TargetLink

# Orientation and Overview Guide

For TargetLink 5.1

Release 2020-B – November 2020

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.

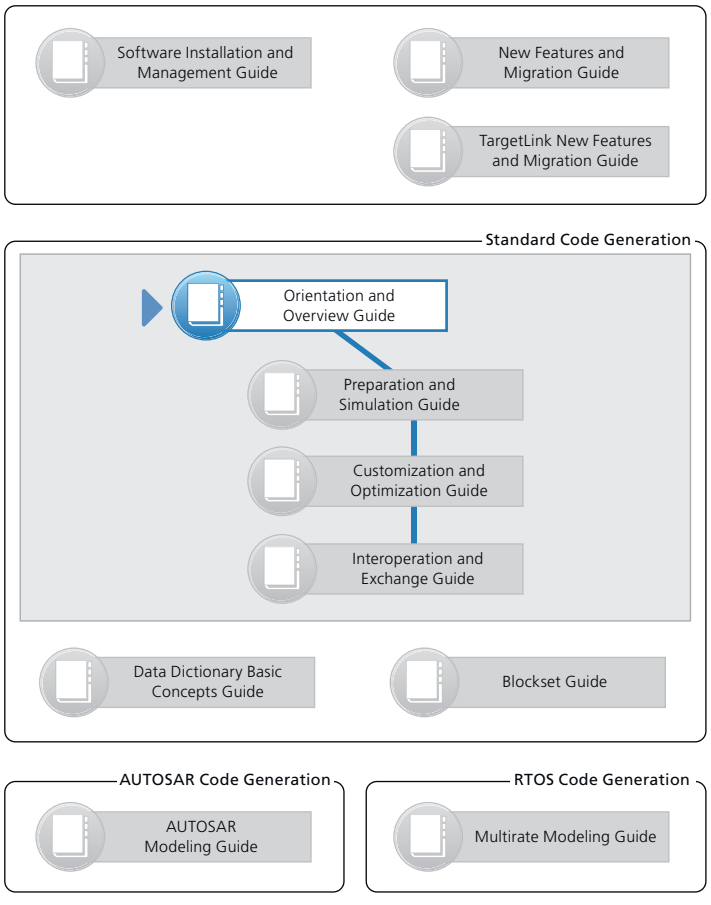
## Important Notice

# Contents

# About This Guide

**Contents**

This guide serves as the first information base when working with TargetLink, regardless of whether you are a beginner, intermediate, advanced, or expert user. It introduces you to TargetLink and helps you find appropriate user documentation with respect to your use case. Furthermore, it lists all the TargetLink limitations and contains a glossary.



## Required knowledge

This guide is most useful to software developers, TargetLink administrators, and dSPACE tool chain managers. Knowledge in handling the host PC, the Microsoft Windows operating system and MATLAB is a requirement.

## Symbols

dSPACE user documentation uses the following symbols:

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

## Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

## Special folders

Some software products use the following special folders:

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

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

or

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

**Documents folder** A standard folder for user-specific documents.

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

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

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

## Accessing dSPACE Help and PDF Files

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

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.

# Getting Started

**Introduction**

Before you start working with TargetLink, you must familiarize yourself with the most important process steps for generating production code, the associated user interface elements, and supported methods.

**Where to go from here**

Information in this section



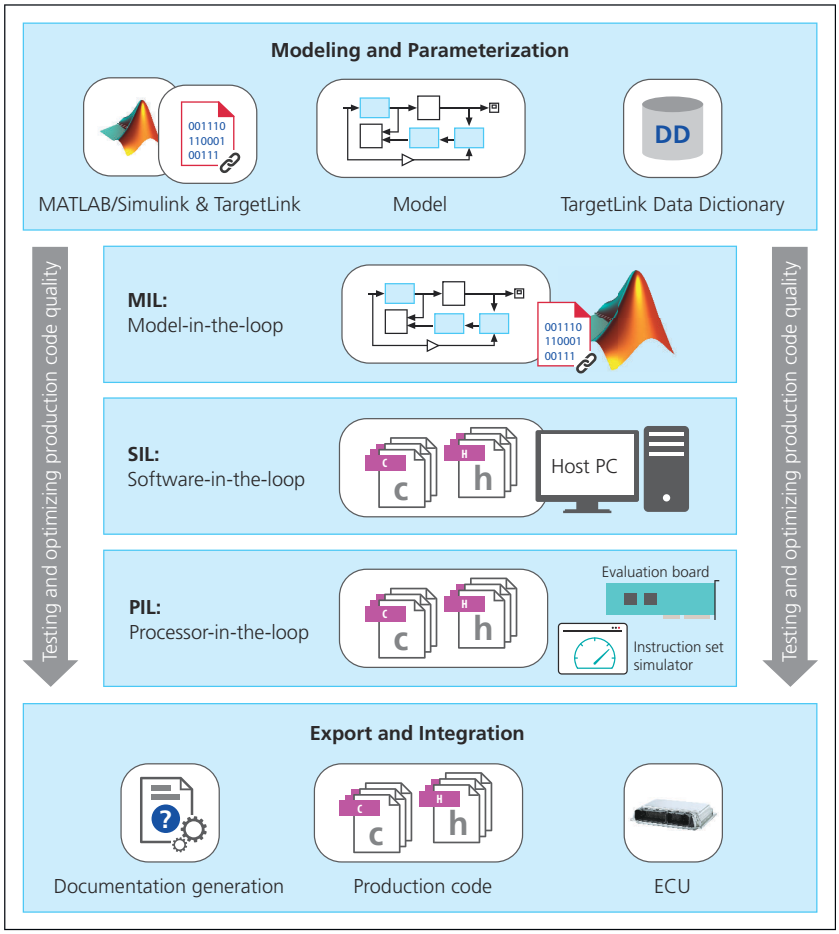
Information in other sections

Required MATLAB Releases (🕮 Installing dSPACE Software)

## Basics on Using TargetLink

**General process steps**

This topic shows a typical workflow for code generation with TargetLink and explains its individual components. The workflow is usually iterated multiple times.




**Comprehensive overview**

For a comprehensive overview of all TargetLink processes and use cases, refer to Overview of Standard TargetLink Use Cases on page 26.

**TargetLink at a glance**

Relevant symbol in the workflow illustration:

TargetLink generates production code (C code) from models created with MATLAB®/Simulink®/Stateflow®. The C code generation options of TargetLink range from plain ANSI C code to optimized fixed-point or floating-point code for AUTOSAR platforms. Versatile code configuration options ensure that the generated production code fits the target.

The built-in simulation features in MIL, SIL, and PIL simulation modes enable high code quality because each simulation step can be tested against the specification. TargetLink closes the gap between the specified model and the code implementation for the target.

Production code generated by TargetLink is proven to be as efficient as handwritten code. Other factors also make TargetLink a useful tool: code readability, traceable model/code dependency, and the ability to configure the code generation to produce precisely the required code.

**Demo models** You can use the provided demo models to become familiar with the operating principles of TargetLink. Refer to How to Open a Demo Model on page 17.

### Based on MATLAB/Simulink and Stateflow

Relevant symbol in the workflow illustration:

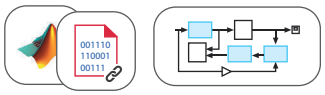
MATLAB/Simulink and Stateflow provide the calculation and simulation engine that is used to simulate the TargetLink model. TargetLink is based on MATLAB/Simulink and uses MATLAB editors to generate the model and to simulate it.

If you installed TargetLink and connected it to a supported MATLAB version in the dSPACE Installation Manager, a message similar to the following is displayed in the MATLAB Command Window after you start MATLAB:

```
================================================================================
Configuring dSPACE(R) Software for MATLAB(R) <Version> (<Release>) …
TargetLink     Production Code Generator     <Version>       <Date> okay
DSDD           Data Dictionary               <Version>       <Date> okay
================================================================================
```

### Creating TargetLink models

Relevant symbols in the workflow illustration:




TargetLink models are Simulink models where certain blocks carry additional data to specify production code settings. You can find all supported blocks in the TargetLink block library (tllib).

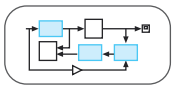
A simple TargetLink model contains at least one TargetLink **Subsystem** block and exactly one **MIL Handler** block.

You can create a model from scratch or prepare an existing Simulink model for code generation. Refer to:

- How to Create a Model from Scratch on page 18
- How to Prepare Simulink Systems for Code Generation with TargetLink (TargetLink Preparation and Simulation Guide)

**Modeling with TargetLink blocks**

Relevant symbol in the workflow illustration:



In the TargetLink Subsystem block, you can implement your control algorithm by using blocks from the TargetLink block library (tllib).

You can add TargetLink blocks to your model with one of the following methods:

- Typing the name of a TargetLink block directly in the Simulink Editor.
- Dragging a TargetLink block from the TargetLink block library to the Simulink Editor. Refer to How to Open the TargetLink Block Library on page 16.
- Using the **add_block** API command.

```
add_block('tllib/Gain','<ModelName>/<SubsystemName>/Gain')
```

**Specifying block properties**

TargetLink blocks provide additional data called block properties. To modify block properties. Refer to How to Modify Block Properties via Block Dialogs (TargetLink Preparation and Simulation Guide).

You can modify multiple block properties at once via the Property Manager. Refer to Modifying Multiple Properties at Once via the Property Manager (TargetLink Preparation and Simulation Guide).

**Globally specifying data in the TargetLink Data Dictionary**

Relevant symbol in the workflow illustration:


You can globally specify data with a fixed set of block properties via the TargetLink Data Dictionary. Managing data specifications globally helps you as follows:

- Keeping data consistent across development stages.
- Separating data from the control algorithm.
- Using the data in different models.
- Sharing the data with team members.

For more information, refer to:

- Basics on the TargetLink Data Dictionary (TargetLink Data Dictionary Basic Concepts Guide)
- Managing Data Specifications in the Data Dictionary (TargetLink Preparation and Simulation Guide)

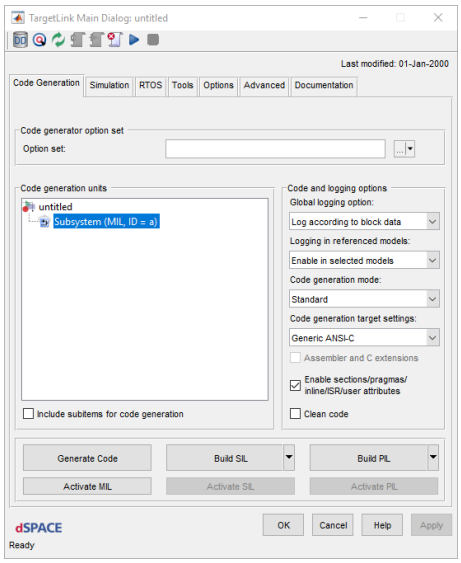
### Using TargetLink operation modes

TargetLink provides different operation modes. You can switch between the following operation modes:

- Modeling Only operation mode: not license-protected and used for designing a model.
- Full-Featured operation mode: license-protected and used for generating code.

For more information, refer to Overview of the TargetLink Operation Modes (TargetLink Blockset Guide).

### Controlling TargetLink via the TargetLink Main Dialog




You can use the Main Dialog to access model data, control the production code generation, start a simulation, and open various TargetLink utilities and tools. You can open the Main Dialog via the TargetLink menu in the Simulink Editor or via the TargetLink Main Dialog block.

### Automating TargetLink tasks via theTargetLink API

You can automate tasks and quickly access commands via the TargetLink API. For an overview of the API functions and the different use cases, refer to API Functions (TargetLink API Reference).

### Configuring the Code Generator

You can use Code Generator options to configure the Code Generator for production code generation. Refer to Basics on Configuring the Code Generator for Production Code Generation (TargetLink Customization and Optimization Guide).

**Simulating models and code**

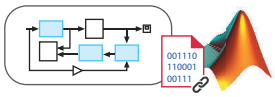
TargetLink supports three simulation modes:

- Model-in-the-loop simulation (MIL)
- Software-in-the-loop simulation (SIL)
- Processor-in-the-loop simulation (PIL)

You can use the different simulation modes to gradually improve your model and to achieve production code quality. For more information, refer to Basics on Simulation Modes and Preconditions (TargetLink Preparation and Simulation Guide).

**Model-in-the-loop (MIL) simulation mode**

Relevant symbols in the workflow illustration:

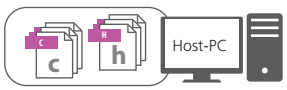



The MIL simulation mode is used for controller design and parameterization, for behavior validity checks and test purposes, to detect overflows and to serve as a reference for the production code simulations. It is the most accurate simulation mode, because the transfer behavior of the TargetLink subsystem is computed using TargetLink blocks that have the same transfer behavior as the corresponding Simulink blocks. All arithmetic calculations are done with the Simulink data type, independently of the production code data types set in the TargetLink block dialogs. Because the MIL signal represents the real physical value, it is generally recommended to continue using the double data type.

For more information, refer to How to Simulate in MIL Simulation Mode (Reference Simulation) (TargetLink Preparation and Simulation Guide).

**Software-in-the-loop (SIL) simulation mode**

Relevant symbols in the workflow illustration:

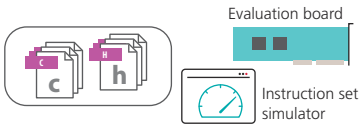



The SIL simulation mode is used to simulate the model using the generated production code and to investigate fixed-point arithmetic effects like quantization errors or saturation. The numerical effects that occur on the development PC are almost the same as on the targeted microprocessor.

For more information, refer to How to Simulate in SIL Simulation Mode (TargetLink Preparation and Simulation Guide).

**Processor-in-the-loop (PIL) simulation mode**

Relevant symbols in the workflow illustration:



The PIL simulation mode is used to verify simulation results from the MIL and SIL simulation modes by executing the code on a physical or virtual evaluation board (EVB). Thus, you can simulate under realistic operating conditions and investigate errors caused by the target compiler or even by the target processor. In addition, final verifications of the generated code can be performed with regard to execution time (physical EVB only) and stack usage during simulation.

For more information, refer to How to Simulate in PIL Simulation Mode (🕮 TargetLink Preparation and Simulation Guide).

**Generating documentation**

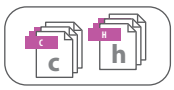
Relevant symbol in the workflow illustration:

In the final phase of developing production code with TargetLink, you might want to document the generated code and the Simulink model, including Stateflow charts, from which the production code has been generated. Refer to How to Generate the Documentation (🕮 TargetLink Interoperation and Exchange Guide).

TargetLink provides several scripts and mechanisms to customize the documentation generation. For more information, refer to Basics on Customizing the Generated Documentation (🕮 TargetLink Interoperation and Exchange Guide).

**Export and integration**

Relevant symbols in the workflow illustration:




You can export the source files generated by TargetLink and integrate them in your own ECU C modules. Refer to:

- How to Export Generated Files from the TargetLink Environment (🕮 TargetLink Interoperation and Exchange Guide)
- How to Embed TargetLink Code into Your Own C Modules (🕮 TargetLink Interoperation and Exchange Guide)

In addition, you can import and export various file formats via the TargetLink Data Dictionary. For more information, refer to Basics on Importing and Exporting Files (🕮 TargetLink Interoperation and Exchange Guide).

**Related topics**

Basics

Basics on Simulation Modes and Preconditions (🕮 TargetLink Preparation and Simulation Guide)
Overview of Standard TargetLink Use Cases ..... 26

# How to Open the TargetLink Block Library

**Objective**

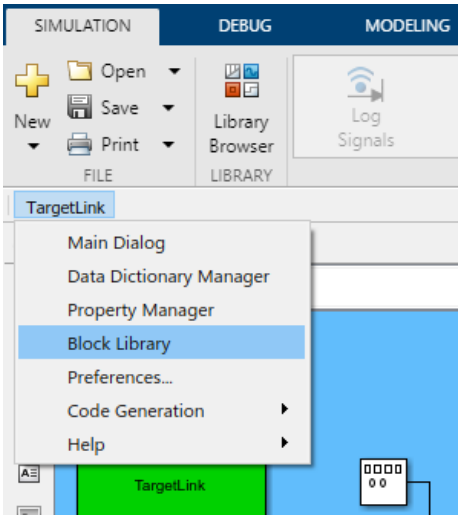
To implement your control algorithms, the TargetLink block library (tllib) provides simulation and utility blocks.

**Method**

**To open the TargetLink block library**

1 On the menu bar of the Simulink Editor, click TargetLink.

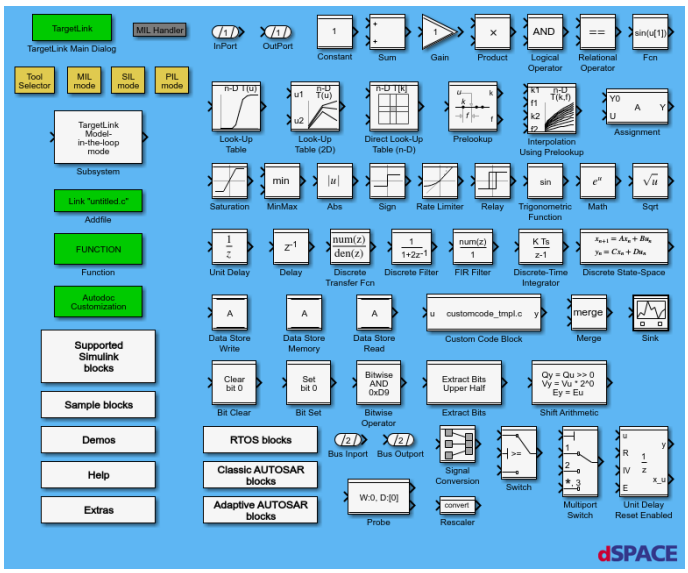
2 Select Block Library to open the TargetLink block library.




Alternatively, you can type **`tllib`** in the MATLAB Command Window.

**Result**

You opened the TargetLink block library (*tllib*).



**Related topics**

Basics

Basics on Using TargetLink ..... 9

HowTos

How to Create a Model from Scratch ..... 18

# How to Open a Demo Model

**Objective**

You can use the provided demo models to become familiar with the operating principles of TargetLink. Refer to 🕮 TargetLink Demo Models.

