementations to avoid naming conflicts.

**Prefixes and names to avoid on all simulation platforms**

- You have to avoid reserved keywords of the C/C++ standard.
- Do not use the names of symbols from standard header files and linked libraries.
- If the model implementation contains code generated by TargetLink, do not use the symbols defined in `dsfxp.h`

**Prefixes and names to avoid when building an offline simulation application for VEOS** There is a risk of naming conflicts for the following prefixes when you build an offline simulation application for VEOS:

- `BscAppFrm`
- `dsrtt`
- `ds`, `Ds`, and `DS`
- `DsDaq`
- `FmiAppFrm`
- `g_`, `ga_`, and `gp_`
  VEOS uses these prefixes for variables, arrays and pointers, respectively.
- `md_`
- `msg`
- `SicAppFrm`
- `SmartFeatureManagers`
- `VEcuAppFrm`
- `VEOS`
- `xcp`, `Xcp`, and `XCP`

**Prefixes and names to avoid when building a real-time application for SCALEXIO or MicroAutoBox III** There is a risk of naming conflicts for the following prefixes when you build a real-time application for SCALEXIO or MicroAutoBox III:

- `Ap`
- `ds`, `Ds`, and `DS`
- `md_`
- `Rtos`
- `VEcu`
- `xcp`, `Xcp`, and `XCP`
- Symbols defined by dSPACE RTLib for dSPACE I/O drivers

**Prefixes and names to avoid in the context of a V-ECU implementation**

To avoid naming conflicts in the context of a V-ECU implementation (VECU) file, there is a number of C symbol prefixes and names that you must not use for user-defined C symbols.

- Do not use the following C symbol prefixes, because they are reserved by the AUTOSAR standard or used by dSPACE products in the context of a V-ECU implementation (VECU) file if the related modules are integrated in the V-ECU:

| Reserved Prefix | Module |
|---|---|
| `Adc_` | Analog/Digital Converter module |
| `Apu_` | Angular processing unit module |
| `Can_` | CAN driver module |
| `CanIf_` | CAN Interface module |
| `Dap_` | Data Access Points module |
| `Det_` | Default Error Tracer module |
| `Dio_` | Digital I/O module |
| `Ds_` | dSPACE RTLIB |
| `DsIdBusIf_` | dSPACE Idealized Bus Interface module |
| `EcuM_` | ECU State Manager module |
| `Edge_` | Edge driver module |
| `Eep_` | EEPROM Abstraction module |
| `Eth_` | Ethernet driver module |
| `Fls_` | Flash driver module |
| `Gpt_` | General Purpose Timer driver module |
| `Icu_` | Input Capture Unit driver module |
| `Lin_` | LIN driver module |
| `LinIf_` | LIN Interface module |
| `Mcu_` | Mircocontroller Unit driver module |
| `Ocu_` | Output Compare Unit driver module |
| `Os_` | Operating system module |
| `Port_` | Port module |
| `Pwm_` | Pulse Width Modulation driver module |
| `RptSi_` | Rapid Prototyping Service Interface module |
| `Sab_` | Simulator Abstraction module |
| `Sfc_` | Signal-to-Frame Converter module |
| `Shu_` | Sample and hold module |
| `Spi_` | Serial Peripheral Interface module |
| `tl_` | TargetLink |
| `Wdg_` | Watchdog driver module |

- The following standard header files defined by AUTOSAR are included:
  - `Compiler.h`
  - `MemMap.h`

  - `Platform_Types.h`
  - `Std_Types.h`

  Do not use the C symbol names defined in these header files.
- Do not use the C symbol names defined in the AUTOSAR OS standard such as `ActivateTask()` since they are used by the operating system implementation.

  For more information on the AUTOSAR OS standard, refer to the *AUTOSAR_SWS_OS* document.

  You can find it at www.autosar.org.
- Do not use the C symbol names that are reserved by the AUTOSAR standard for basics software (BSW) modules.

  For more information, refer to the *AUTOSAR_TR_BSWModuleList* document.

  You can find it at www.autosar.org.

**Prefixes and names to avoid in the context of a Simulink implementation container and bus simulation container**

To avoid naming conflicts in the context of a Simulink implementation container (SIC) or bus simulation container (BSC) file, there is a number of C symbol prefixes and names that you must not use for user-defined C symbols.

Do not use the following prefixes:

- `Asm` and `ASM`
- `ap_`
- `DSRT`
- Symbols reserved by MathWorks® for S-functions such as the symbols defined in `simstruc.h` and `mex.h`
- Symbols used by Simulink® Coder™ during code generation such as the symbols defined in `rtwtypes.h` and `rt_*.h`
- `<modelName>_{In|Out}port_Block<n>_Port<m>` and `Bus_<modelName>_Data{In|Out}port_Block<n>_Port<m>`

| | |
|---|---|
| `<modelName>` | Simulink model name |
| `<n>` | Index of model port block |
| `<m>` | Index of model port block port |

**Prefixes and names to avoid in the context of a Functional Mock-up Unit**

To avoid naming conflicts in the context of a Functional Mock-up Unit (FMU) file, there is a number of C symbol prefixes and names that you must not use for user-defined C symbols.

- Do not use the C symbol names defined in the FMI standard.

  For more information on the FMI standard, refer to http://fmi-standard.org/downloads/.
- Do not use the `fmi2` prefix, because it is used for functions and variables of an FMU.

## Limitations

| | |
|---|---|
| **Limited basic software support** | The basic software support depends on the simulation platform. For more information, refer to Basic Software Module Support for Classic V-ECUs on page 101. |
| **Limitation for SYNECT** | The SYNECT add-on for System Model Integration lets you import FMU 2.0 containers and SIC, BSC, V-ECU containers from dSPACE Release 2020-A and earlier. To import model containers from later releases, contact dSPACE Support. |
| **Limitations for VEOS** | For information on the limitations for VEOS, refer to:<br>▪ Limitations for Integrating the Simulation System (VEOS Manual 🕮)<br>▪ Limitations for Loading and Running an Offline Simulation Application, and Debugging Source Code (VEOS Manual 🕮)<br>▪ Limitations for Automating the VEOS Player (VEOS Manual 🕮) |
| **Limitations for ConfigurationDesk and SCALEXIO/MicroAutoBox III** | For information on the limitations for ConfigurationDesk and SCALEXIO/MicroAutoBox III, refer to Limitations Concerning V-ECU Implementations (ConfigurationDesk Real-Time Implementation Guide 🕮). |