

TargetLink

Limitation Reference

For TargetLink 5.1

Release 2020-B – November 2020

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

© 2017 - 2020 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

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

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

The ability of dSPACE TargetLink to generate C code from certain MATLAB code in Simulink®/Stateflow® models is provided subject to a license granted to dSPACE by The MathWorks, Inc. MATLAB, Simulink, and Stateflow are trademarks or registered trademarks of The MathWorks, Inc. in the United States of America or in other countries or both.

Contents

About This Reference	5
TargetLink Limitations	7
General Limitations.....	8
ASAM MCD-2 MC File Generation Limitations.....	18
Classic AUTOSAR Limitations.....	20
Block-Specific Limitations.....	24
Code Generation Limitations.....	40
Component-Based Development Limitations.....	50
Data Dictionary Limitations.....	53
Documentation Generation Limitations.....	56
FMU Generation Limitations.....	57
Generic Multirate Limitations.....	58
Processing of Non-Scalar Signal Limitations.....	58
RTOS Multirate Limitations.....	60
Stateflow Limitations.....	65
MATLAB Language Limitations.....	71
Subsystem Creation Limitations.....	71
User Interface Limitations.....	77
V-ECU Implementation Generation Limitations.....	78
SIC Generation Limitations.....	78
Adaptive AUTOSAR-Related Limitations.....	80
Limitations for Arrays of Buses and Arrays of Structs.....	81
Simulink-Functions-Related Limitations.....	83
Glossary	85
Index	115





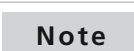



About This Reference

Contents

When working with TargetLink, you have to note the following limitations, which apply to TargetLink blocks as opposed to Simulink blocks:

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
	Indicates a hazard that, if not avoided, could result in property damage.
	Indicates important information that you should take into account to avoid malfunctions.
	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.

TargetLink Limitations

Where to go from here

Information in this section

General Limitations.....	8
ASAM MCD-2 MC File Generation Limitations.....	18
Classic AUTOSAR Limitations.....	20
Block-Specific Limitations.....	24
Code Generation Limitations.....	40
Component-Based Development Limitations.....	50
Data Dictionary Limitations.....	53
Documentation Generation Limitations.....	56
FMU Generation Limitations.....	57
Generic Multirate Limitations.....	58
Processing of Non-Scalar Signal Limitations.....	58
RTOS Multirate Limitations.....	60
Stateflow Limitations.....	65
MATLAB Language Limitations.....	71
Subsystem Creation Limitations.....	71
User Interface Limitations.....	77
V-ECU Implementation Generation Limitations.....	78
SIC Generation Limitations.....	78
Adaptive AUTOSAR-Related Limitations.....	80
Limitations for Arrays of Buses and Arrays of Structs.....	81

General Limitations

Introduction

The following general limitations apply to this Release of TargetLink.

Differences in MIL and SIL/PIL simulations

If the conditions listed below apply, the results of MIL and SIL/PIL simulations can differ even though the code was correctly generated:

- The same variable is specified for both a TargetLink Inport and a TargetLink Outport block, which is generally possible with mergeable variables.

Note

For AUTOSAR code, this can occur if the same data element of a port is referenced by both a TargetLink Inport and a TargetLink Outport block.

- The TargetLink Inport and TargetLink Outport blocks are connected to the TargetLink subsystem only by non-enhanced Simulink Inport and Outport blocks, or the TargetLink Inport block resides in the TargetLink subsystem on root level.
- The TargetLink Outport block resides in a conditionally executed subsystem that is triggered from within the TargetLink subsystem.

Function-call subsystems

If an output of a function-call triggered subsystem is fed back to one of its inports, the MIL simulation can change due to renaming blocks in this subsystem. The generated production code, however, is not affected.

Multiple calls of restart functions in simulations

If several S-functions are involved in a SIL/PIL simulation that substitutes a TargetLink subsystem in a SIL/PIL mode the restart functions of the DD-based CGU are called multiple times from the S-functions at simulation start.