**Method**

**To open a demo model**

1. In the MATLAB Command Window, type `tl_demos`.
2. In the MATLAB Help Browser, select a demo model.
3. In the top right corner, click Open this example.

**Tip**

You can also open a desired demo model directly from the MATLAB Command Window. Example:

```
tl_demos poscontrol
```

**Result**

The demo model opens.

**Related topics**

Basics

Basics on Using TargetLink ..... 9

HowTos

How to Open the TargetLink Block Library ..... 16

References

tl_demos (🕮 TargetLink API Reference)

# How to Create a Model from Scratch

**Objective**

You can create a simple TargetLink model from scratch by adding at least one TargetLink **Subsystem** block and exactly one **MIL Handler** block to a model.

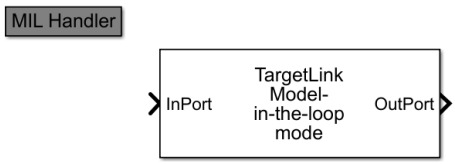
**Method**

**To create a model from scratch**

1. Open a blank Simulink model.
2. Add a TargetLink **Subsystem** block.
3. Add a TargetLink **MIL Handler** block.
4. Save the model.

**Result**

A simple TargetLink model




You created a simple TargetLink model from scratch.

**Next steps**

You can model the control algorithm in the TargetLink subsystem with TargetLink blocks from the TargetLink block library (tllib).

**Related topics**

Basics

Basics on Using TargetLink ..... 9

HowTos

How to Open the TargetLink Block Library ..... 16

# How to Make User Libraries Upgrade-Capable

**Objective**

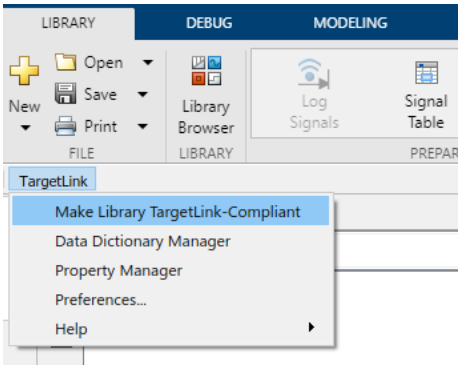
User libraries must be TargetLink-compliant to ensure a seamless upgrade to newer TargetLink versions.

**Method**

**To make user libraries upgrade-capable**

1 Open your user library.

2 In the TargetLink menu, select Make Library TargetLink-Compliant.




3 Save your user library.

**Result**

You made your user library upgrade-capable. TargetLink automatically performs an upgrade when your library is opened with a newer TargetLink version.

**Related topics**

HowTos

How to Prepare Simulink Systems for Code Generation with TargetLink (TargetLink Preparation and Simulation Guide)

References

tl_prepare_system (TargetLink API Reference)

# How to Set TargetLink to Batch Mode

**Method**

**To set TargetLink to batch mode**

1 Save the current batch mode state.

```
BatchState = ds_error_get('BatchMode');
```

2 Set TargetLink to batch mode.

```
ds_error_set('BatchMode','on');
```

**Result**

You set TargetLink to batch mode.

You can restore the saved batch mode state as follows:

```
ds_error_set('BatchMode',BatchState);
```

**Related topics**

References

ds_error_get('BatchMode') (TargetLink API Reference)
ds_error_set (TargetLink API Reference)

# Compatibility Information

**Where to go from here**

Information in this section



Information in other sections

Information on the new features of TargetLink 5.1 and TargetLink Data Dictionary 5.1.

New Features of TargetLink 5.1 (📖 New Features and Migration)

Information on the software installation process and on handling dSPACE software and dSPACE licenses.

Required MATLAB Releases (📖 Installing dSPACE Software)

## Compatibility with Other dSPACE Products

**Overview**

dSPACE recommends using only software products from the same dSPACE Release. This will provide maximum run-time compatibility. For more information, refer to Run-Time Compatibility of dSPACE Software (📖 Installing dSPACE Software).

TargetLink is currently released once a year (B Release) and is therefore compatible with all the dSPACE products of that dSPACE Release. If not stated otherwise (Software Installation and Management Guide or other product-

specific compatibility information), TargetLink is also compatible with dSPACE products released in between two TargetLink releases, i.e., with the A Release. For further, possibly updated or more detailed information, refer to: http://www.dspace.com/go/ds_sw_combi.

TargetLink 5.1 can be installed and used with the dSPACE products in the following table:

| Product | Version[1)] | Example TargetLink Use Case |
|---|---|---|
| AUTOSAR Compare | 1.0 | Comparing and merging AUTOSAR files |
| ConfigurationDesk | 6.6 | Generating a V-ECU implementation to use in ConfigurationDesk (for real-time simulation) |
| ControlDesk | 7.3 | Building V-ECUs (OSA and A2L file) for calibration and measurement |
| Model Compare | 3.1<br>3.0 | Comparing and merging of models |
| RTI | 7.15 | Using TargetLink production code in real-time applications<br>Using TargetLink models in real-time applications |
| RTI Bypass Blockset | 3.15 | Using the RTI Bypass Blockset to configure ECU interfaces and implement new ECU functions for bypassing or for tests. |
| SystemDesk | 5.5<br>5.4 | Exchanging AUTOSAR files and SWC containers |
| VEOS | 5.1 | Generating a V-ECU implementation to integrate with the VEOS Player (for offline simulation) |

1) Version numbers are valid only for the current release (Release 2020-B)

## Related topics

Basics

Basics on dSPACE AUTOSAR Compare (dSPACE AUTOSAR Compare Manual)
Interoperating with Other dSPACE Tools for Virtual Validation (TargetLink Interoperation and Exchange Guide)
Interoperating with SystemDesk via SWC Containers (TargetLink Interoperation and Exchange Guide)
Introduction to Using Classic AUTOSAR in TargetLink (TargetLink Classic AUTOSAR Modeling Guide)
Using a Software Architecture Tool Together with TargetLink (TargetLink Classic AUTOSAR Modeling Guide)

References

Generating V-ECU Implementations (TargetLink API Reference)
Required MATLAB Releases (Installing dSPACE Software)
Stand-Alone Model Manager (TargetLink Tool and Utility Reference)
TargetLink V-ECU Manager (TargetLink Tool and Utility Reference)
tl_tl2rti (TargetLink API Reference)

# AUTOSAR Releases Supported by TargetLink

**Supported Classic AUTOSAR releases**

| Classic AUTOSAR Release | Revision |
|---|---|
| R19-11 | 19-11 |
| 4.4 | 4.4.0 |
| 4.3 | 4.3.1<br>4.3.0 |
| 4.2 | 4.2.2<br>4.2.1 |
| 4.1 | 4.1.3<br>4.1.2<br>4.1.1 |
| 4.0 | 4.0.3<br>4.0.2 |

# Combinations of Evaluation Boards, Microcontrollers, and Compilers

**Target Simulation Module**

To perform a PIL simulation on a specific evaluation board, TargetLink requires a suitable compiler. The following table shows the combinations of evaluation boards and compilers included in the Target Simulation Module (TSM). The TSM is licensed separately from the TargetLink Base Suite.

| Microcontroller Family | Microcontroller Unit | Evaluation Board | Compiler[1)] | Patch[2)] |
|---|---|---|---|---|
| ARM Cortex-M3 | STMicroelectronics STM32F107 | Emerge-Engineering ARM MEDKit | Keil 5.2 | .0 |
| ARM Cortex-M3 | ARM Cortex M3 | Lauterbach Simulator for ARM CortexM3 | Keil 5.2 | .0 |
| Freescale MPC5700VLE | Freescale MPC5748G | Freescale MPC5748GEVB | Green Hills 2019 | .1.4 |
| Freescale MPC5700VLE | Freescale MPC5748G | Freescale MPC5748GEVB | Wind River Diab 5.9 | .0 |
| Freescale S12X | Freescale MC9S12XEP100 | Freescale EVB9S12XEP100 | Cosmic 4.8 | .11 |
| Freescale S12X | Freescale MC9S12XEP100 | Freescale EVB9S12XEP100 | Metrowerks CodeWarrior 5.1 | 5.0.41 build 10203 |
| Infineon c166 | Infineon c167 | I+ME Promotion Package 166 | Altium TASKING C166/ST10 Toolset 8.6 | r1 p3 |
| Infineon TriCore 1-1.3 | Infineon TC1766 | Infineon TriBoard TriCore 1766 | Altium TASKING TriCore VX-Toolset 3.2 | r1 |
| Infineon TriCore 1-1.3 | Infineon TC1766 | Infineon TriBoard TriCore 1766 20 MHz | Altium TASKING TriCore VX-Toolset 3.2 | r1 |

| Microcontroller Family | Microcontroller Unit | Evaluation Board | Compiler[1)] | Patch[2)] |
|---|---|---|---|---|
| Infineon TriCore 1-1.3 | Infineon TC1767 | Infineon TriBoard TriCore 1767 | Altium TASKING TriCore VX-Toolset 3.2 | r1 |
| Infineon TriCore 1-1.3 | Infineon TC1796 | Infineon TriBoard TriCore 1796 | Altium TASKING TriCore VX-Toolset 3.2 | r1 |
| Infineon TriCore 1-1.6 | Infineon TC275 | Infineon TriBoard TriCore 275 | Altium TASKING TriCore VX-Toolset 6.3 | r1 |
| Infineon TriCore 1-1.6 | Infineon TC275 | Infineon TriBoard TriCore 275 | HighTec GNU 4.9 | .2 |
| Infineon TriCore 1-1.6 | Infineon TC275 | Lauterbach Simulator for TriCore 275 | Altium TASKING TriCore VX-Toolset 4.2 | r2 |
| Infineon XC2000 | Infineon XC2287 | Infineon EasyKit XC2287 | Altium TASKING C166/ST10 VX-Toolset 3.0 | r3 |
| Renesas RH850 | Renesas RH850/F1L_R7F7010354 | Renesas YRH850F1L_R7F7010354 | Green Hills 2019 | .1.5 |
| Renesas SH-2 | Renesas SH-2E/SH7058 | Renesas EVB7058 | Renesas 9.3 | .0 |
| Renesas SH-2 | Renesas SH-2A-FPU/SH72513 | Renesas SH72513 System Development Kit | Renesas 9.4 | .0 |
| Renesas V850E2 | Renesas V850E2/Fx4-µPD70F4012 | Renesas AB_050_Fx4_70F4012 | Green Hills 2019 | .1.5 |
| Texas Instruments TMS570 | Texas Instruments TMS570LC43 | Texas Instruments LAUNCHXL2570LC43 | Texas Instruments Code Composer Studio 7.0 | .3.0 |

1) Compiler Suite Version Supported
2) Tested with Compiler Version

### Target Simulation Module Extensions

For a full list of all supported evaluation board and compiler combinations, refer to www.dspace.com/go/tlpil. The combinations in the list are free of charge if you have a valid Software Maintenance Service (SMS) contract.

If you do not have a valid SMS contract or are looking for a different combination, contact dSPACE Support or your sales representative.

All Target Simulation Module Extensions will be delivered as TSM Extension Packages and can be installed via the TargetLink Preferences Editor. For more information on the installation, refer to How To Install TSM Extension Packages (TargetLink Customization and Optimization Guide).

Target Simulation Module Extensions require the Target Simulation Module.

# Overviews of Use Cases, Features and User Documentation

**Introduction**

To help you navigate from the use cases and their supported features to the appropriate user documentation.

**Where to go from here**

Information in this section

# Overview of Use Cases

**Introduction**

To give you an overview of the essential TargetLink use cases and the corresponding TargetLink features. The use cases are categorized according to the main TargetLink code generation scenarios and representative TargetLink features.

**Where to go from here**

Information in this section



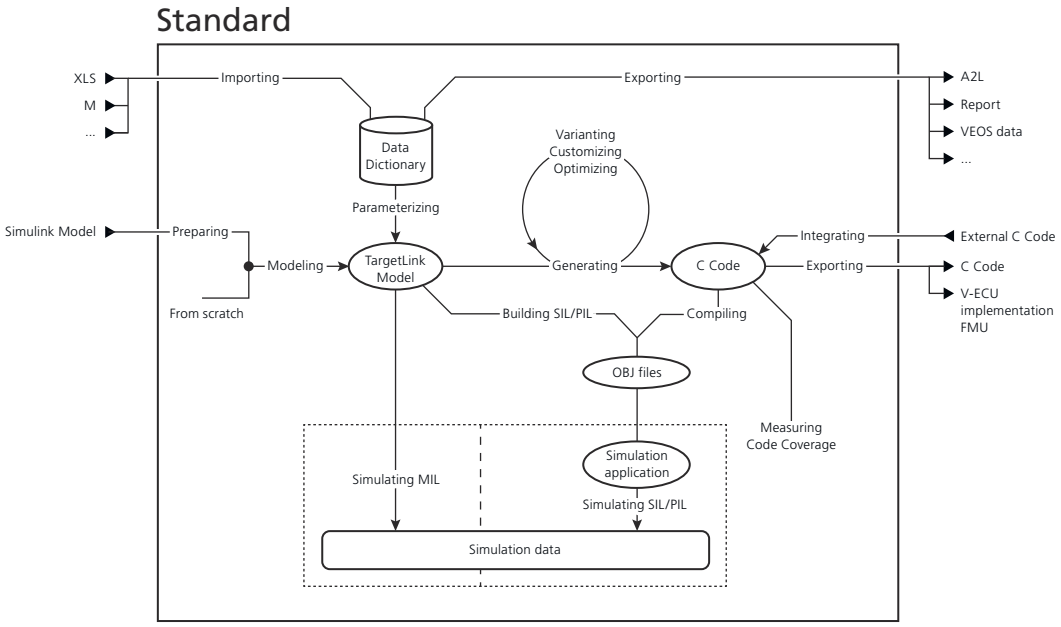
## Overview of Standard TargetLink Use Cases

**Introduction**

This category typically makes use of all the features that come with the **TargetLink Base Suite** software module, which includes the standard ANSI C Code Generator.

**Use cases overview**

The map visualizes the main use cases, the interdependencies and interactions between them, their resulting work products and their input and output interfaces.




**Use case details**

The table lists the use cases shown in the map above and for each use case a link that guides you to detailed information on this use case (features, user documentation) in this guide. The use cases are alphabetically sorted.

| Use Case | More Information ... [1)] |
|---|---|
| Building SIL/PIL | Building SIL/PIL Applications on page 36 |
| Compiling | Compiling Code on page 37 |
| Customizing | Customizing Code on page 38 |
| Exporting | Exporting Code on page 40 |
| Exporting | Exporting Data from the Data Dictionary on page 41 |
| Generating | Generating Code on page 42 |
| Importing | Importing Data to the Data Dictionary on page 46 |
| Integrating | Integrating External Code on page 48 |
| Measuring Code Coverage | Code Coverage Measurement on page 36 |
| Modeling | Modeling with TargetLink on page 48 |
| Optimizing | Optimizing C Code on page 51 |
| Parameterizing | Parameterizing a Model on page 53 |
| Preparing | Preparing a Simulink Model for TargetLink on page 55 |

| Use Case | More Information ... [1)] |
|---|---|
| Simulating | Simulating MIL/SIL/PIL on page 56 |
| Varianting | Varianting C Code on page 57 |

1) Refer to TargetLink Features Sorted by Use Case on page 35

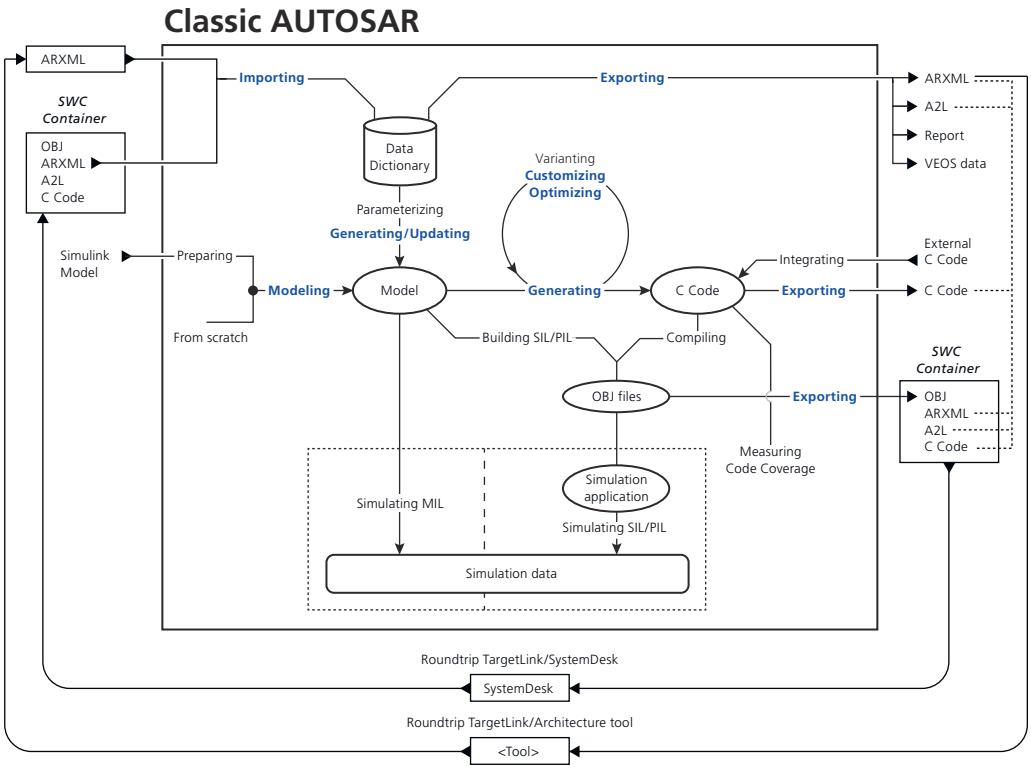
# Overview of Classic AUTOSAR-Specific Use Cases

**Introduction**

This category makes use of all the features that come with the TargetLink AUTOSAR Module which you can use for generating AUTOSAR software components (SWCs) containing function code for ECUs and for modeling and simulating them.

**Use cases overview**

The map visualizes the main use cases, the interdependencies and interactions between them, their resulting work products and their input and output interfaces.




ARXML, A2L, and C Code files can be exported either as part of an SWC container or alone. OBJ files can be exported only via SWC container. Round trips can be made by using SWC containers or ARXML files.

Use cases are marked blue in the illustration if they not only relate to standard TargetLink features but also to exclusive AUTOSAR features.

**Use case details**

The table lists the use cases shown in the map above and for each use case a link that guides you to detailed information on this use case (features, user documentation) in dSPACE Help. The use cases are alphabetically sorted.

| Use Case | More Information ... [1] |
|---|---|
| Building SIL/PIL | Building SIL/PIL Applications on page 36 |
| Compiling | Compiling Code on page 37 |
| Customizing | Customizing Code on page 38 |
| Exporting | Exporting Code on page 40 |
| Exporting | Exporting Data from the Data Dictionary on page 41 |
| Generating | Generating Code on page 42 |
| Generating/Updating[2] | Generating/Updating Classic AUTOSAR Frame Models on page 44 |
| Importing | Importing Data to the Data Dictionary on page 46 |
| Integrating | Integrating External Code on page 48 |
| Measuring Code Coverage | Code Coverage Measurement on page 36 |
| Modeling | Modeling with TargetLink on page 48 |
| Optimizing | Optimizing C Code on page 51 |
| Parameterizing | Parameterizing a Model on page 53 |
| Preparing | Preparing a Simulink Model for TargetLink on page 55 |
| Simulating | Simulating MIL/SIL/PIL on page 56 |

[1] Refer to TargetLink Features Sorted by Use Case on page 35
[2] Exclusive AUTOSAR use case

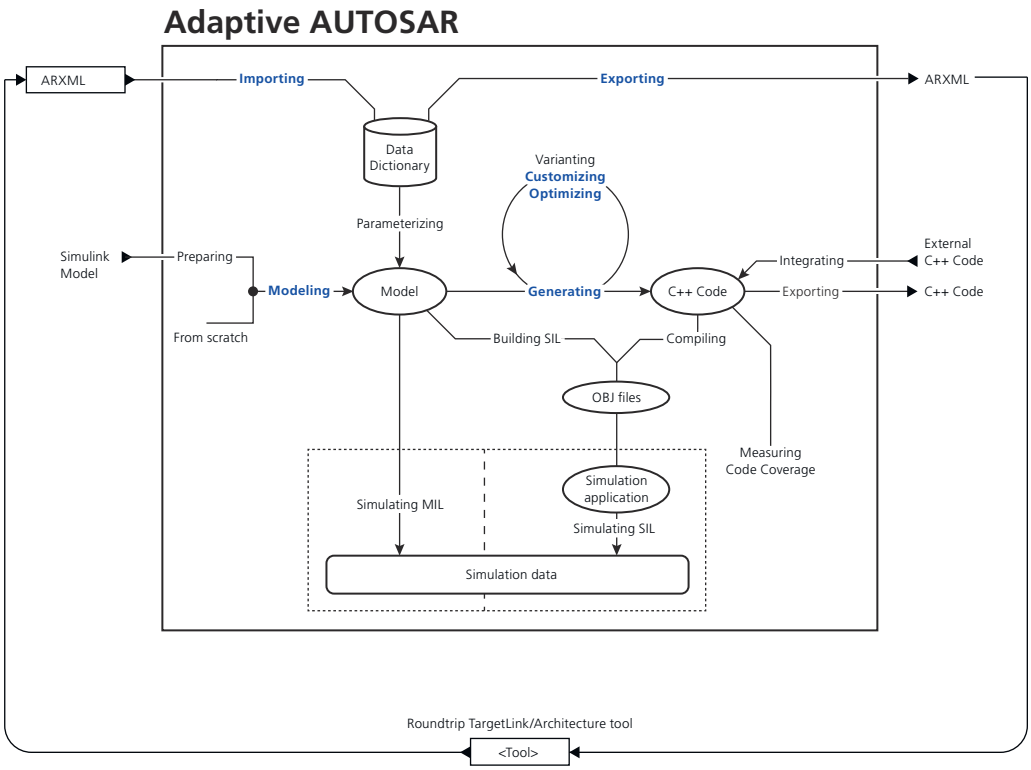
## Overview of Adaptive AUTOSAR-specific Use Cases

**Introduction**

This category makes use of all the features that come with the **TargetLink Adaptive AUTOSAR Module** which you can use for modeling and generating code for parts of Adaptive Applications.

**Use cases overview**

The map visualizes the main use cases, the interdependencies and interactions between them, their resulting work products and their input and output interfaces.




ARXML and C++ Code files can be exported. Round trips can be made by using ARXML files.

Use cases are marked blue in the illustration if they not only relate to standard TargetLink features but also to exclusive AUTOSAR features.

**Use case details**

The table lists the use cases shown in the map above and for each use case a link that guides you to detailed information on this use case (features, user documentation) in dSPACE Help. The use cases are alphabetically sorted.

| Use Case | More Information ... [1)] |
|---|---|
| Building SIL | Building SIL/PIL Applications on page 36 |
| Compiling | Compiling Code on page 37 |
| Customizing | Customizing Code on page 38 |
| Exporting | Exporting Code on page 40 |
| Exporting | Exporting Data from the Data Dictionary on page 41 |
| Generating | Generating Code on page 42 |

| Use Case | More Information ... [1)] |
|---|---|
| Importing | Importing Data to the Data Dictionary on page 46 |
| Integrating | Integrating External Code on page 48 |
| Modeling | Modeling with TargetLink on page 48 |
| Optimizing | Optimizing C Code on page 51 |
| Parameterizing | Parameterizing a Model on page 53 |
| Preparing | Preparing a Simulink Model for TargetLink on page 55 |
| Simulating | Simulating MIL/SIL/PIL on page 56 |

[1)] Refer to TargetLink Features Sorted by Use Case on page 35

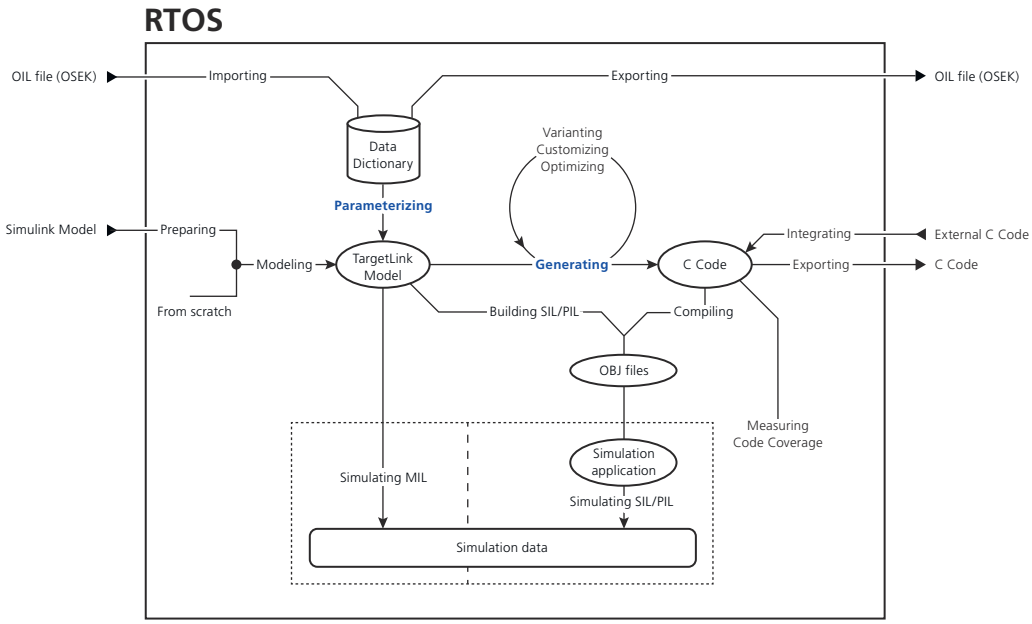
## Overview of RTOS-Specific Use Cases

**Introduction**

This category makes use of features that come with the TargetLink Base Suite software module. Features that relate to code generation for the OSEK real-time operating system require the TargetLink TMOS-OSEK Module.

**Use cases overview**

The map visualizes the main use cases, the interdependencies and interactions between them, their resulting work products and their input and output interfaces.



Use cases are marked blue in the illustration if they not only relate to standard TargetLink features but also to exclusive RTOS features.

**Use case details**

The table lists the use cases shown in the map above and for each use case a link that guides you to detailed information on this use case (features, user documentation) in this guide. The use cases are alphabetically sorted.

| Use Case | More Information ... [1] |
|---|---|
| Building SIL/PIL | Building SIL/PIL Applications on page 36 |
| Compiling | Compiling Code on page 37 |
| Customizing | Customizing Code on page 38 |
| Exporting | Exporting Code on page 40 |
| Exporting | Exporting Data from the Data Dictionary on page 41 |
| Generating | Generating Code on page 42 |
| Importing | Importing Data to the Data Dictionary on page 46 |
| Integrating | Integrating External Code on page 48 |
| Measuring Code Coverage | Code Coverage Measurement on page 36 |
| Modeling | Modeling with TargetLink on page 48 |
| Optimizing | Optimizing C Code on page 51 |
| Parameterizing | Parameterizing a Model on page 53 |
| Preparing | Preparing a Simulink Model for TargetLink on page 55 |
| Simulating | Simulating MIL/SIL/PIL on page 56 |
| Varianting | Varianting C Code on page 57 |

[1] Refer to TargetLink Features Sorted by Use Case on page 35

# Overview of RCP Use Cases

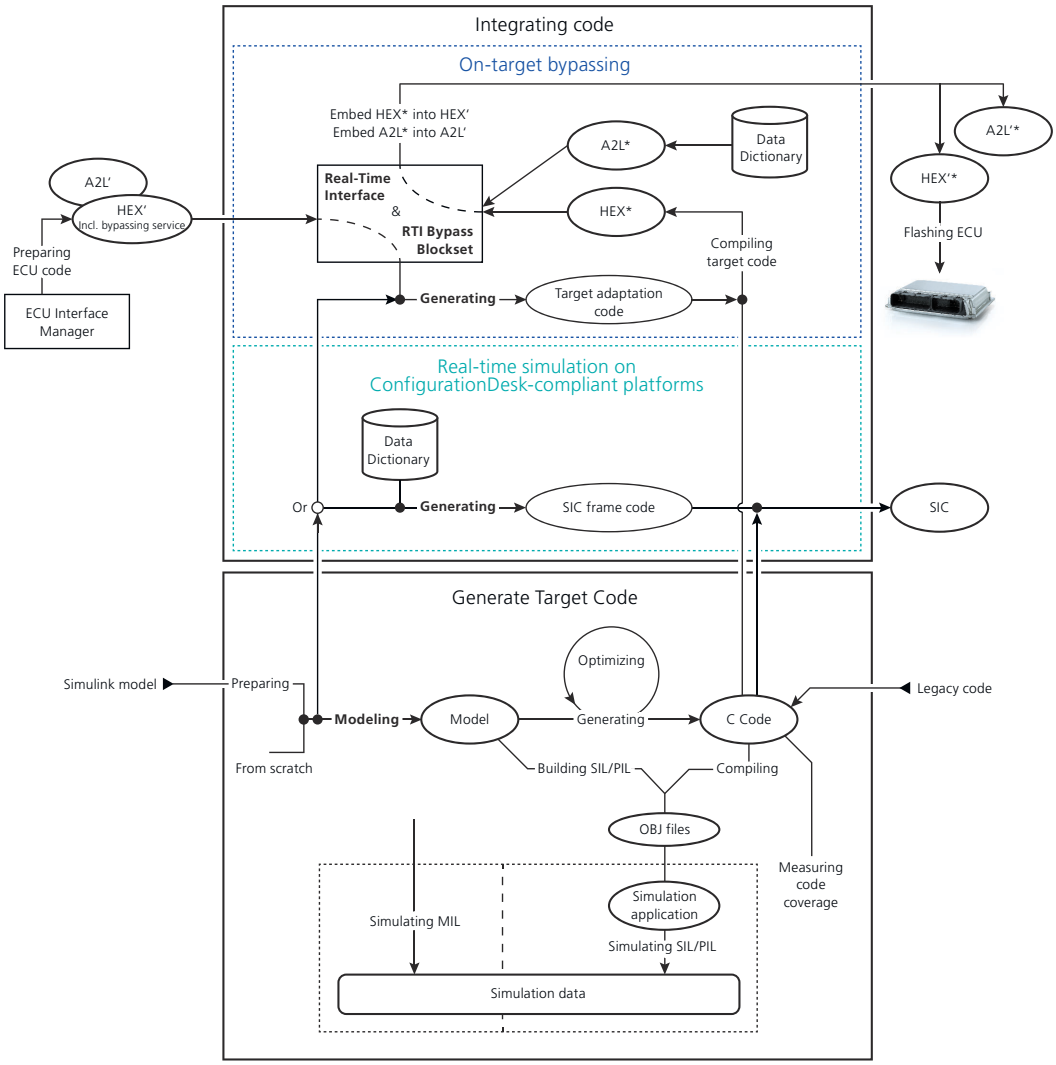
**Introduction**

This category typically uses all the features that come with the **TargetLink Base Suite** software module. For the *On-Target Bypassing* use case, the **RTI Bypass Blockset** is required, because this use case addresses on-target prototyping with dSPACE's RTI Bypass blocks, including full code generation for on-target bypassing.

**Use cases overview**

The map visualizes the main use cases, the interdependencies and interactions between them, their resulting work products and their input and output

interfaces. The two RCP use cases *On-Target Bypassing* and *Real-time simulation on ConfigurationDesk-compliant platforms* are shown as separate boxes within the *Integrating Code* box.




## Use case details

The table lists the use cases shown in the map above. For each use case, it provides a link that guides you to detailed information on this use case (features, user documentation) in this guide. The use cases are sorted alphabetically.

| Use Case | More Information ... [1)] |
|---|---|
| Building SIL/PIL | Building SIL/PIL Applications on page 36 |
| Compiling | Compiling Code on page 37 |
| Customizing | Customizing Code on page 38 |
| Exporting | Exporting Code on page 40 |

| Use Case | More Information ... [1)] |
|---|---|
| Exporting | Exporting Data from the Data Dictionary on page 41 |
| Generating | Generating Code on page 42 |
| Generating | Generating Target Adaption Code for ECU Bypassing on page 44 |
| Generating | Generating an SIC file for ConfigurationDesk on page 46 |
| Importing | Importing Data to the Data Dictionary on page 46 |
| Integrating | Integrating External Code on page 48 |
| Measuring Code Coverage | Code Coverage Measurement on page 36 |
| Modeling | Modeling with TargetLink on page 48 |
| Modeling | Modeling with the RTI Bypass Blockset in TargetLink on page 51 |
| Optimizing | Optimizing C Code on page 51 |
| Parameterizing | Parameterizing a Model on page 53 |
| Preparing | Preparing a Simulink Model for TargetLink on page 55 |
| Simulating | Simulating MIL/SIL/PIL on page 56 |
| Varianting | Varianting C Code on page 57 |

[1)] Refer to TargetLink Features Sorted by Use Case on page 35

# TargetLink Features Sorted by Use Case

**Introduction**

To give you an overview of the features that relate to a certain use case including links to the user documentation.

**Where to go from here**

Information in this section

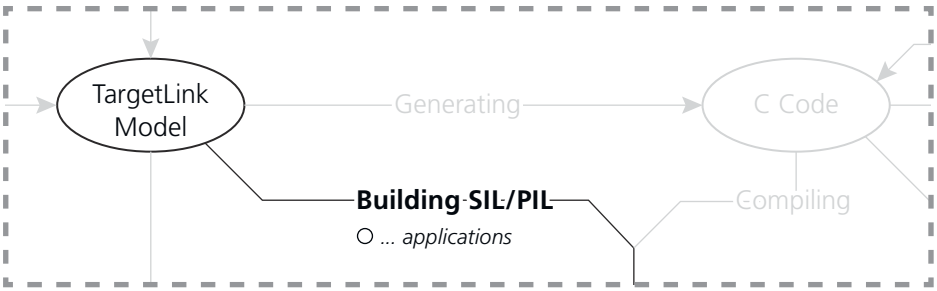
# Building SIL/PIL Applications

**Introduction**

To validate the generated production code and to profile execution time and resource consumption by running SIL and PIL simulations, you must build SIL and PIL simulation applications first. TargetLink provides a build process for this.

**Sub use cases**

The map lists the sub use cases for *Building SIL/PIL Applications*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Building ... | Feature | User Documentation |
|---|---|---|
| SIL/PIL applications[1)] | Build process:<br>1. Code generation<br>2. Code compilation<br>3. Application download | Building Simulation Applications in a Single-Step Build Process[2)]<br>Generating and Compiling Code[3)] |

1) PIL not available for Adaptive AUTOSAR.
2) Refer to TargetLink Preparation and Simulation Guide.
3) Refer to TargetLink API Reference.

**Expert features**

No such features.

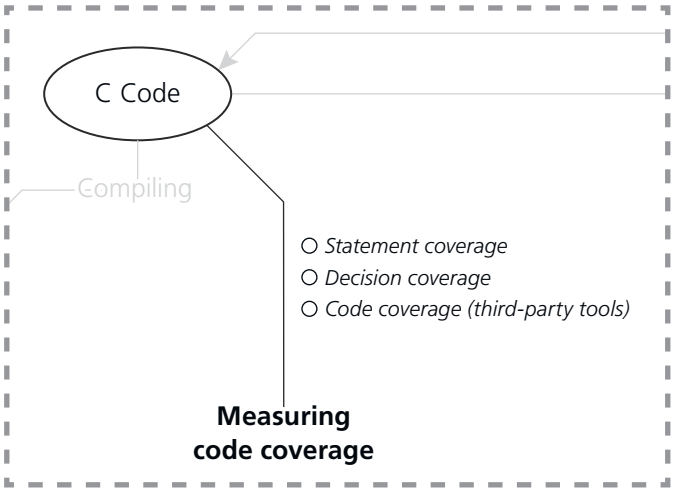
# Code Coverage Measurement

**Introduction**

Code coverage measurement allows you to find parts of your production code which are not executed and thus have not been tested adequately. When you have this information, you can increase code coverage by varying the model's stimulus signals.