MATLAB variables and Stateflow data

TargetLink does not support MATLAB variables and Stateflow data with the String built-in data type.

TargetLink does not support different Simulink data types for MATLAB variables and Stateflow data in library subsystems that are instantiated multiple times.

TargetLink does not support the following variables for MATLAB variables and Stateflow Data objects:

- Array-of-struct variables
- Non-scalar buses

String data types	TargetLink does not support string signals.
Property Manager	If you minimize the progress bar and maximize it again after the action was completed, the progress bar is still displayed and seems active. Clicking anywhere on the progress bar closes it.
Property Manager: Unloading library instances	A library cannot be unloaded if instances of its model elements are used in another model or library that is currently loaded.
Property Manager: Keyboard selection	TargetLink supports keyboard selection only on read-only cells. Using keyboard selection on cells that can be edited activates the input editor.
Simulink In Bus Element and Out Bus Element blocks	TargetLink does not support Simulink's In Bus Element and Out Bus Element blocks.
Plot channel specification of matrix signals	The plot channel specification of exactly two elements of a matrix signal via linear indices is not supported. TargetLink interprets these two elements as the row index and column index of one matrix element.
No support of template models	Using template models with TargetLink content is not supported.
Consistency check of signal dimensions	<p>During code generation, TargetLink checks the consistency of signal dimensions according to the following rule:</p> <ul style="list-style-type: none"> ▪ If the signal dimension(s) at the input of a block <i>does not</i> match the signal dimension(s) at the output of the block's predecessor, an error message is displayed. <p>Scalar signals and vectors/matrices that contain only one element are excluded from this consistency check.</p> <p>This check is a limited version of Simulink's <i>Vector/matrix block input conversion</i> check.</p>

Online parameter modification

The following limitations apply to online parameter modification:

- With the `tlSimInterface('Write', ...)` API function, the variable that is to be written must not reside in ROM or flash ROM. Whether a variable is stored in ROM or flash ROM depends on the linker. Refer to your linker manual or the MAP file. For details, refer to [tlSimInterface\('Write', hPlatform, propertyName, propertyValue, ...\)](#) ([TargetLink API Reference](#)).
- With the `tlSimInterface('CallFcn', ...)` API function, you can call only functions with a function prototype of the form:

```
void <fcnName>(void);
```

If you use the command to call functions with another prototype, the application will abort. For details, refer to [tlSimInterface\('CallFcn', hPlatform, propertyName, propertyValue, ...\)](#) ([TargetLink API Reference](#)).

Injecting or tunneling signals

During SIL/PIL simulation, signal injection/tunneling works only for variables whose address can be obtained or for variables that are protected by access functions whose address can be obtained.

For signal injection: The variable must not reside in ROM or flash ROM. Whether a variable is stored in ROM or flash ROM depends on the linker. Refer to the manual of the linker you are using or the MAP file.

The Simulink data stores used with signal injection must have either Simulink built-in data types (plain variables) or Simulink bus data types (structures/buses). The Simulink data stores with built-in data types must have the same data type.

Solution: Use the `Double` built-in data type for Simulink data stores. During simulation, TargetLink recalculates the double value of the Simulink data store to the scaled data type value of the corresponding code variable, if necessary.

The Simulink data stores used with signal tunneling must have either Simulink built-in data types (plain variables) or Simulink bus data types (structures/buses). The Simulink data stores with built-in data types must have the same data type.

Solution: Use the `Double` built-in data type for Simulink data stores. During simulation, TargetLink recalculates the scaled data type value of the code variable to the double value of the corresponding Simulink data store, if required.

Simulink Accelerator mode

Overflow detection and Min/Max logging are possible in Simulink Accelerator mode only if a signal's history is logged. You can enable signal history logging as follows:

- The Global logging option in the TargetLink Main Dialog block is set to Log signal histories, or
- The Global logging option in the TargetLink Main Dialog block is set to Log according to block data and the block-specific Data to log logging option is set to Signal history.