**Sub use cases**

The map lists the sub use cases for *Code Coverage Measurement*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Measuring ... | Feature | User Documentation |
|---|---|---|
| ▪ Statement coverage<br>▪ Decision coverage | TargetLink's code coverage measurement | Measuring the Code Coverage of Generated Code[1)] |
| Code coverage (e.g., multiple condition/decision coverage) | Third-party tool integration | How to Measure Code Coverage via Third-Party Tools (Testwell CTC)[1)] |

[1)] Refer to TargetLink Preparation and Simulation Guide.

**Expert features**

No such features.

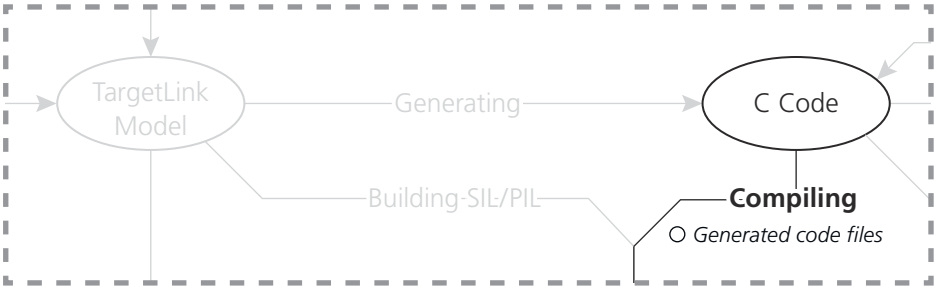
# Compiling Code

**Introduction**

The code compilation process compiles and links the files generated during the code generation process and creates the simulation application as output. Which type of simulation application is created (SIL or PIL) depends on the compile process you start.

**Sub use cases**

The map lists the sub use cases for *Compiling Code*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Compiling ... | Feature | User Documentation |
|---|---|---|
| Generated code files | ▪ SIL compilation<br>▪ PIL compilation[1)] | Compiling Production Code[2)]<br>Generating and Compiling Code[3)] |

1) Not available for Adaptive AUTOSAR.
2) Refer to TargetLink Preparation and Simulation Guide.
3) Refer to TargetLink API Reference.

**Expert features**

No such features.

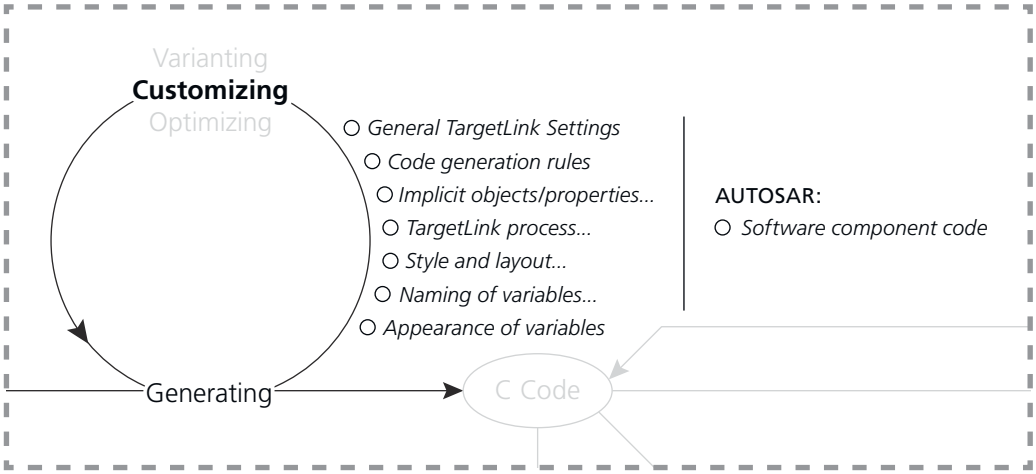
## Customizing Code

**Introduction**

To adapt the code to your specific needs, for example, by influencing the code generation process, configuring the Code Generator, or using templates.

**Sub use cases**

The map lists the sub use cases for *Customizing Code*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Customizing ... | Feature | User Documentation |
|---|---|---|
| General TargetLink Settings | Preferences Editor | Customizing the TargetLink Environment[1)]<br>Adapting Code to Company Coding Styles[1)] |
| Code generation rules | Code Generator options | Configuring the Code Generator for Production Code Generation[1)]<br>Effects of Code Generator Option Sets[2)] |
| Implicit objects/properties used by Code Generator | Templates | Controlling Objects and Properties Created by the Code Generator (Templates)[1)] |
| TargetLink processes, e.g., code generation | Hook scripts | Customizing Production Code Generation via Hook Scripts[1)] |
| Style and layout of generated code | Style Definition File | Customizing TargetLink's Code Formatting[1)]<br>Adapting Code to Company Coding Styles[1)] |
| Naming of variables etc. used in the generated code | Name macros | Generating Unique Names via Name Macros[1)] |

| Customizing ... | Feature | User Documentation |
|---|---|---|
| Appearance of variables in generated code | Variable classes | Customizing the Appearance of Variables via Variable Classes[1)] |
| | | Examples of Working with Variable Classes[1)] |
| **Classic AUTOSAR** | | |
| Software component code | AUTOSAR compiler abstraction | Basics on Compiler Abstraction According to Classic AUTOSAR[3)] |

1) Refer to TargetLink Customization and Optimization Guide.
2) Refer to TargetLink Model Element Reference
3) Refer to TargetLink Classic AUTOSAR Modeling Guide.

**Expert features**

Refer to Customizing Code at Expert Level on page 59 in this guide.

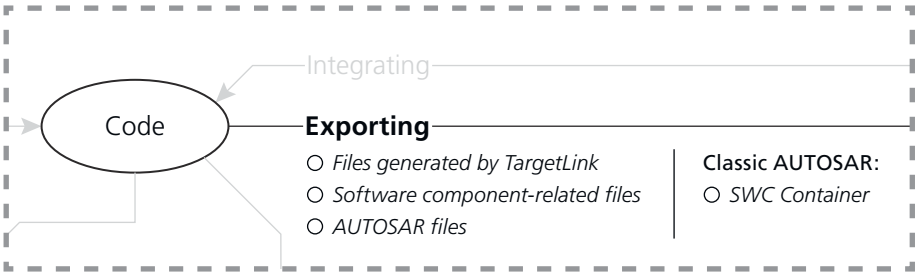
## Exporting Code

**Introduction**

You can export generated source files from the TargetLink environment and integrate them in your own ECU modules.

**Sub use cases**

The map lists the sub use cases for *Exporting Code*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Exporting ... | Feature | User Documentation |
|---|---|---|
| Files generated by TargetLink | TargetLink file export utility | How to Export Generated Files from the TargetLink Environment[1)]<br>tl_export_files[2)] |

| Exporting ... | Feature | User Documentation |
|---|---|---|
| Software component-related files | Data Dictionary Manager (container export) | Exchanging Software Component Containers[1)] AUTOSAR_FUELSYS[3)] |
| AUTOSAR files | Data Dictionary Manager (file export) | Exporting AUTOSAR Files[1)] |
| **Classic AUTOSAR** | | |
| SWC Container | Component Container (TargetLink/SystemDesk round trip) | Interoperating with SystemDesk via SWC Containers[1)] |

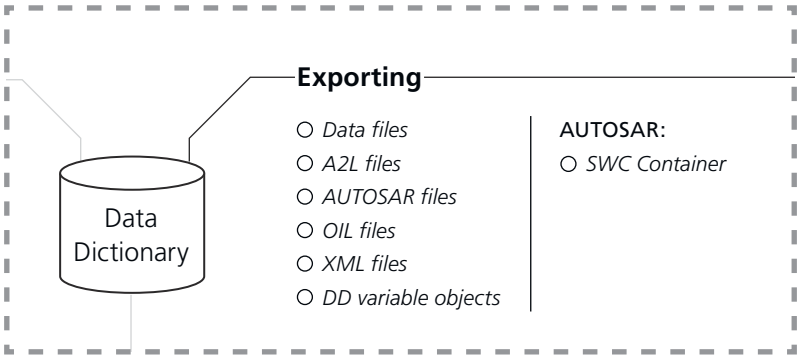
1) Refer to TargetLink Interoperation and Exchange Guide.
2) Refer to TargetLink API Reference.
3) Refer to TargetLink Demo Models.

**Expert features** No such features.

## Exporting Data from the Data Dictionary

**Introduction** To export data files in various file formats. In addition, variables of Simulink or Stateflow models are often kept in separate MATLAB files. To make these variables available in the DD, you can import them. You can work with these variables in the DD and later export them back to MATLAB.

**Sub use cases** The map lists the sub use cases for *Exporting Data from the Data Dictionary*.



**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Exporting ... | Feature | User Documentation |
|---|---|---|
| Data files | Data Dictionary Manager (file export) | How to Export Data Files from DD Workspaces[1)] |
| A2L files | Data Dictionary Manager (file export) | Exchanging A2L Files[2)] |
| AUTOSAR files | Data Dictionary Manager (file export) | Exporting AUTOSAR Files[2)]<br>Export as Container[3)] |
| OIL files | Data Dictionary Manager (file export) | Exporting OIL Files[2)] |
| XML files | Data Dictionary Manager (file export) | Exchanging XML Files[2)] |
| DD Variable objects | Data Dictionary Manager (object export) | How to Export Variable Objects via the Data Dictionary Manager[2)] |
| | MATLAB Export API | How to Export Variable Objects via the MATLAB Import Export API[2)] |
| SWC container (Classic AUTOSAR) | Component Container (SystemDesk/TargetLink round trip) | Interoperating with SystemDesk via SWC Containers[2)]<br>Using a Software Architecture Tool Together with TargetLink[4)] |
| | ARXML file export | Exporting AUTOSAR Files[2)] |
| | A2L file export | Exchanging A2L Files[2)] |

1) Refer to TargetLink Data Dictionary Basic Concepts Guide.
2) Refer to TargetLink Interoperation and Exchange Guide.
3) Refer to TargetLink Data Dictionary Manager Reference
4) Refer to TargetLink Classic AUTOSAR Modeling Guide.

**Expert features**

No such features.

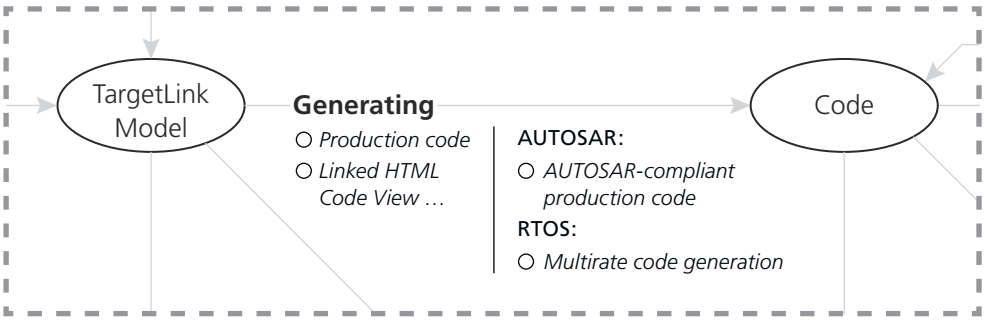
# Generating Code

**Introduction**

TargetLink provides different methods to start the code generation process. This facilitates your daily work by enabling you to start code generation directly from the level you are currently working on.

**Sub use cases**

The map lists the sub use cases for *Generating Code*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Generating ... | Feature | User Documentation |
|---|---|---|
| Production code | TargetLink Production Code Generator | Generating Production Code[1)] |
| Linked HTML code view for code review | Model-linked code view | Tracing Objects between Model and Code (Model-Linked Code View)[1)] |
| Production code compliant to C99/C11 | TargetLink Production Code Generator | Configuring the Code Generator for Production Code Generation[2)] |
| **Classic AUTOSAR** | | |
| AUTOSAR-compliant production code | Classic AUTOSAR code generation mode | Generating Classic AUTOSAR-Compliant Code[3)] |
| **Adaptive AUTOSAR** | | |
| Adaptive AUTOSAR-compliant code | Adaptive AUTOSAR code generation mode | Basics on Code Artifacts generated by TargetLink[4)] |
| **RTOS** | | |
| Multirate code generation | RTOS code generation mode | Generic Multirate Code Generation[1)]<br>Building a Multitasking Application[5)] |

1) Refer to TargetLink Preparation and Simulation Guide.
2) Refer to TargetLink Customization and Optimization Guide.
3) Refer to TargetLink Classic AUTOSAR Modeling Guide.
4) Refer to TargetLink Adaptive AUTOSAR Modeling Guide.
5) Refer to TargetLink Multirate Modeling Guide.

**Expert features**

No such features.

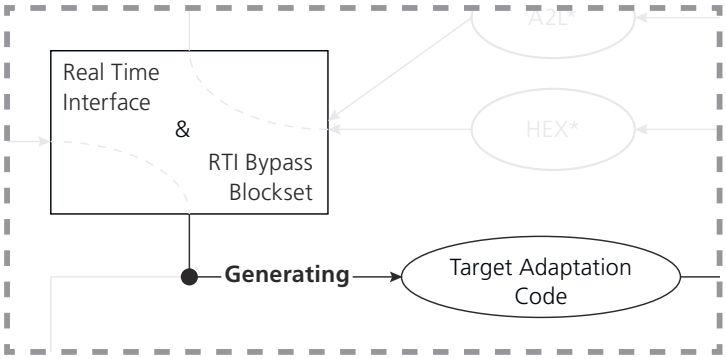
# Generating Target Adaption Code for ECU Bypassing

**Introduction**

To generate Target Adaption Code from modeled parts containing blocks of the RTI Bypass Blockset.

**Sub use cases**

The map lists the sub use cases for *Generating Target Adaption Code for ECU Bypassing*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Generating ... | Feature | User Documentation |
|---|---|---|
| Target Adaption Code | Target Adaption Code for ECU Bypassing | Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (📖 TargetLink Interoperation and Exchange Guide) |

**Expert features**

No such features.

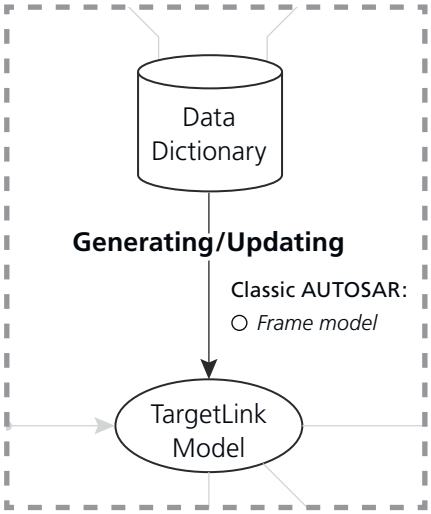
# Generating/Updating Classic AUTOSAR Frame Models

**Introduction**

By using an ARXML file as the data input you can either create an AUTOSAR frame model from scratch or update an existing one.

**Sub use cases**

The map lists the sub use cases for *Generating/Updating Classic AUTOSAR Frame Models*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Generating/Updating ... | Feature | User Documentation |
|---|---|---|
| **Classic AUTOSAR** | | |
| Frame model | Frame model generation/update via ARXML file | Generating/Updating a Frame Model from Classic AUTOSAR Data [1)] |

[1)] Refer to TargetLink Classic AUTOSAR Modeling Guide.

**Expert features**

No such features.

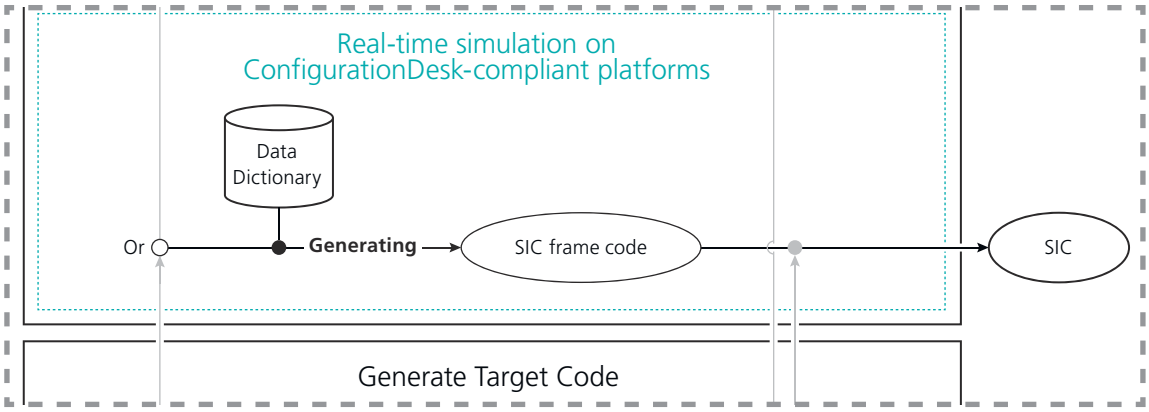
# Generating an SIC file for ConfigurationDesk

**Introduction**

To simulate the production code generated by TargetLink on real-time hardware supported by ConfigurationDesk.

**Sub use cases**

The map lists the sub use cases for *Generating an SIC file for ConfigurationDesk*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Generating ... | Feature | User Documentation |
|---|---|---|
| Simulink implementation container (SIC) | RCP on CPU | Interoperating with ConfigurationDesk (TargetLink Interoperation and Exchange Guide) |

**Expert features**

No such features.

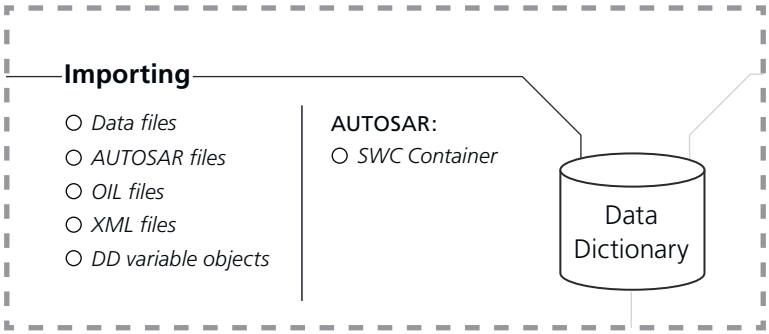
# Importing Data to the Data Dictionary

**Introduction**

To import data files in various file formats. In addition, variables of Simulink or Stateflow models are often kept in separate MATLAB files. To make these variables available in the Data Dictionary, you can import them. You can work with these variables in the Data Dictionary and later export them back to MATLAB.

**Sub use cases**

The map lists the sub use cases for *Importing Data to the Data Dictionary*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Importing ... | Feature | User Documentation |
|---|---|---|
| Data files | Data Dictionary Manager (file export) | How to Import Data Files to DD Workspaces[1)] |
| AUTOSAR files | Data Dictionary Manager (file export) | Import from AUTOSAR File[2)]<br>Import from Container[2)] |
| OIL files | Data Dictionary Manager (file export) | Importing OIL Files[3)] |
| XML files | Data Dictionary Manager (file export) | Importing XML Files[3)] |
| DD Variable objects | Data Dictionary Manager (object export) | How to Import Variable Objects via the Data Dictionary Manager[3)] |
| | MATLAB Export API | How to Import Variable Objects via the MATLAB Import Export API[3)] |
| SWC container (Classic AUTOSAR) | Component Container (SystemDesk/TargetLink round trip) | Interoperating with SystemDesk via SWC Containers[3)]<br>Using a Software Architecture Tool Together with TargetLink[4)] |
| | ARXML file import | Importing AUTOSAR Files[3)] |

1) Refer to TargetLink Data Dictionary Basic Concepts Guide.
2) Refer to TargetLink Data Dictionary Manager Reference.
3) Refer to TargetLink Interoperation and Exchange Guide.
4) Refer to TargetLink Classic AUTOSAR Modeling Guide.

**Expert features**

No such features.

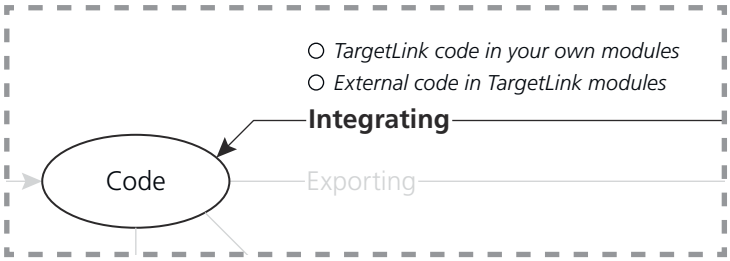
# Integrating External Code

**Introduction**

You can integrate generated source files with your own ECU C modules. In addition, TargetLink offers various methods for integrating external code into the model and the generated code.

**Sub use cases**

The map lists the sub use cases for *Integrating External Code*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Integrating ... | Feature | User Documentation |
|---|---|---|
| TargetLink code in your own modules | ▪ TargetLink file export utility<br>▪ TargetLink code inclusion | Integrating TargetLink Code in External Applications[1)] |
| External code in TargetLink modules | ▪ Custom Code block<br>▪ Function block<br>▪ Addfile block<br>▪ Blackbox mask | Overview of Methods for Embedding External Code[2)] |

1) Refer to TargetLink Interoperation and Exchange Guide.
2) Refer to TargetLink Customization and Optimization Guide.

**Expert features**

No such features.

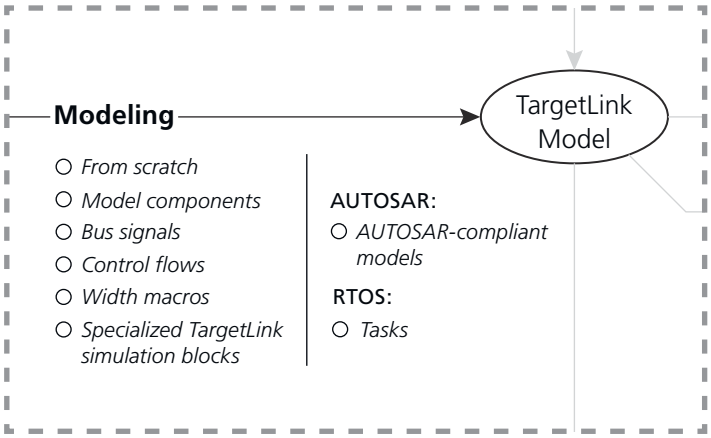
# Modeling with TargetLink

**Introduction**

You can develop your own models in TargetLink or take over existing Simulink models. In addition, TargetLink provides various capabilities for developing and partitioning large software projects.

**Sub use cases**

The map lists the sub use cases for *Modeling with TargetLink*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Modeling ... | Feature | User Documentation |
|---|---|---|
| From scratch | TargetLink block library (tllib) | How to Create a Model from Scratch on page 18<br>http://www.dspace.com/go/tl_ModelingGuidelines |
| | MathWorks® Stateflow toolbox | Using Stateflow in TargetLink[1)] |
| | TargetLink operation modes | tlOperationMode[2)] |
| Model components | Model partitioning and referencing | Decomposing Models for Distributed Development[3)]<br>Mapping of Subsystems and Functions[3)] |
| Bus signals | Bus signal support | Customizing the Representation of Buses in Production Code[3)] |
| Control flows | Stateflow support | Working with Stateflow[1)] |
| Width macros | Variable vector widths | Specifying Variable Vector Widths via Width Macros[3)] |
| Specialized TargetLink simulation blocks | ▪ Lookup tables<br>▪ Discrete Filters<br>▪ Discrete Integrators<br>▪ etc. | Working with Specialized TargetLink Blocks[1)] |
| Matrix signals | Matrix support | Working With Matrix Signals[1)] |

| Modeling ... | Feature | User Documentation |
|---|---|---|
| **Classic AUTOSAR** | | |
| AUTOSAR-compliant models by:<br>▪ Developing top-down (Input: `*.arxml`)<br>▪ Reusing non-AUTOSAR models (Input: `*.arxml`, `*.mdl`)<br>▪ Migrating non-AUTOSAR models (Input: `*.mdl`)<br>▪ Developing from scratch | TargetLink AUTOSAR Module | Introduction to Using Classic AUTOSAR in TargetLink[4]<br>Development Approaches with the TargetLink Classic AUTOSAR Module[4] |
| | TargetLink AUTOSAR Migration Tool | http://www.dspace.com/go/tl_ar_migration |
| | TargetLink AUTOSAR blocks | Refer to TargetLink Model Element Reference |
| **Adaptive AUTOSAR** | | |
| Adaptive AUTOSAR-compliant models by:<br>▪ Developing top-down (Input: `*.arxml`)<br>▪ Reusing non-AUTOSAR models (Input: `*.arxml`, `*.mdl`) | TargetLink Adaptive AUTOSAR Module | Overview of Supported Modeling Styles for Adaptive AUTOSAR[5] |
| **RTOS** | | |
| Tasks | ▪ OSEK tasks<br>▪ Alarms<br>▪ OIL files | Support of OSEK-Compliant RTOS[6] |

1) Refer to TargetLink Preparation and Simulation Guide.
2) Refer to TargetLink API Reference.
3) Refer to TargetLink Customization and Optimization Guide.
4) Refer to TargetLink Classic AUTOSAR Modeling Guide.
5) Refer to TargetLink Adaptive AUTOSAR Modeling Guide.
6) Refer to TargetLink Multirate Modeling Guide.

**Expert features**

Refer to Modeling With TargetLink at Expert Level on page 60 in this guide.

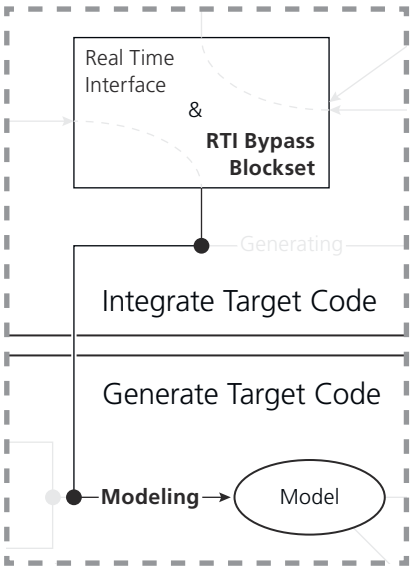
# Modeling with the RTI Bypass Blockset in TargetLink

**Introduction**

To model with the RTI Bypass Blockset in TargetLink for On-Target Bypassing.

**Sub use cases**

The map lists the sub use cases for *Generating Target Adaption Code for ECU Bypassing on page 44*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Modeling... | Feature | User Documentation |
|---|---|---|
| with the RTI Bypass blockset | RTI Bypass Blockset in TargetLink | Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing (📖 TargetLink Interoperation and Exchange Guide) |

**Expert features**

No such features.

# Optimizing C Code

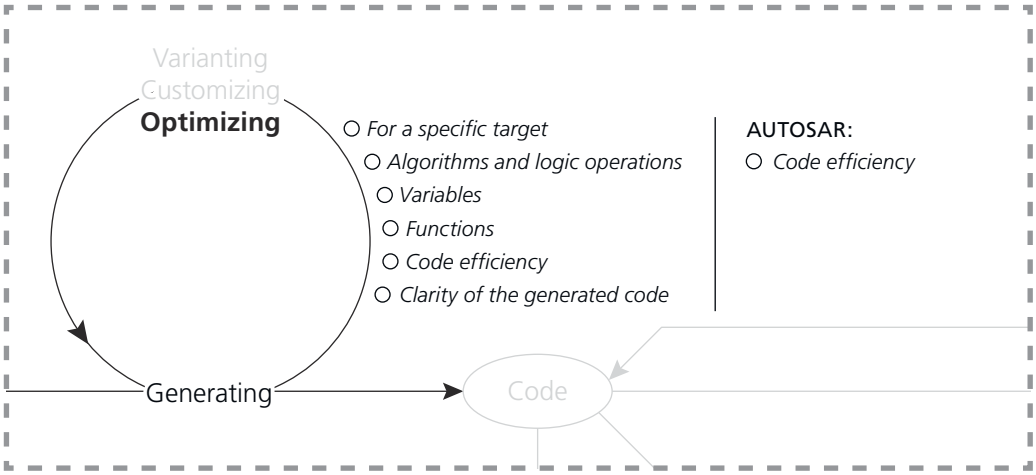
**Introduction**

TargetLink provides various methods to optimize the production code's execution time, stack consumption and code size. These methods are used by the Code

Generator when it generates and compiles the production code for the current TargetLink subsystem.

**Sub use cases**

The map lists the sub use cases for *Optimizing C Code*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Optimizing ... | Feature | User Documentation |
|---|---|---|
| For a specific target | Target optimization modules (TOMs) | Optimizing Target-Specific Code[1)] |
| Algorithms and logic operations | Control flow optimization | Introduction to Optimizing Production Code[1)] |
| Variables | ▪ Variable optimization attributes<br>▪ Scope reduction of variables<br>▪ Elimination of temporary and struct variables<br>▪ Variable range propagation | Introduction to Optimizing Production Code[1)] |
| | Block output saturation | Optimizing Block-Specific Code[1)] |

| Optimizing ... | Feature | User Documentation |
|---|---|---|
| Functions | ▪ C function reuse<br>▪ Variable propagation<br>▪ Scaling-invariant functions | Reusing C Functions and Variables for Identical Model Parts (Function Reuse)[1)] |
| | ▪ Merging external functions<br>▪ Scope reduction of functions<br>▪ Function optimization attributes<br>▪ Function interface | Introduction to Optimizing Production Code[1)] |
| | Table function reuse | Reducing Code Size via Table Function Reuse[2)] |
| Code efficiency | Code Generator options | Basics on Configuring the Code Generator for Production Code Generation[1)] |
| Clarity of the generated code | Code partitioning | Basics on Configuring the Code Generator for Production Code Generation[1)] |
| | Enumerations | Applying Simulink Enumeration Data Types in TargetLink[2)] |
| **Classic AUTOSAR** | | |
| Code efficiency | AUTOSAR Code Generator options | AUTOSAR-Related Code Generator Options[3)] |

1) Refer to TargetLink Customization and Optimization Guide.
2) Refer to TargetLink Preparation and Simulation Guide
3) Refer to TargetLink Classic AUTOSAR Modeling Guide.

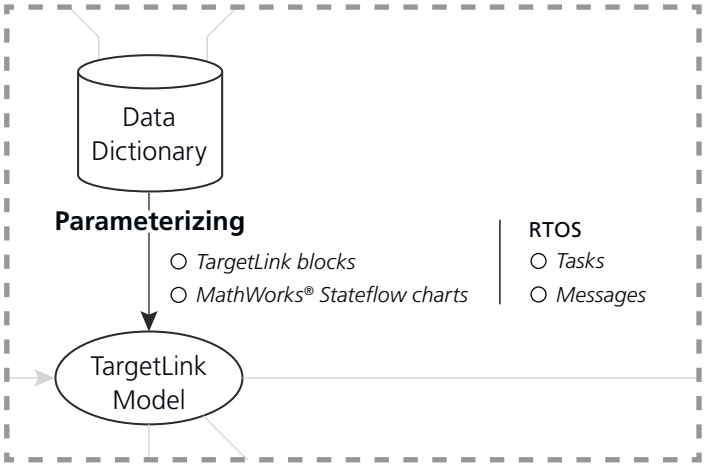
**Expert features** No such features.

## Parameterizing a Model

**Introduction** There are various ways to parameterize a model. TargetLink block properties can be accessed and modified via the block-specific dialog, the Property Manager or the TargetLink Data Dictionary.

**Sub use cases**

The map lists the sub use cases for *Parameterizing a Model*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Parameterizing ... | Feature | User Documentation |
|---|---|---|
| TargetLink blocks | ▪ Block dialog<br>▪ Property Manager<br>▪ Property inheritance<br>▪ Data Dictionary objects<br>▪ Mask parameter | Specifying TargetLink Block Properties[1)] |
| | M-Script Interface (API) | Using the TargetLink M-Script Interface (API)[2)] |
| | Autoscaling | Transforming Floating-Point Code to Fixed-Point Code (Scaling Variables)[1)] |
| MathWorks® Stateflow charts | Property Manager | Property Manager[3)] |
| | M-Script Interface (API) | Using the TargetLink M-Script Interface (API)[2)] |
| **RTOS** | | |
| Tasks | ▪ RTOS DD objects<br>▪ RTOS blocks | Managing RTOS Data for Multirate Models[4)] |

| Parameterizing ... | Feature | User Documentation |
|---|---|---|
| Messages | ▪ Global buffer<br>▪ OSEK message | Managing Intertask Communication[4)] |

1) Refer to TargetLink Preparation and Simulation Guide.
2) Refer to TargetLink Interoperation and Exchange Guide.
3) Refer to TargetLink Tool and Utility Reference.
4) Refer to TargetLink Multirate Modeling Guide.

**Expert features**

No such features.

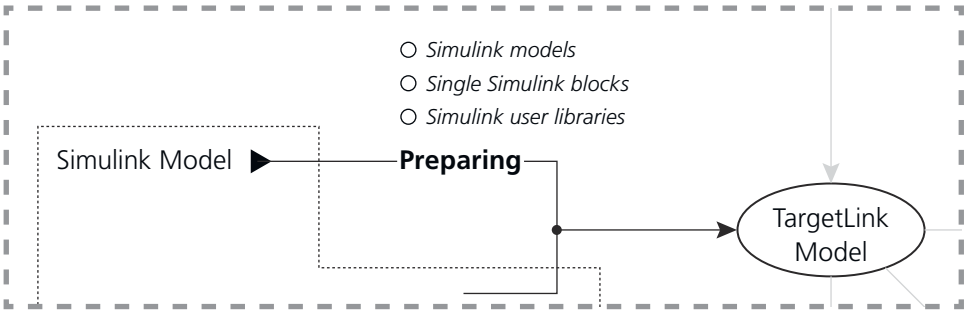
## Preparing a Simulink Model for TargetLink

**Introduction**

Preparing a Simulink system (block, subsystem, library, or model) for TargetLink means that TargetLink can generate production code for it, while it is still fully compatible with Simulink.

**Sub use cases**

The map lists the sub use cases for *Preparing a Simulink Model for TargetLink*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Preparing ... | Feature | User Documentation |
|---|---|---|
| Simulink models | System preparation tool | Making a Simulink Model TargetLink-Compliant[1)] |
| Single Simulink blocks | Enhance block command | |
| Simulink user libraries | System preparation tool | |

1) Refer to TargetLink Preparation and Simulation Guide.

**Expert features**

No such features.

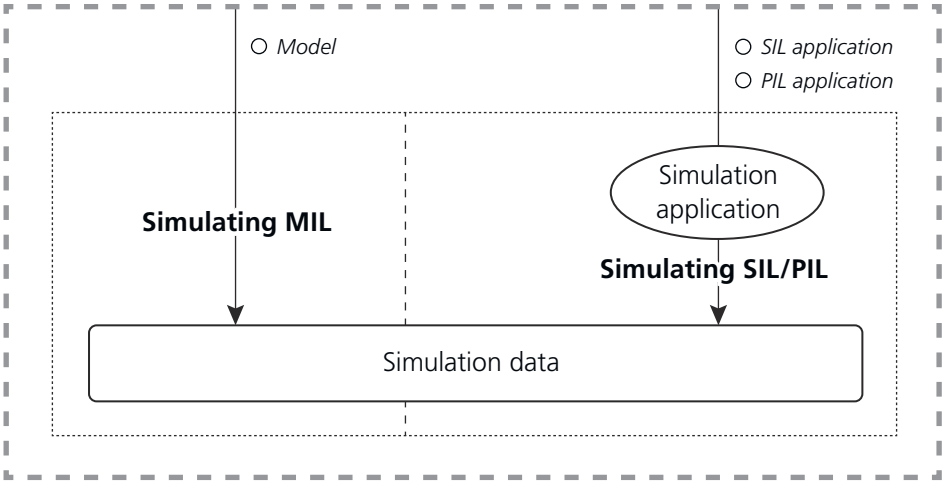
# Simulating MIL/SIL/PIL

**Introduction**

To verify the model behavior, you first have to simulate the graphical model specification (MIL simulation). This simulation serves as a reference for the subsequent SIL and PIL simulations with the generated production code. The SIL and PIL simulations help you validate the generated production code and profile the execution time and resource consumption.