Encapsulated TargetLink systems	TargetLink does not support nested TargetLink subsystems, i.e., a TargetLink subsystem containing other TargetLink subsystems.
Names of generated files	Generated files must not have names that differ only in their capitalization because Microsoft Windows operating systems are not case-sensitive. For example, the subsystem_IDs <code>a</code> (lowercase) and <code>A</code> (uppercase) result in code files that overwrite each other.
Name of a TargetLink subsystem	With this version of TargetLink, the maximum number of characters for TargetLink subsystems is 58. Subsystem names must be valid C identifiers, i.e., they must consist only of letters, digits, and underscores.
Generating a model-linked code view file	<p>Displaying the model-linked code view file The Generate model-linked code view feature uses the MATLAB Web browser to display the model-linked code. This browser has some limitations that vary according to MATLAB Release and that affect the Generate model-linked code view feature. These limitations are:</p> <ul style="list-style-type: none"> ▪ Pages exceeding a certain limit cannot be displayed in the MATLAB Web browser. If this limit is reached, the code is displayed in the system Web browser instead. However, this browser cannot be used to navigate to the TargetLink model or to the Data Dictionary Manager. ▪ If it takes too long to calculate and load the linked Web sites, a timeout is triggered and a message is displayed (MATLAB is unable to link to this URL : <LocalPath>/MenuVar<No>). <p>Preprocessor directives ignored Generating the model-linked code view is based on C code and does not include considerations of any preprocessor directives or system paths of Include files.</p> <p>Enumeration types ignored DD enumeration typedefs cannot be traced back to the TargetLink Data Dictionary, because no HTML files are generated for these typedefs.</p> <p>Only TargetLink-generated code displayed C code, including links, is displayed only if it was generated by TargetLink. Code from user-defined source code files is not displayed. If code is generated from Custom Code blocks, only the block variables defined in these blocks can be linked. The same limitation applies to Stateflow custom code.</p> <p>No model-linked code view TargetLink does not support the generation of model-linked code view files for generated C++ code files.</p> <p>Variable actions only for direct referencing Code-to-code links can be generated only for variables that are directly referenced by name. They cannot be generated in the following cases, for example:</p> <pre>Int MyVar1 = 23; Int* pMyVar2 = &MyVar1; (*pMyVar2) = 23; /* no link to MyVar1 */</pre>

This limitation also applies if variable names are defined via preprocessor macros.

Show computation in model-linked code view

Tracing a variable from the model to its computation in the code is possible only if the variable was directly assigned its value. However, if a variable was assigned its value indirectly, tracing does not work. For example:

Computation of Var	Tracking of Var
<code>int Var; Var = 23;</code>	<code>Var = 23;</code> would be found
<code>int Var; Test(Var); function Test(int& rVar) { rVar = 37; }</code>	<code>rVar = 37;</code> would not be found
<code>int Var = 23; int* pVar = &Var; (*pVar) = 47;</code>	<code>(*pVar) = 47;</code> would not be found

Unsupported blocks

Since TargetLink is designed as a production C code generator, it focuses on models that contain only discrete blocks and state charts. Continuous time blocks such as continuous integrators are not supported in a TargetLink subsystem, although they can be used in other parts of a Simulink model. You can check the TargetLink library to find out whether a specific block is supported by TargetLink, or refer to [TargetLink Model Element Reference](#).

Note

- A TargetLink subsystem is a Simulink subsystem prepared for production code generation with TargetLink.
- Some blocks are supported in MIL simulation mode but ignored in the generated production code, for example, To Workspace, To File, Scope, Model Verification blocks.

Variable-size signals

TargetLink does not support variable-size signals.

Multidimensional signals

TargetLink does not support 3-D signals or signals with even more dimensions. Parameters with three or more dimensions are not possible.

Tip

The [TargetLink Utilities](#) provide workarounds for 3-D Look-Up Table parameter modeling.

Fixed-point types	Fixed-point types with a width other than 8, 16, or 32 bits are not supported by TargetLink. However, if such types are used in Simulink, you can have them mapped to TargetLink types of 8, 16, or 32 bit.
Saturation of block outputs	Block output signals and Stateflow data cannot be saturated in MIL simulation mode by TargetLink (? MIL simulation).
Multiple versions of the Data Dictionary Manager installed	<p>If multiple, different versions of TargetLink are installed on the same system, opening a .dd file via context menu or double-click on the file can lead to the stand-alone Data Dictionary Manager of the older TargetLink version being opened.</p> <p>In this case, open the .dd file by selecting Open with on the context menu and select the correct Data Dictionary Manager Stand-Alone application in the dialog that opens. Optionally, select the checkbox Always use this app to open .dd files.</p>
Editing DD objects	Cross-dependencies between properties are not checked if you edit Data Dictionary objects via the property value list in the Data Dictionary Manager. There are special dialogs for Variable, Typedef, DDIncludeFile, and Message objects that help you to avoid inconsistencies. The dialogs can be opened by double-clicking the objects. It is recommended to validate the Pool area (? Pool area) of the TargetLink Data Dictionary after modifications.
MSVC compiler must be installed at default location	If the Microsoft Developer Studio is not installed in the default folder, problems can occur during the build process of the host application because of inappropriate settings of the PATH variable in the makefile used for the build.
Data logging for bit variables	If a target processor supports bit addressable memory, and target-specific code is generated using a TargetLink TOM, the bit data type is then used to implement Boolean variables. Variables in the bit addressable memory cannot be logged via their memory addresses. Note that data logging using log macros is possible even for bit variables.
Buffer length for data logging	<p>In the Logging group box of the TargetLink block dialogs, you can limit the number of samples for each log variable stored in the data server (<code>logdata.maxnumvals</code> property). The limited buffer length is supported only for MIL simulations. It is not supported for:</p> <ul style="list-style-type: none"> Stateflow data logging in general Log variables of TargetLink blocks in SIL and PIL simulation mode Log variables selected in the Data Dictionary Manager after code generation


Reserved identifiers	<p>The following restrictions apply to Simulink model names and identifiers in the generated C code:</p> <ul style="list-style-type: none"> ▪ The name of a Simulink model must not be the same as reserved MATLAB keywords, such as <code>if</code> function or <code>switch</code> function. Thus, the name of a TargetLink subsystem must not be the same as reserved C keywords such as <code>if</code> or <code>switch</code>. ▪ Do not name a TargetLink subsystem or a generated C code module the same as a system header file defined by your compiler or by TargetLink, for example, <code>math</code>, <code>tl_types</code>, etc. ▪ Do not use reserved C keywords for variable, typedef, or function names in the generated C code, for example, <code>volatile</code>, <code>for</code>, <code>switch</code>, <code>case</code>, <code>typedef</code>, etc. Conflicts with names of registers can also occur for some target compilers.
Floating-point exception handling	<p>The generated code does not contain floating-point exception handlers. Be careful when performing arithmetic operations with floating-point variables to avoid divide-by-zero, underflow, and overflow exceptions.</p> <p>TargetLink's simulation frame detects floating-point exceptions during SIL simulation and PIL simulation, but you have to implement your own exception handling on the ECU.</p>
Block priorities and block scheduling	<p>Simulink block priorities are not considered in TargetLink's production code. If the block execution order is not completely determined by the control flow and data flow, TargetLink's block scheduling might be different from Simulink. This can lead to different simulation results (MIL versus SIL/PIL), for example, in data store memories.</p>
Model upgrade	<p>The upgrade routine in TargetLink 5.1 updates only models created under a TargetLink version ≥ 3.1. Models that were created under TargetLink 3.0 or even earlier versions cannot be updated directly to TargetLink 5.1. Instead, you must first perform an update to a TargetLink 3.1 ... 3.5 version before you can update to TargetLink 5.1.</p>
Checksum calculation	<p>TargetLink does not support checksum calculation for Simulink models saved as <code>*.slx</code> files.</p>
Detection of algebraic loops	<p>If a state holding block resides behind a Simulink inport block in an atomic subsystem and the Simulink inport is driven directly or indirectly by an output of the atomic subsystem, TargetLink's code generation detects an algebraic loop and throws an error message.</p>