**Sub use cases**

The map lists the sub use cases for *Simulating MIL/SIL/PIL*.




**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Simulating ... | Feature | User Documentation |
|---|---|---|
| Model | Model-in-the-loop simulation (MIL) | Simulating Models and Analyzing Simulation Results[1)]<br>Model-in-the-Loop Simulation Mode[1)] |
| SIL application | Software-in-the-loop simulation (SIL) | Software-in-the-Loop Simulation Mode[1)] |
| PIL application[2)] | Processor-in-the-loop simulation (PIL) | Processor-in-the-Loop Simulation Mode[1)] |

| Simulating ... | Feature | User Documentation |
|---|---|---|
| **Adaptive AUTOSAR** | | |
| SIL application of a software component | Software-in-the-loop simulation (SIL) | Basics on SIL Simulation for Adaptive AUTOSAR Models[3)] |

[1)] Refer to TargetLink Preparation and Simulation Guide.
[2)] Not available for Adaptive AUTOSAR.
[3)] Refer to TargetLink Adaptive AUTOSAR Modeling Guide.

**Expert features**

No such features.

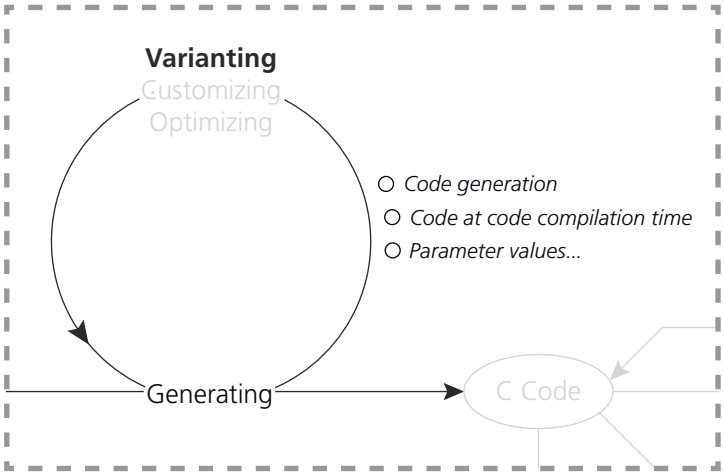
## Varianting C Code

**Introduction**

TargetLink supports different methods for varianting the generated C code. The methods mainly differ in the time at which a variant is coded or activated, that is, code generation, code compilation, or code execution.

**Sub use cases**

The map lists the sub use cases for *Varianting C Code*.



**Features and user documentation**

The table lists sub use cases, related TargetLink features, and the related user documentation:

| Varianting ... | Feature | User Documentation |
|---|---|---|
| Code generation | Code variants | Specifying Code Variants[1)] |
| Code at code compilation time | Variable vector widths | Specifying Variable Vector Widths via Width Macros[1)] |
| | Conditional control flows (preprocessor directives) | Specifying Conditional Control Flows[1)] |
| Parameter values at ECU run time | Data variants | Specifying Data Variants[1)] |

[1)] Refer to TargetLink Customization and Optimization Guide.

**Expert features**

No such features.

# Expert TargetLink Features Sorted by Use Case

**Introduction**

Beyond the features focusing on beginner, intermediate, and advanced users, there are also various features that exclusively focus on experts. The usage of these features requires a deep level of technical knowledge and experience.

**Where to go from here**

Information in this section



Information in other sections



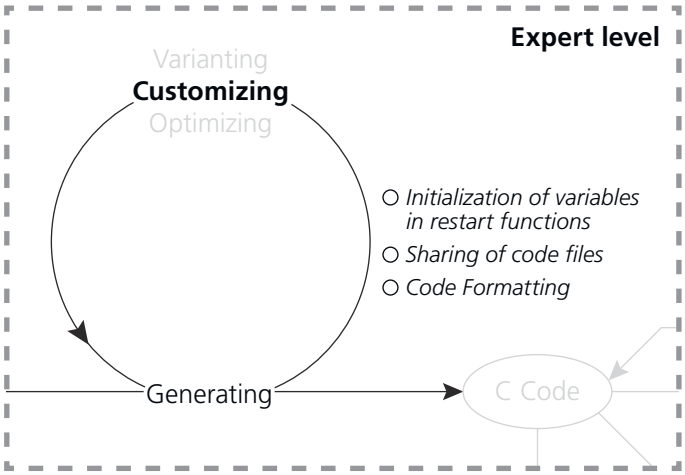
## Customizing Code at Expert Level

**Introduction**

To give you an in-depth look at the expert features and the user documentation that relate to *Customizing Code on page 38*.

**Sub use cases**

The map lists the sub use cases for *Customizing Code at Expert Level*.



**Expert features and user documentation**

The table lists sub uses cases, related TargetLink expert features, and the related user documentation:

| Customizing ... | Expert Feature | User Documentation |
|---|---|---|
| Initialization of variables in restart functions | Variable class templates | Example of Initializing Variables via Restart Functions[1)] |
| | | Basics on the Scope Reduction of Variables[1)] |
| Sharing of code files | Module ownership | Mapping Code Generation Units to Code (Module Ownership)[1)] |
| C code for integration in C++ projects | Code Generator option | Basics on Generating Code Compliant with C99/C11/C++[1)] |
| Code Formatting | XML style definition file | How to Edit the Style Definition File[1)] |
| | XSL style sheet | How to Select the Predefined Root Style Sheet[1)] |

1) Refer to TargetLink Customization and Optimization Guide.

**Related topics**

Basics

Customizing Code ..... 38

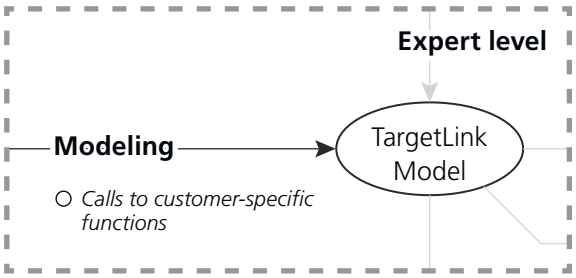
# Modeling With TargetLink at Expert Level

**Introduction**

To give you an in-depth look at the expert features and the user documentation that relate to *Modeling with TargetLink on page 48*.

**Sub use cases**

The map lists the sub use cases for *Modeling With TargetLink at Expert Level*.



**Expert features and user documentation**

The table lists sub uses cases, related TargetLink expert features, and the related user documentation:

| Modeling ... | Expert Feature | User Documentation |
|---|---|---|
| Calls to customer-specific functions | Extern C Stateflow symbols | Using External Code in Stateflow via a Scripting Mechanism (Customer-Specific Functions)[1)] |

[1)] Refer to TargetLink Preparation and Simulation Guide.

**Related topics**

Basics

Modeling with TargetLink ..... 48

# Getting Support

## Getting Further Support

**Getting further support**

If you need further support to solve your problem, refer to our Web sites at:

- http://www.dspace.com/goto?Problems_tl

  Here you find a constantly updated list of known problems and workarounds for TargetLink.

- http://www.dspace.com/goto?Support

  Here particularly the *Knowledge Base* including the FAQ section might be of help.

If you still cannot solve the problem, contact dSPACE Support (www.dspace.com/go/supportrequest).

# Glossary

**Introduction**

The glossary briefly explains the most important expressions and naming conventions used in the TargetLink documentation.

**Where to go from here**

Information in this section



# Numerics

**1-D look-up table** A look-up table that maps one input value (x) to one output value (y).

**2-D look-up table** A look-up table that maps two input values (x,y) to one output value (z).

# A

**Abstract interface** An interface that allows you to map a project-specific, physical specification of an interface (made in the TargetLink Data Dictionary) to a logical interface of a modular unit. If the physical interface changes, you do not have to change the Simulink subsystem or the partial DD file and therefore neither the generated code of the modular unit.

**Access function (AF)** A C function or function-like preprocessor macro that encapsulates the access to an interface variable.

See also read/write access function and variable access function.

**Acknowledgment** Notification from the RTE that a data element or an event message have been transmitted.

**Activating RTE event** An RTE event that can trigger one or more runnables.

See also activation reason.

**Activation reason** The activating RTE event that actually triggered the runnable.

Activation reasons can group several RTE events.

**Active page pointer** A pointer to a data page. The page referenced by the pointer is the active page whose values can be changed with a calibration tool.

**Adaptive AUTOSAR** Short name for the AUTOSAR *Adaptive Platform* standard. It is based on a service-oriented architecture that aims at on-demand software updates and high-end functionalities. It complements Classic AUTOSAR.

**Adaptive AUTOSAR behavior code** Code that is generated for model elements in Adaptive AUTOSAR Function subsystems or Method Behavior subsystems. This code represents the behavior of the model and is part of an adaptive application. Must be integrated in conjunction with ARA adapter code.

**Adaptive AUTOSAR Function** A TargetLink term that describes a C++ function representing a partial functionality of an adaptive application. This function can be called in the C++ code of an adaptive application. From a higher-level perspective, Adaptive AUTOSAR functions are analogous to runnables in Classic AUTOSAR.

**Adaptive AUTOSAR Function subsystem** An atomic subsystem used to generate code for an Adaptive AUTOSAR Function. It contains a **Function** block whose **AUTOSAR mode** is set to `Adaptive` and whose **Role** is set to `Adaptive AUTOSAR Function`.

**ANSI C** Refers to C89, the C language standard ANSI X3.159-1989.

**Application area** An optional DD object that is a child object of the DD root object. Each **Application** object defines how an ECU program is built from the generated subsystems. It also contains some experiment data, for example, a list of variables to be logged during simulations and results of code coverage tests.

**Build** objects are children of **Application** objects. They contain all the information about the binary programs built for a certain target platform, for example, the symbol table for address determination.

**Application data type** Abstract type for defining types from the application point of view. It allows you to specify physical data such as measurement data. Application data types do not consider implementation details such as bit-size or endianness.

**Application data type (ADT)** According to AUTOSAR, application data types are used to define types at the application level of abstraction. From the application point of view, this affects physical data and its numerical representation. Accordingly, application data types have physical semantics but do not consider implementation details such as bit width or endianness.

Application data types can be constrained to change the resolution of the physical data's representation or define a range that is to be considered.

See also implementation data type (IDT).

**Application layer** The topmost layer of the ECU software. The application layer holds the functionality of the ECU software and consists of atomic software components (atomic SWCs).

**ARA adapter code** Adapter code that connects Adaptive AUTOSAR behavior code with the Adaptive AUTOSAR API or other parts of an adaptive application.

**Array-of-struct variable** An array-of-struct variable is a structure that either is non-scalar itself or that contains at least one non-scalar substructure at any nesting depth. The use of array-of-struct variables is linked to arrays of buses in the model.

**Artifact** A file generated by TargetLink:

- Code coverage report files
- Code generation report files
- Metadata files
- Model-linked code view files
- Production code files
- Simulation application object files
- Simulation frame code files
- Stub code files

**Artifact location** A folder in the file system that contains an artifact. This location is specified relatively to a project folder.

**ASAP2 File Generator** A TargetLink tool that generates ASAP2 files for the parameters and signals of a Simulink model as specified by the corresponding TargetLink settings and generated in the production code.

**ASCII** In production code, strings are usually encoded according to the ASCII standard. The ASCII standard is limited to a set of 127 characters implemented by a single byte. This is not sufficient to display special characters of different languages. Therefore, use another character encoding, such as UTF-8, if required.

**Asynchronous operation call subsystem** A subsystem used when modeling *asynchronous* client-server communication. It is used to generate the call of the `Rte_Call` API function and for simulation purposes.

See also operation result provider subsystem.

**Asynchronous server call returns event** An RTE event that specifies whether to start or continue the execution of a runnable after the execution of a server runnable is finished.

**Atomic software component (atomic SWC)** The smallest element that can be defined in the application layer. An atomic SWC describes a single functionality and contains the corresponding algorithm. An atomic SWC communicates with the outside only via the interfaces at the SWC's ports. An atomic SWC is defined by an internal behavior and an implementation.

**Atomic software component instance** An atomic software component (atomic SWC) that is actually used in a controller model.

**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 import/export** Exchanging standardized software component descriptions between AUTOSAR tools.

**AUTOSAR subsystem** An atomic subsystem that contains a Function block whose AUTOSAR mode property is set to `Classic`. See also operation subsystem, operation call with runnable implementation subsystem, and runnable subsystem.

**AUTOSAR tool** Generic term for the following tools that are involved in the ECU network software development process according to AUTOSAR:

- Behavior modeling tool
- System-level tool
- ECU-centric tool

TargetLink acts as a behavior modeling tool in the ECU network software development process according to AUTOSAR.

**Autoscaling** Scaling is performed by the Autoscaling tool, which calculates worst-case ranges and scaling parameters for the output, state and parameter variables of TargetLink blocks. The Autoscaling tool uses either worst-case ranges or simulated ranges as the basis for scaling. The upper and lower worst-case range limits can be calculated by the tool itself. The Autoscaling tool always focuses on a subsystem, and optionally on its underlying subsystems.

# B

**Basic software** The generic term for the following software modules:

- System services (including the operating system (OS) and the ECU State Manager)
- Memory services (including the NVRAM manager)
- Communication services
- I/O hardware abstraction
- Complex device drivers

Together with the RTE, the basic software is the platform for the application layer.

**Batch mode** The mode for batch processing. If this mode is activated, TargetLink does not open any dialogs. Refer to How to Set TargetLink to Batch Mode on page 20.

**Behavior model** A model that contains the control algorithm for a controller (function prototyping system) or the algorithm of the controlled system (hardware-in-the-loop system). Can be connected in ConfigurationDesk via model ports to build a real-time application (RTA). The RTA can be executed on real-time hardware that is supported by ConfigurationDesk.

**Block properties** Properties belonging to a TargetLink block. Depending on the kind of the property, you can specify them at the block and/or in the Data Dictionary. Examples of block properties are:

- Simulink properties (at a masked Simulink block)
- Logging options or saturation flags (at a TargetLink block)
- Data types or variable classes (referenced from the DD)
- Variable values (specified at the block or referenced from the DD)

**Bus** A bus consists of subordinate bus elements. A bus element can be a bus itself.

**Bus element** A bus element is a part of a bus and can be a bus itself.

**Bus port block** Bus Inport, Bus Outport are bus port blocks. They are similar to the TargetLink Input and Output blocks. They are virtual, and they let you configure the input and output signals at the boundaries of a TargetLink subsystem and at the boundaries of subsystems that you want to generate a function for.

**Bus signal** Buses combine multiple signals, possibly of different types. Buses can also contain other buses. They are then called nested buses.

**Bus-capable block** A block that can process bus signals. Like bus port blocks, they allow you to assign a type definition and, therefore, a variable class to all the bus elements at once. The following blocks are bus-capable:

- Constant
- Custom Code (type II) block
- Data Store Memory, Data Store Read, and Data Store Write

- Delay
- Function Caller
- ArgIn, ArgOut
- Merge
- Multiport Switch (Data Input port)
- Probe
- Sink
- Signal Conversion
- Switch (Data Input port)
- Unit Delay
- Stateflow Data
- MATLAB Function Data

# C

**Calibratable variable** Variable whose value can be changed with a calibration tool during run time.

**Calibration** Changing the calibration parameter values of ECUs.

**Calibration parameter** Any ECU variable type that can be calibrated. The term *calibration parameter* is independent of the variable type's dimension.

**Calprm** Defined in a calprm interface. Calprms represent calibration parameters that are accessible via a measurement and calibration system.

**Calprm interface** An interface that is provided or required by a software component (SWC) via a port (AUTOSAR).

**Calprm software component** A special software component (SWC) that provides calprms. Calprm software components have no internal behavior.

**Canonical** In the DD, array-of-struct variables are specified canonically. Canonical means that you specify one array element as a representative for all array elements.

**Catalog file (CTLG)** A description of the content of an SWC container. It contains file references and file category information, such as source code files (`C` and `H`), object code files (such as `O` or `OBJ`), variable description files (`A2L`), or AUTOSAR files (`ARXML`).

**Characteristic table (Classic AUTOSAR)** A look-up table as described by Classic AUTOSAR whose values are measurable or calibratable. See also compound primitive data type

**Classic AUTOSAR** Short name for the AUTOSAR *Classic Platform* standard that complements Adaptive AUTOSAR.

**Classic initialization mode** The initialization mode used when the Simulink diagnostics parameter Underspecified initialization detection is set to `Classic`.

See also simplified initialization mode.

**Client port** A require port in client-server communication as described by Classic AUTOSAR. In the Data Dictionary, client ports are represented as DD ClientPort objects.

**Client-server interface** An interface that describes the operations that are provided or required by a software component (SWC) via a port (AUTOSAR).

**Code generation mode** One of three mutually exclusive options for generating TargetLink standard production code, AUTOSAR-compliant production code or RTOS-compliant (multirate RTOS/OSEK) production code.

**Code generation unit (CGU)** The smallest unit for which you can generate code. These are:

- TargetLink subsystems
- Subsystems configured for incremental code generation
- Referenced models
- DD CodeGenerationUnit objects

**Code output style definition file** To customize code formatting, you can modify a code output style definition file (XML file). By modifying this file, you can change the representation of comments and statements in the code output.

**Code output style sheets** To customize code formatting, you can modify code output style sheets (XSL files).

**Code section** A section of generated code that defines and executes a specific task.

**Code size** Amount of memory that an application requires specified in RAM and ROM after compilation with the target cross-compiler. This value helps to determine whether the application generated from the code files fits in the ECU memory.

**Code variant** Code variants lead to source code that is generated differently depending on which variant is selected (i.e., varianted at code generation time). For example, if the Type property of a variable has the two variants Int16 and Float32, you can generate either source code for a fixed-point ECU with one variant, or floating-point code with the other.

**Compatibility mode** The default operation mode of RTE generators. The object code of an SWC that was compiled against an application header generated in compatibility mode can be linked against an RTE generated in compatibility mode (possibly by a different RTE generator). This is due to using standardized data structures in the generated RTE code.