Logging a simulation	<p>The following TargetLink limitations apply:</p> <ul style="list-style-type: none"> ▪ The specification of logging options for block states is not supported. ▪ Signals cannot be logged if the compiled signal width differs from the logged signal data. This can occur in MIL simulation mode for TargetLink blocks residing in Simulink For Each Subsystems. TargetLink does not support For Each Subsystems. ▪ TargetLink cannot perform logging and overflow detection during MIL simulation if the Log Dataset data to file option is enabled. Refer to Data Import/Export in the Simulink Configuration Parameters dialog. ▪ The output of Interpolation Using Prelookup blocks cannot be logged if it is a bus.
No signal logging for non-scalar bus signals	<p>TargetLink does not support signal logging for non-scalar bus signals.</p>
Logging of signals in Enabled, If Action, or Switch Action subsystems	<p>You cannot log a signal history in MIL simulation mode if the signal meets the following conditions:</p> <ul style="list-style-type: none"> ▪ The signal resides in either the Enabled, If Action, or Switch Action subsystem. ▪ The plot interval is different from Inf. <p>Instead, the minimum and maximum values of the signal are logged.</p>
No data logging for referenced subsystems in library blocks	<p>TargetLink does not support signal logging for instances of referenced subsystems if the instance is used as part of a Library block.</p>
Signal logging in referenced subsystems with Fast Restart	<p>If a model contains an instance of a referenced subsystem and the Fast Restart mode is active, TargetLink does not support the following for signals in the model :</p> <ul style="list-style-type: none"> ▪ Signal logging ▪ Min/Max logging ▪ Overflow detection
Logging restriction	<p>No Stateflow Local object is logged during MIL simulation if the simulation was started by either clicking the Simulink Run button or by using the Simulink <code>sim</code> command, instead of the TargetLink Start simulation button or <code>tl_sim</code> command.</p>

Performance of logging in MIL simulation mode


Due to the way MATLAB treats time series objects, MATLAB does not respond during/after MIL simulation (this might last for hours), if the following applies:

- The specified plot interval is too small.
- The number of plotted signals during MIL simulation is too large.

For detailed information on logging, refer to [Basics on Logging](#) ( [TargetLink Preparation and Simulation Guide](#)).

Logging signals in referenced models

For blocks that belong to a referenced model, simulated ranges are available for further processing only (e.g., overflow detection), if all the following conditions are satisfied:

- Logging for referenced models is enabled. For details refer to [Logging options for model referencing](#) ( [TargetLink Customization and Optimization Guide](#)).
- Logging for the relevant block is enabled.
- The simulation is started via the **Start simulation** button of a TargetLink window, or via the `tl_sim` command.

Simulation rewind

Rewinding (stepping backward) is not possible in SIL and PIL simulations. It is possible to a limited extent in MIL simulations, depending on whether or how signal data is logged:

- Logging is disabled or min/max values are logged:
Signal overflow warnings and min/max values are kept even if this data belongs to the "future" after the rewind.
- Signal history is logged:
Signal histories, signal overflow warnings, and min/max values remain valid during stepping back and forth if the following parameters are specified appropriately:
 - If signals are plotted, the **Plot Interval** parameter of the TargetLink Main Dialog block must be set to `inf`.
 - The number of samples that the Simulink log buffer holds must not be limited. Hence, the `SetNumberOfBufferedSimulinkLogSamples` parameter of the `tl_access_logdata` command must be set to `inf` (= default).

Artifact locations

The following limitations apply to [artifact locations](#) and [project folder specification](#):

- Relative paths specified in FolderStructure objects must not contain double dots.
- UNC paths are not supported.
- Windows environment variables are not supported.
- Artifact locations as defined in FolderStructure objects must not be changed after code generation so that following processes in TargetLink can find them

No data displayed in Scope block	If a Simulink Scope block resides in a referenced model and the simulation is started from a TargetLink dialog or via the <code>tl_sim</code> API command, the block does not display signal data.
No automatic update of plots during MIL simulations if started via API function	<p>TargetLink does not support an automatic update of plots during MIL simulations if all of the following conditions are fulfilled:</p> <ul style="list-style-type: none"> ▪ The FastRestart mode is active. ▪ The simulation was started in the MATLAB Command Window via the <code>tl_sim</code> API function. ▪ The signal to be plotted is located in a MIL subsystem or outside a TargetLink subsystem. <p>In this case, TargetLink updates the plots at the end of the simulation. TargetLink behaves as if the plot interval was set to <code>inf</code>. The actually specified plot interval is not considered. If the plot interval is not set to <code>inf</code>, TargetLink displays a message.</p>
Subsequently selected signal logging in Fast Restart mode	After the first simulation with activated Fast Restart mode, subsequently selected signal logging is not respected as long as the Fast Restart mode remains active.
No logging in Fast Restart mode started via function <code>sim</code>	If you start a simulation with the Simulink function <code>sim</code> in Fast Restart mode, signal logging is not possible.
Logging of variables without a MIL Handler block in Fast Restart mode	<p>Logging of variables during SIL and PIL simulation is not possible for a model if all of the following conditions are true:</p> <ul style="list-style-type: none"> ▪ Fast Restart mode is active and at least one simulation in Fast Restart mode has been performed. ▪ The model does not contain a MIL Handler block that is not commented out. ▪ The simulation is started via the Simulink Run button or the Simulink function <code>sim</code>.
Flexible port placement	TargetLink does not support moving ports of the TargetLink subsystem to a non-standard position, i.e., a position where input and output ports of a TargetLink subsystem do not reside on opposite sides. This positioning leads to subsequent problems when switching from MIL simulation to SIL simulation mode or when adding or removing a TargetLink simulation frame.
Warnings for blocks for which no code is generated	Sometimes TargetLink displays warnings or notes for blocks for which no code is generated.

Schedule as property of Simulink subsystems

TargetLink supports only the value **Sample time** for the **Schedule as property** of Simulink subsystems (MATLAB R2019a).

Mixed bitfield structures

It is not possible to log or calibrate structured variables in SIL/PIL, if they contain both bitfield elements and non-bitfield elements on the same level of the structure.

Related topics**References**

[tl_sim](#) ( [TargetLink API Reference](#))


ASAM MCD-2 MC File Generation Limitations

Introduction

The following limitations apply when you generate an ASAM MCD-2 MC (A2L) file for a TargetLink model:

ASAM MCD-1 interfaces

IF_DATA entries for an A2L **MODULE** element required to configure the ASAM MCD-1 (ASAP1b) interface for ECU access cannot be generated automatically. You can use the following option for adding the missing data to the A2L file:

- Enter the data manually or via MATLAB script at the appropriate `/Subsystems/<SubsystemName>/IF_DATA` object in the TargetLink Data Dictionary. `<SubsystemName>` denotes the TargetLink subsystem the ASAM MCD-2 MC (A2L) file is generated for.
- For detailed information, refer to [How to Create and Specify an IFData Object for a MODULE IF_DATA Block](#) ( [TargetLink Interoperation and Exchange Guide](#)).

Address information of static variables

For supported target compilers, TargetLink is able to generate an A2L file with address information for global and global static variables (if a corresponding MAP or ELF file exists). Depending on the information found in the MAP file, the addresses of local static variables can also be evaluated.

Note

If the A2L file does not contain the addresses of local variables, change the **Scope** property of the variable class to **Global**, or change the **Class** parameter of these variables to **DISP** or **CAL**.