See also vendor mode.

**Compiler inlining** The process of replacing a function call with the code of the function body during compilation by the C compiler via inline expansion.

This reduces the function call overhead and enables further optimizations at the potential cost of larger code size.

**Composition** A structuring element in the application layer. A composition consists of software components and their interconnections via ports.

**Compound primitive data type** A primitive application data type (ADT) as defined by Classic AUTOSAR whose category is one of the following:

- COM_AXIS
- CUBOID
- CUBE_4
- CUBE_5
- CURVE
- MAP
- RES_AXIS
- VAL_BLK
- STRING

**Compute-through-overflow (CTO)** Calculation method for additions and subtraction where overflows are allowed in intermediate results without falsifying the final result.

**Concern** A concept in component-based development. It describes the idea that components separate their concerns. Accordingly, they must be developed in such a way that they provide the required functionality, are flexible and easy to maintain, and can be assembled, reused, or replaced by newer, functionally equivalent components in a software project without problems.

**Config area** A DD object that is a child object of the DD root object. The Config object contains configuration data for the tools working with the TargetLink Data Dictionary and configuration data for the TargetLink Data Dictionary itself. There is only one Config object in each DD workspace. The configuration data for the TargetLink Data Dictionary is a list of included DD files, user-defined views, data for variant configurations, etc. The data in the Config area is typically maintained by a Data Dictionary administrator.

**ConfigurationDesk** A dSPACE software tool for implementing and building real-time applications (RTA).

**Constant value expression** An expression for which the Code Generator can determine the variable values during code generation.

**Constrained range limits** User-defined minimum (Min) or maximum (Max) values that the user ensures will never be exceeded. The Code Generator relies on these ranges to make the generated production code more efficient. If no

Min/Max values are entered, the implemented range limits are used during production code generation.

**Constrained type** A DD Typedef object whose Constraints subtree is specified.

**Container** A bundle of files. The files are described in a catalog file that is part of the container. The files of a container can be spread over your file system.

**Container Manager** A tool for handling containers.

**Container set file (CTS)** A file that lists a set of containers. If you export containers, one container set file is created for every TargetLink Data Dictionary.

**Conversion method** A method that describes the conversion of a variable's integer values in the ECU memory into their physical representations displayed in the Measurement and Calibration (MC) system.

**Custom code** Custom code consists of C code snippets that can be included in production code by using custom code files that are associated with custom code blocks. TargetLink treats this code as a black box. Accordingly, if this code contains custom code variables you must specify them via custom code symbols.. See also external code.

**Custom code symbol** A variable that is used in a custom code file. It must be specified on the Interface page of custom code blocks.

**Customer-specific C function** An external function that is called from a Stateflow diagram and whose interface is made known to TargetLink via a scripting mechanism.

## D

**Data element** Defined in a sender-receiver interface. Data elements are information units that are exchanged between sender ports, receiver ports and sender-receiver ports. They represent the data flow.

**Data page** A structure containing all of the calibratable variables that are generated during code generation.

**Data prototype** The generic term for one of the following:

- Data element
- Operation argument
- Calprm
- Interrunnable variable (IRV)
- Shared or PerInstance Calprm
- Per instance memory

**Data receive error event** An RTE event that specifies to start or continue the execution of a runnable related to receiver errors.

**Data received event** An RTE event that specifies whether to start or continue the execution of a runnable after a data element is received by a receiver port or sender-receiver port.

**Data semantics** The communication of data elements with last-is-best semantics. Newly received data elements overwrite older ones regardless of whether they have been processed or not.

**Data send completed event** An RTE event that specifies whether to start or continue the execution of a runnable related to a sender acknowledgment.

**Data transformation** A transformation of the data of inter-ECU communication, such as end-to-end protection or serialization, that is managed by the RTE via transformers.

**Data type map** Defines a mapping between implementation data types (represented in TargetLink by DD **Typedef** objects) and application data types.

**Data type mapping set** Summarizes all the data type maps and mode request type maps of a software component (SWC).

**Data variant** One of two or more differing data values that are generated into the same C code and can be switched during ECU run time using a calibratable variant ID variable. For example, the **Value** property of a gain parameter can have the variants 2, 3, and 4.

**DataItemMapping (DIM)** A DataItemMapping object is a DD object that references a ReplaceableDataItem (RDI) and a DD variable. It is used to define the DD variable object to map an RDI object to, and therefore also the implementation variable in the generated code.

**DD child object** The DD object below another DD object in the DD object tree.

**DD data model** The DD data model describes the object kinds, their properties and constraints as well as the dependencies between them.

**DD file** A DD file (*.dd) can be a DD project file or a partial DD file.

**DD object** Data item in the **Data Dictionary** that can contain DD child objects and DD properties.

**DD object tree** The tree that arranges all DD objects according to the DD data model.

**DD project file** A file containing the DD objects of a DD workspace.

**DD root object** The topmost DD object of the DD workspace.

**DD subtree** A part of the DD object tree containing a DD object and all its descendants.

**DD workspace** An independent organizational unit (central data container) and the largest entity that can be saved to file or loaded from a DD project file. Any number of DD workspaces is supported, but only the first (DD0) can be used for code generation.

**Default enumeration constant** Represents the default constant, i.e., the name of an enumerated value that is used for initialization if an initial value is required, but not explicitly specified.

**Direct reuse** The Code Generator adds the instance-specific variables to the reuse structure as leaf struct components.

## E

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

**ECU software** The ECU software consists of all the software that runs on an ECU. It can be divided into the basic software, run-time environment (RTE), and the application layer.

**ECU State Manager** A piece of software that manages modes. An ECU state manager is part of the basic software.

**Enhanceable Simulink block** A Simulink® block that corresponds to a TargetLink simulation block, for example, the Gain block.

**Enumerated value** An enumerated value consists of an enumeration constant and a corresponding underlying integer value (enumeration value).

**Enumeration constant** An enumeration constant defines the name for an enumerated value.

**Enumeration data type** A data type with a specific name, a set of named enumerated values and a default enumeration constant.

**Enumeration value** An enumeration value defines the integer value for an enumerated value.

**Event message** Event messages are information units that are defined in a sender-receiver interface and exchanged between sender ports or receiver ports. They represent the control flow. On the receiver side, each event message is related to a buffer that queues the received messages.

**Event semantics** Communication of data elements with first-in-first-out semantics. Data elements are received in the same order they were sent. In simulations, TargetLink behaves as if data semantics was specified, even if you specified event semantics. However, TargetLink generates calls to the correct RTE API functions for data and event semantics.

**ExchangeableWidth** A DD object that defines code variants or improves code readability by using macros for signal widths.

**Exclusive area** Allows for specifying critical sections in the code that cannot preempt/interrupt each other. An exclusive area can be used to specify the mutual exclusion of runnables.

**Executable application** The generic term for offline simulation applications and real-time applications.

**Explicit communication** A communication mode in Classic AUTOSAR. The data is exchanged whenever data is required or provided.

**Explicit object** An explicit object is an object in production code that the Code Generator created from a direct specification made at a DD object or at a model element. For comparison, see implicit object.

**Extern C Stateflow symbol** A C symbol (function or variable) that is used in a Stateflow chart but that is defined in an external code module.

**External code** Existing C code files/modules from external sources (e.g., legacy code) that can be included by preprocessor directives and called by the C code generated by TargetLink. Unlike Custom code, external code is used as it is.

**External container** A container that is owned by the tool with that you are exchanging a software component but that is not the tool that triggers the container exchange. This container is used when you import files of a software component which were created or changed by the other tool.

# F

**Filter** An algorithm that is applied to received data elements.

**Fixed-Point Library** A library that contains functions and macros for use in the generated production code.

**Function AF** The short form for an access function (AF) that is implemented as a C function.

**Function algorithm object** Generic term for either a MATLAB local function, the interface of a MATLAB local function or a local MATLAB variable.

**Function class** A class that represents group properties of functions that determine the function definition, function prototypes and function calls of a function in the generated production code. There are two types of function classes: predefined function class objects defined in the `/Pool/FunctionClasses` group in the DD and implicit function classes (default function classes) that can be influenced by templates in the DD.

**Function code** Code that is generated for a modular unit that represents functionality and can have abstract interfaces to be reused without changes in different contexts, e.g. in different integration models.

**Function inlining** The process of replacing a function call with the code of the function body during code generation by TargetLink via inline expansion. This reduces the function call overhead and enables further optimizations at the potential cost of larger code size.

**Function interface** An interface that describes how to pass the inputs and outputs of a function to the generated production code. It is described by the function signature.

**Function subsystem** A subsystem that is atomic and contains a Function block. When generating code, TargetLink generates it as a C function.

**Functional Mock-up Unit (FMU)** An archive file that describes and implements the functionality of a model based on the Functional Mock-up Interface (FMI) standard.

## G

**Global data store** The specification of a DD DataStoreMemoryBlock object that references a variable and is associated with either a Simulink.Signal object or Data Store Memory block. The referenced variable must have a module specification and a fixed name and must be global and non-static. Because of its central specification in the Data Dictionary, you can use it across the boundaries of CGUs.

## I

**Implementation** Describes how a specific internal behavior is implemented for a given platform (microprocessor type and compiler). An implementation mainly consists of a list of source files, object files, compiler attributes, and dependencies between the make and build processes.

**Implementation data type (IDT)** According to AUTOSAR, implementation data types are used to define types on the implementation level of abstraction. From the implementation point of view, this regards the storage and manipulation of digitally represented data. Accordingly, implementation data types have data semantics and do consider implementation details, such as the data type.

Implementation data types can be constrained to change the resolution of the digital representation or define a range that is to be considered. Typically, they correspond to typedef statements in C code and still abstract from platform specific details such as endianness.

See also application data type (ADT).

**Implementation variable** A variable in the generated production code to which a ReplaceableDataItem (RDI) object is mapped.

**ImplementationPolicy** A property of data element and Calprm elements that specifies the implementation strategy for the resulting variables with respect to consistency.

**Implemented range** The range of a variable defined by its scaling parameters. To avoid overflows, the implemented range must include the maximum and minimum values the variable can take in the simulation application and in the ECU.

**Implicit communication** A communication mode in Classic AUTOSAR. The data is exchanged at the start and end of the runnable that requires or provides the data.

**Implicit object** Any object created for the generated code by the TargetLink Code Generator (such as a variable, type, function, or file) that may not have been specified explicitly via a TargetLink block, a Stateflow object, or the TargetLink Data Dictionary. Implicit objects can be influenced via DD templates. For comparison, see explicit object.

**Implicit property** If the property of a DD object or of a model based object is not directly specified at the object, this property is created by the Code Generator and is based on internal templates or DD **Template** objects. These properties are called implicit properties. Also see implicit object and explicit object.

**Included DD file** A partial DD file that is inserted in the proper point of inclusion in the DD object tree.

**Incremental code generation unit (CGU)** Generic term for code generation units (CGUs) for which you can incrementally generate code. These are:

- Referenced models
- Subsystems configured for incremental code generation

Incremental CGUs can be nested in other model-based CGUs.

**Indirect reuse** The Code Generator adds pointers to the reuse structure which reference the indirectly reused instance-specific variables.

Indirect reuse has the following advantages to direct reuse:

- The combination of shared and instance-specific variable.
- The reuse of input/output variables of neighboring blocks.

**Inline expansion** The process of replacing a function call with the code of the function body. See also function inlining and compiler inlining.

**Instance-specific variable** A variable that is accessed by one reusable system instance. Typically, instance-specific variables are used for states and parameters whose value are different across instances.

**Instruction set simulator (ISS)** A simulation model of a microprocessor thatcan execute binary code compiled for the corresponding microprocessor. This allows the ISS to behave in the same way as the simulated microprocessor.

**Integration model** A model or TargetLink subsystem that contains modular units which it integrates to make a larger entity that provides its functionality.

**Interface** Describes the data elements, NvData, event messages, operations, or calibration parameters that are provided or required by a software component (SWC) via a port (AUTOSAR).

**Internal behavior** An element that represents the internal structure of an atomic software component (atomic SWC). It is characterized by the following entities and their interdependencies:

- Exclusive area
- Interrunnable variable (IRV)
- Per instance memory
- Per instance parameter
- Runnable
- RTE event
- Shared parameter

**Interrunnable variable (IRV)** Variable object for specifying communication between the runnables in one atomic software component (atomic SWC).

**Interrupt service routine (ISR) function** A function that implements an ISR and calls the step functions of the subsystems that are assigned by the user or by the TargetLink Code Generator during multirate code generation.

**Intertask communication** The flow of data between tasks and ISRs, tasks and tasks, and between ISRs and ISRs for multirate code generation.

**Is service** A property of an interface that indicates whether the interface is provided by a basic software service.

**ISV** Abbreviation for instance-specific variable.

## L

**Leaf bus element** A leaf bus element is a subordinate bus element that is not a bus itself.

**Leaf bus signal** See also leaf bus element.

**Leaf struct component** A leaf struct component is a subordinate struct component that is not a struct itself.

**Legacy function** A function that contains a user-provided C function.

**Library subsystem** A subsystem that resides in a Simulink® library.

**Local container** A container that is owned by the tool that triggers the container exchange.

The tool that triggers the exchange transfers the files of a software component to this container when you export a software component. The external container is not involved.

**Local MATLAB variable** A variable that is generated when used on the left side of an assignment or in the interface of a MATLAB local function. TargetLink does not support different data types and sizes on local MATLAB variables.

**Look-up function** A function for a look-up table that returns a value from the look-up table (1-D or 2-D).

# M

**Macro** A literal representing a C preprocessor definition. Macros are used to provide a fixed sequence of computing instructions as a single program statement. Before code compilation, the preprocessor replaces every occurrence of the macro by its definition, i.e., by the code that it stands for.

**Macro AF** The short form for an access function (AF) that is implemented as a function-like preprocessor macro.

**MATLAB code elements** MATLAB code elements include MATLAB local functions and local MATLAB variables. MATLAB code elements are not available in the Simulink Model Explorer or the Property Manager.

**MATLAB local function** A function that is scoped to a MATLAB main function and located at the same hierarchy level. MATLAB local functions are treated like MATLAB main functions and have the same properties as the MATLAB main function by default.

**MATLAB main function** The first function in a MATLAB function file.

**Matrix AF** An access function resulting from a DD AccessFunction object whose VariableKindSpec property is set to `APPLY_TO_MATRIX`.

**Matrix signal** Collective term for 2-D signals implemented as matrix variable in production code.

**Matrix variable** Collective term for 2-D arrays in production code that implement 2-D signals.

**Measurement** Viewing and analyzing the time traces of calibration parameters and measurement variables, for example, to observe the effects of ECU parameter changes.

**Measurement and calibration system** A tool that provides access to an ECU for measurement and calibration. It requires information on the calibration parameters and measurement variables with the ECU code.

**Measurement variable** Any variable type that can be measured but not calibrated. The term *measurement variable* is independent of a variable type's dimension.

**Memory mapping** The process of mapping variables and functions to different memory sections.

**Memory section** A memory location to which the linker can allocate variables and functions.

**Message Browser** A TargetLink component for handling fatal (F), error (E), warning (W), note (N), and advice (A) messages.

**MetaData files** Files that store metadata about code generation. The metadata of each code generation unit (CGU) is collected in a DD Subsystem object that is written to the file system as a partial DD file called `<CGU>_SubsystemObject.dd`.

**Method Behavior subsystem** An atomic subsystem used to generate code for a method implementation. From the TargetLink perspective, this is an Adaptive AUTOSAR Function that can take arguments.

It contains a Function block whose AUTOSAR mode is set to `Adaptive` and whose Role is set to `Method Behavior`.

**Method Call subsystem** An atomic subsystem that is used to generate a method call in the code of an Adaptive AUTOSAR Function. The subsystem contains a Function block whose AUTOSAR mode is set to `Adaptive` and whose Role is set to `Method Call`. The subsystem interface is used to generate the function interface while additional model elements that are contained in the subsystem are only for simulation purposes.

**Microcontroller family (MCF)** A group of microcontroller units with the same processor, but different peripherals.

**Microcontroller unit (MCU)** A combination of a specific processor with additional peripherals, e.g. RAM or AD converters. MCUs with the same processor, but different peripherals form a microcontroller family.

**MIL simulation** A simulation method in which the function model is computed (usually with double floating-point precision) on the host computer as an executable specification. The simulation results serve as a reference for SIL simulations and PIL simulations.

**MISRA** Organization that assists the automotive industry to produce safe and reliable software, e.g., by defining guidelines for the use of C code in automotive electronic control units or modeling guidelines.

**Mode** An operating state of an ECU, a single functional unit, etc..

**Mode declaration group** Contains the possible operating states, for example, of an ECU or a single functional unit.

**Mode manager** A piece of software that manages modes. A mode manager can be implemented as a software component (SWC) of the application layer.

**Mode request type map** An entity that defines a mapping between a mode declaration group and a type. This specifies that mode values are instantiated in the software component (SWC)'s code with the specified type.

**Mode switch event** An RTE event that specifies to start or continue the execution of a runnable as a result of a mode change.

**Model Compare** A dSPACE software tool that identifies and visualizes the differences in the contents of Simulink/TargetLink models (including Stateflow). It can also merge the models.

**Model component** A model-based code generation unit (CGU).

**Model element** A model in MATLAB/Simulink consists of model elements that are TargetLink blocks, Simulink blocks, and Stateflow objects, and signal lines connecting them.

**Model port** A port used to connect a behavior model in ConfigurationDesk. In TargetLink, multiple model ports of the same kind (data in or data out) can be grouped in a model port block.

**Model port block** A block in ConfigurationDesk that has one or more model ports. It is used to connect the behavior model in ConfigurationDesk.

**Model port variable** A DD Variable object that represents a model port of a behavior model in ConfigurationDesk.

**Model-dependent code elements** Code elements that (partially) result from specifications made in the model.

**Model-independent code elements** Code elements that can be generated from specifications made in the Data Dictionary alone.

**Modular unit** A submodel containing functionality that is reusable and can be integrated in different integration models. The production code for the modular unit can be generated separately.