Vectorized look-up tables	No input quantity is assigned for look-up table blocks with a vectorized input signal.
Common axis	No input quantity is assigned to the AXIS_PTS entry if this axis is shared by several MAP or CURVE characteristics with different input quantities. The input quantity, if known, is specified only via AXIS_DESCR of the corresponding MAP or CURVE characteristic.
Boundary points	<ul style="list-style-type: none"> ▪ An ASAM MCD-2 MC file entry for tables with boundary points is supported only if extrapolation is disabled. ▪ For 2-D look-up tables, boundary points must be selected for both axes.
Distances between table entries	If the Distances between table entries less than half of the implemented range (implemented range) option is selected for the Look-Up Table block, the look-up algorithm might not work properly if the condition is violated after the table data is calibrated.
FIR filter	The entry for FIR filter coefficients in the ASAM MCD-2 MC file does not take into account the limit that is set by Maximum sum of coefficients . You should therefore ensure that this condition applies if the FIR filter coefficients are calibrated.
Variant coding	<p>The description of data variants by VARIANT_CODING keyword is not supported.</p> <p>If production code is generated for several data variants, a separate entry is generated in the A2L file for each data variant instance of a calibratable or measurable variable, for example:</p> <pre> /begin CHARACTERISTIC dataVariant1.variable1 ... /end CHARACTERISTIC /begin CHARACTERISTIC dataVariant2.variable1 ... /end CHARACTERISTIC </pre>
Encoding of A2L files	<p>With Version 1.6.1 of the ASAM MCD-2 MC standard, A2L files now have to be encoded as UTF-8 with BOM.</p> <p>A2L files generated by TargetLink are encoded in UTF-8 without BOM to ensure compatibility with the pure ASCII format. This might cause problems with measurement and calibration systems that require the A2L files to be encoded as specified by Version 1.6.1 of the ASAM MCD-2 MC standard.</p>

If your calibration and measurement system requires the encoding as specified by Version 1.6.1 of the ASAM MCD-2 MC standard, open the generated A2L file in an editor that supports conversion to UTF-8 with BOM and save it accordingly or run a command line converter tool.

No entries for array-of-struct components in A2L export files

If you generate an A2L file during container export, the A2L import/export handler does not generate entries for the components of an array-of-struct variable.

Related topics

Basics

[Introducing A2L Files](#) ( [TargetLink Interoperation and Exchange Guide](#))

Classic AUTOSAR Limitations

Introduction

There are some limitations for this version of the TargetLink AUTOSAR Module.

AUTOSAR 2.x/3.x

TargetLink does not support the versions 2.x and 3.x of the AUTOSAR standard.

No SIL support for RTE-API-calls via AccessFunctions at terminated Runnable-Inports

TargetLink does not support building a SIL application if the following conditions are fulfilled:

- An Inport block of a Runnable subsystem references an RTE-API-function call that is modeled by an AccessFunction.
- The output of the inport is terminated.

Data Store Memory blocks modeling sender-receiver communication and Stateflow

A Data Store Memory block that is used to model sender-receiver communication for AUTOSAR must not be used by a Stateflow Data whose Scope is set to Data Store Memory.

Names of NvPort and NvDataElement

The names of a DD NvReceiverPort/NvSenderPort/NvSenderReceiverPort object and the corresponding DD NvDataElement object must be different in the following cases:

- Explicit sender-receiver communication is specified for the port.
- The associated DD SoftwareComponent object's SupportsMultipleInstantiation property is set to on.
- The DD NvReceiverPort/NvSenderPort/NvSenderReceiverPort object's IndirectAPI property is specified.