**Module** A DD object that specifies code modules, header files, and other arbitrary files.

**Module specification** The reference of a DD Module object at a **Function Block** (📖 TargetLink Model Element Reference) block or DD object. The resulting code elements are generated into the module. See also production code and stub code.

**ModuleOwnership** A DD object that specifies an owner for a module (module owner) or module group, i.e. the owning code generation unit (CGU) that generates the production code for it or declares the module as external code that is not generated by TargetLink.

# N

**Nested bus** A nested bus is a bus that is a subordinate bus element of another bus.

**Nested struct** A nested struct is a struct that is a subordinate struct component of another struct.

**Non-scalar signal** Collective term for vector and matrix signals.

**Non-standard scaling** A scaling whose LSB is different from `2^0 or whose Offset is not 0.`

**Nv receiver port** A require port in NvData communication as described by Classic AUTOSAR. In the Data Dictionary, nv receiver ports are represented as DD NvReceiverPort objects.

**Nv sender port** A provide port in NvData communication as described by Classic AUTOSAR. In the Data Dictionary, nv sender ports are represented as DD NvSenderPort objects.

**Nv sender-receiver port** A provide-require port in NvData communication as described by Classic AUTOSAR. In the Data Dictionary, nv sender-receiver ports are represented as DD NvSenderReceiverPort objects.

**NvData** Data that is exchanged between an atomic software component (atomic SWC) and the ECU's NVRAM.

**NvData interface** An interface used in NvData communication.

**NVRAM** Abbreviation of *non volatile random access memory*.

**NVRAM manager** A piece of software that manages an ECU's NVRAM. An NVRAM manager is part of the basic software.

## O

**Offline simulation application (OSA)** An application that can be used for offline simulation in VEOS.

**Online parameter modification** The modification of parameters in the production code before or during a SIL simulation or PIL simulation.

**Operation** Defined in a client-server interface. A software component (SWC) can request an operation via a client port. A software component can provide an operation via a server port. Operations are implemented by server runnables.

**Operation argument** Specifies a C-function parameter that is passed and/or returned when an operation is called.

**Operation call subsystem** A collective term for synchronous operation call subsystem and asynchronous operation call subsystem.

**Operation call with runnable implementation subsystem** An atomic subsystem that contains a Function block whose AUTOSAR mode property is set to `Classic` and whose Role is set to `Operation call with runnable implementation`.

**Operation invoked event** An RTE event that specifies to start or continue the execution of a runnable as a result of a client call. A runnable that is related to an operation invoked event represents a server.

**Operation result provider subsystem** A subsystem used when modeling *asynchronous* client-server communication. It is used to generate the call of the `Rte_Result` API function and for simulation purposes.

See also asynchronous operation call subsystem.

**Operation subsystem** A collective term for operation call subsystem and operation result provider subsystem.

**OSEK Implementation Language (OIL)** A modeling language for describing the configuration of an OSEK application and operating system.

# P

**Package** A structuring element for grouping elements of software components in any hierarchy. Using package information, software components can be spread across or combined from several software component description (SWC-D) files during AUTOSAR import/export scenarios.

**Parent model** A model containing references to one or more other models by means of the Simulink Model block.

**Partial DD file** A DD file that contains only a DD subtree. If it is included in a DD project file, it is called Included DD file. The partial DD file can be located on a central network server where all team members can share the same configuration data.

**Per instance memory** The definition of a data prototype that is instantiated for each atomic software component instance by the RTE. A data type instance can be accessed only by the corresponding instance of the atomic SWC.

**Per instance parameter** A parameter for measurement and calibration unique to the instance of a software component (SWC) that is instantiated multiple times.

**Physical evaluation board (physical EVB)** A board that is equipped with the same target processor as the ECU and that can be used for validation of the generated production code in PIL simulation mode.

**PIL simulation** A simulation method in which the TargetLink control algorithm (production code) is computed on a microcontroller target (physical or virtual).

**Plain data type** A data type that is not struct, union, or pointer.

**Platform** A specific target/compiler combination. For the configuration of platforms, refer to the **Code generation target settings** in the **TargetLink Main Dialog Block** block.

**Pool area** A DD object which is parented by the DD root object. It contains all data objects which can be referenced in TargetLink models and which are used for code generation. Pool data objects allow common data specifications to be reused across different blocks or models to easily keep consistency of common properties.

**Port (AUTOSAR)** A part of a software component (SWC) that is the interaction point between the component and other software components.

**Port-defined argument values** Argument values the RTE can implicitly pass to a server.

**Preferences Editor** A TargetLink tool that lets users view and modify all user-specific preference settings after installation has finished.

**Production code** The code generated from a code generation unit (CGU) that owns the module containing the code. See also stub code.

**Project folder** A folder in the file system that belongs to a TargetLink code generation project. It forms the root of different artifact locations that belong to this project.

**Property Manager** The TargetLink user interface for conveniently managing the properties of multiple model elements at the same time. It can consist of menus, context menus, and one or more panes for displaying property–related information.

**Provide calprm port** A provide port in parameter communication as described by Classic AUTOSAR. In the Data Dictionary, provide calprm ports are represented as DD **ProvideCalPrmPort** objects.

## R

**Read/write access function** An access function (AF) that *encapsulates the instructions* for reading or writing a variable.

**Real-time application** An application that can be executed in real time on dSPACE real-time hardware such as SCALEXIO.

**Receiver port** A require port in sender-receiver communication as described by Classic AUTOSAR. In the Data Dictionary, receiver ports are represented as DD **ReceiverPort** objects.

**ReplaceableDataItem (RDI)** A ReplaceableDataItem (RDI) object is a DD object that describes an abstract interface's basic properties such as the data type, scaling and width. It can be referenced in TargetLink block dialogs and is generated as a global macro during code generation. The definition of the RDI macro can then be generated later, allowing flexible mapping to an implementation variable.

**Require calprm port** A require port in parameter communication as described by Classic AUTOSAR. In the Data Dictionary, require calprm ports are represented as DD **RequireCalPrmPort** objects.

**RequirementInfo** An object of a DD RequirementInfo object. It describes an item of requirement information and has the following properties: Description, Document, Location, UserTag, ReferencedInCode, SimulinkStateflowPath.

**Restart function** A production code function that initializes the global variables that have an entry in the **RestartfunctionName** field of their variable class.

**Reusable function definition** The function definition that is to be reused in the generated code. It is the code counterpart to the reusable system definition in the model.

**Reusable function instance** An instance of a reusable function definition. It is the code counterpart to the reusable system instance in the model.

**Reusable model part** Part of the model that can become a reusable system definition. Refer to Basics on Function Reuse (TargetLink Customization and Optimization Guide).

**Reusable system definition** A model part to which the function reuse is applied.

**Reusable system instance** An instance of a reusable system definition.

**Root bus** A root bus is a bus that is not a subordinate part of another bus.

**Root function** A function that represents the starting point of the TargetLink-generated code. It is called from the environment in which the TargetLink-generated code is embedded.

**Root model** The topmost parent model in the system hierarchy.

**Root module** The module that contains all the code elements that belong to the production code of a code generation unit (CGU) and do not have their own module specification.

**Root step function** A step function that is called only from outside the production code. It can also represent a non-TargetLink subsystem within a TargetLink subsystem.

**Root struct** A root struct is a struct that is not a subordinate part of another struct.

**Root style sheet** A root style sheet is used to organize several style sheets defining code formatting.

**RTE event** The abbreviation of run-time environment event.

**Runnable** A part of an atomic SWC. With regard to code execution, a runnable is the smallest unit that can be scheduled and executed. Each runnable is implemented by one C function.

**Runnable execution constraint** Constraints that specify runnables that are allowed or not allowed to be started or stopped before a runnable.

**Runnable subsystem** An atomic subsystem that contains a Function block whose AUTOSAR mode property is set to `Classic` and whose Role is set to `Runnable`.

**Run-time environment (RTE)** A generated software layer that connects the application layer to the basic software. It also interconnects the different SWCs of the application layer. There is one RTE per ECU.

**Run-time environment event** A part of an internal behavior. It defines the situations and conditions for starting or continuing the execution of a specific runnable.

## S

**Scaling** A parameter that specifies the fixed-point range and resolution of a variable. It consists of the data type, least significant bit (LSB) and offset.

**Sender port** A provide port in sender-receiver communication as described by Classic AUTOSAR. In the Data Dictionary, sender ports are represented as DD SenderPort objects.

**Sender-receiver interface** An interface that describes the data elements and event messages that are provided or required by a software component (SWC) via a port (AUTOSAR).

**Sender-receiver port** A provide-require port in sender-receiver communication as described by Classic AUTOSAR. In the Data Dictionary, sender-receiver ports are represented as DD SenderReceiverPort objects.

**Server port** A provide port in client-server communication as described by Classic AUTOSAR. In the Data Dictionary, server ports are represented as DD ServerPort objects.

**Server runnable** A runnable that provides an operation via a server port. Server runnables are triggered by operation invoked events.

**Shared parameter** A parameter for measurement and calibration that is used by several instances of the same software component (SWC).

**Shared variable** A variable that is accessed by several reusable system instances. Typically, shared variables are used for parameters whose values are the same across instances. They increase code efficiency.

**SIC runnable function** A void (void) function that is called in a task. Generated into the Simulink implementation container (SIC) to call the root function that is generated by TargetLink from a TargetLink subsystem. In ConfigurationDesk, this function is called *runnable function*.

**SIL simulation** A simulation method in which the control algorithm's generated production code is computed on the host computer in place of the corresponding model.

**Simple TargetLink model** A simple TargetLink model contains at least one TargetLink Subsystem block and exactly one MIL Handler block.

**Simplified initialization mode** The initialization mode used when the Simulink diagnostics parameter Underspecified initialization detection is set to `Simplified`.

See also classic initialization mode.

**Simulation application** An application that represents a graphical model specification (implemented control algorithm) and simulates its behavior in an offline Simulink environment.

**Simulation code** Code that is required only for simulation purposes. Does not belong to the production code.

**Simulation S-function** An S-function that calls either the root step functions created for a TargetLink subsystem, or a user-specified step function (only possible in test mode via API).

**Simulink data store** Generic term for a memory region in MATLAB/Simulink that is defined by one of the following:

- A Simulink.Signal object
- A Simulink Data Store Memory block

**Simulink function call** The location in the model where a Simulink function is called. This can be:

- A Function Caller block
- The action language of a Stateflow Chart
- The MATLAB code of a MATLAB function

**Simulink function definition** The location in the model where a Simulink function is defined. This can be one of the following:

- Simulink Function subsystem
- Exported Stateflow graphical function
- Exported Stateflow truthtable function
- Exported Stateflow MATLAB function

**Simulink function ports** The ports that can be used in a Simulink Function subsystem. These can be the following:

- TargetLink ArgIn and ArgOut blocks

  These ports are specific for each Simulink function call.
- TargetLink InPort/OutPort and Bus Inport/Bus Outport blocks

  These ports are the same for all Simulink function calls.

**Simulink Function subsystem** A subsystem that contains a Trigger block whose Trigger Type is `function-call` and whose Treat as Simulink Function checkbox is selected.

**Simulink implementation container (SIC)** A file that contains all the files required to import production code generated by TargetLink into ConfigurationDesk as a behavior model with model ports.

**Slice** A section of a vector or matrix signal, whose elements have the same properties. If all the elements of the vector/matrix have the same properties, the whole vector/matrix forms a slice.

**Software component (SWC)** The generic term for atomic software component (atomic SWC), compositions, and special software components, such as calprm software components. A software component logically groups and encapsulates single functionalities. Software components communicate with each other via ports.

**Software component description (SWC-D)** An XML file that describes software components according to AUTOSAR.

**Stateflow action language** The formal language used to describe transition actions in Stateflow.

**Struct** A struct (short form for structure) consists of subordinate struct components. A struct component can be a struct itself.

**Struct component** A struct component is a part of a struct and can be a struct itself.

**Structure** A structure (long form for struct) consists of subordinate struct components. A struct component can be a struct itself.

**Stub code** Code that is required to build the simulation application but that belongs to another code generation unit (CGU) than the one used to generate production code.

**Subsystem area** A DD object which is parented by the DD root object. This object consists of an arbitrary number of **Subsystem** objects, each of which is the result of code generation for a specific code generation unit (CGU). The **Subsystem** objects contain detailed information on the generated code, including C modules, functions, etc. The data in this area is either automatically generated or imported from ASAM MCD-2 MC, and must not be modified manually.

**Supported Simulink block** A TargetLink-compliant block from the Simulink library that can be directly used in the model/subsystem for which the Code Generator generates production code.

**SWC container** A container for files of one SWC.

**Synchronous operation call subsystem** A subsystem used when modeling *synchronous* client-server communication. It is used to generate the call of the `Rte_Call` API function and for simulation purposes.

## T

**Table function** A function that returns table output values calculated from the table inputs.

**Target config file** An XML file named `TargetConfig.xml`. It contains information on the basic data types of the target/compiler combination such as the byte order, alignment, etc.

**Target Optimization Module (TOM)** A TargetLink software module for optimizing production code generation for a specific microcontroller/compiler combination.

**Target Simulation Module (TSM)** A TargetLink software module that provides support for a number of evaluation board/compiler combinations. It is used to test the generated code on a target processor. The TSM is licensed separately.

**TargetLink AUTOSAR Migration Tool** A software tool that converts classic, non-AUTOSAR TargetLink models to AUTOSAR models at a click.

**TargetLink AUTOSAR Module** A TargetLink software module that provides extensive support for modeling, simulating, and generating code for AUTOSAR software components.

**TargetLink Base Suite** The base component of the TargetLink software including the ANSI C Code Generator and the Data Dictionary Manager.

**TargetLink base type** One of the types used by TargetLink instead of pure C types in the generated code and the delivered libraries. This makes the code platform independent.

**TargetLink Blockset** A set of blocks in TargetLink that allow production code to be generated from a model in MATLAB/Simulink.

**TargetLink Data Dictionary** The central data container thats holds all relevant information about an ECU application, for example, for code generation.

**TargetLink simulation block** A block that processes signals during simulation. In most cases, it is a block from standard Simulink libraries but carries additional information required for production code generation.

**TargetLink subsystem** A subsystem from the TargetLink block library that defines a section of the Simulink model for which code must be generated by TargetLink.

**Task** A code section whose execution is managed by the real-time operating system. Tasks can be triggered periodically or based on events. Each task can call one or more SIC runnable functions.

**Task function** A function that implements a task and calls the functions of the subsystems which are assigned to the task by the user or via the TargetLink Code Generator during multirate code generation.

**Term function** A function that contains the code to be executed when the simulation finishes or the ECU application terminates.

**Terminate function** A runnable that finalizes a SWC, for example, by calling code that has to run before the application shuts down.

**Timing event** An RTE event that specifies to start or continue the execution of a runnable at constant time intervals.

**tllib** A TargetLink block library that is the source for creating TargetLink models graphically. Refer to How to Open the TargetLink Block Library on page 16.

**Transformer** The Classic AUTOSAR entity used to perform a data transformation.

**TransformerError** The parameter passed by the run-time environment (RTE) if an error occurred in a data transformation. The `Std_TransformerError` is a struct whose components are the transformer class and the error code. If the error is a hard error, a special runnable is triggered via the TransformerHardErrorEvent to react to the error.

In AUTOSAR releases prior to R19-11 this struct was named `Rte_TransformerError`.

**TransformerHardErrorEvent** The RTE event that triggers the runnable to be used for responding to a hard TransformerError in a data transformation for client-server communication.

**Type prefix** A string written in front of the variable type of a variable definition/declaration, such as `MyTypePrefix Int16 MyVar`.

## U

**Unicode** The most common standard for extended character sets is the Unicode standard. There are different schemes to encode Unicode in byte format, e.g., UTF-8 or UTF-16. All of these encodings support all Unicode characters. Scheme conversion is possible without losses. The only difference between these encoding schemes is the memory that is required to represent Unicode characters.

**User data type (UDT)** A data type defined by the user. It is placed in the Data Dictionary and can have associated constraints.

**Utility blocks** One of the categories of TargetLink blocks. The blocks in the category keep TargetLink-specific data, provide user interfaces, and control the simulation mode and code generation.

## V

**Validation Summary** Shows unresolved model element data validation errors from all model element variables of the **Property View**. It lets you search, filter, and group validation errors.

**Value copy AF** An access function (AF) resulting from DD AccessFunction objects whose AccessFunctionKind property is set to `READ_VALUE_COPY` or `WRITE_VALUE_COPY`.

**Variable access function** An access function (AF) that *encapsulates the access* to a variable for reading or writing.

**Variable class** A set of properties that define the role and appearance of a variable in the generated production code, e.g. CAL for global calibratable variables.

**VariantConfig** A DD object in the Config area that defines the code variants and data variants to be used for simulation and code generation.

**VariantItem** A DD object in the DD Config area used to variant individual properties of DD Variable and ExchangeableWidth objects. Each variant of a property is associated with one variant item.

**V-ECU implementation container (VECU)** A file that consists of all the files required to build an offline simulation application (OSA) to use for simulation with VEOS.

**V-ECU Manager** A component of TargetLink that allows you to configure and generate a V-ECU implementation.

**Vendor mode** The operation mode of RTE generators that allows the generation of RTE code which contains vendor-specific adaptations, e.g., to reduce resource consumption. To be linkable to an RTE, the object code of an SWC must have been compiled against an application header that matches the RTE code generated by the specific RTE generator. This is the case because the data structures and types can be implementation-specific.

See also compatibility mode.

**VEOS** A dSPACE software platform for the C-code-based simulation of virtual ECUs and environment models on a PC.

**Virtual ECU (V-ECU)** Software that emulates a real ECU in a simulation scenario. The virtual ECU comprises components from the application and the basic software, and provides functionalities comparable to those of a real ECU.

**Virtual ECU testing** Offline and real-time simulation using virtual ECUs.

**Virtual evaluation board (virtual EVB)** A combination of an instruction set simulator (ISS) and a simulated periphery. This combination can be used for validation of generated production code in PIL simulation mode.

# W

**Worst-case range limits** A range specified by calculating the minimum and maximum values which a block's output or state variable can take on with respect to the range of the inputs or the user-specified constrained range limits.