No import of data prototypes defined as splittable elements	TargetLink does not import data prototypes if they are defined in a different AUTOSAR file than the internal behavior (splittable elements).
Import or export of provide-require port prototypes	TargetLink does not support provide-require port prototypes for the following communication kinds: <ul style="list-style-type: none"> ▪ Client-server communication ▪ Mode communication
Differences in MIL and SIL/PIL simulations of externally triggered runnable subsystems	<p>There can be differences between MIL and SIL/PIL simulations of runnable subsystems that are function-call-triggered by signals located outside the TargetLink Subsystem that contains the runnable subsystems.</p> <p>These differences can occur in situations such as the following:</p> <ul style="list-style-type: none"> ▪ Their outputs (port blocks) are connected to a Merge block that is placed within the TargetLink Subsystem that contains the runnable subsystems. ▪ Their port blocks are connected to blocks that perform calculations and that are located outside the runnable system but inside the TargetLink Subsystem that contains the runnable subsystem. ▪ The output (port block) of a runnable system is connected to the input (port block) of another runnable system that is placed in the same TargetLink Subsystem and these ports specify an AUTOSAR communication kind that uses different buffers for reading and writing.
RestartFunctionName not "default" for AUTOSAR	If a variable is used in a Runnable and the RestartFunctionName property of its variable class is specified (the name is not set to default), this variable is initialized in a restart function named according to the RestartFunctionName property. However, the restart function is not called in the AUTOSAR restart hierarchy but in the main restart function. If the variable resides in a subsystem configured for incremental code generation or in a referenced model, it is not called at all.
Runnables and DD ActivationReason objects	Runnables that have their Kind property set to RestartFunction or TerminateFunction, cannot possess DD ActivationReason objects as child objects.
Explicit sender-receiver communication	<p>AUTOSAR port and data element must have different names if the following applies:</p> <ul style="list-style-type: none"> ▪ Explicit sender-receiver communication is specified for the port. ▪ One of the following properties is set to on: <ul style="list-style-type: none"> ▪ The software component's SupportsMultipleInstantiation property ▪ A port's IndirectAPI property

TargetLink RTOS Blockset	TargetLink does not support using RTOS Blocks and AUTOSAR Blocks in the same model.
Logging of RTE status and acknowledgment notifications	TargetLink does not support logging the RTE status and acknowledgment notifications in generic code generation mode . SenderComSpec and ReceiverComSpec blocks of the TargetLink AUTOSAR Block Library let you access the RTE status and acknowledgment notifications when modeling according to AUTOSAR.
Simulation of data element invalidation	TargetLink does not support the simulation of data invalidation in MIL simulation mode for struct-based data elements. If a data element represents a struct, i.e., the ingoing data signal of a SenderComSpec block is a bus, the ingoing data at input In1 is just passed through to output Out1 regardless of the signal at the invalidate port. In SIL/PIL simulation mode, signal invalidation for structs is supported.
Runnables in reused subsystems	TargetLink does not support function reuse for subsystems that contain runnables. Function reuse is supported for subsystems that are located in a runnable subsystem and do not contain a runnable subsystem.
Reuse of incremental subsystems in multiple runnables	TargetLink does not support the reuse of incremental subsystems in multiple runnables.
Updating frame models	Referenced model When you update a frame model from AUTOSAR data, TargetLink does not take referenced models into account. Blocks contained in referenced models are treated as non-existent in the model.
No pointers/pointer components in auto-generated PIM	Pointer variables and structures with at least one pointer component, e.g., look-up table map structures, cannot be implemented as auto-created PIM components. TargetLink generates different reuse struct types and reused function code in order to facilitate function reuse inside AUTOSAR software components with the SupportsMultipleInstantiation parameter set to on .
No simultaneous simulation of several SWC instances	The simultaneous simulation of several instances of one SWC type is not supported.
Runnables in external systems not supported	TargetLink does not support runnables placed in external systems or runnables with external function classes.

No AUTOSAR array types at non-AUTOSAR interfaces	Constraints that are solely defined at an array's scalar type are supported only at AUTOSAR interfaces in AUTOSAR code generation mode.
No standard mode code generation for explicit bus structs	<p>The following structured Typedef objects are not supported in standard code generation mode:</p> <ul style="list-style-type: none"> ▪ The structured Typedef object is referenced at a Bus Inport/Bus Outport block whose Apply explicit struct checkbox is selected. ▪ The structured Typedef object has one or more vector/matrix elements. ▪ These vector/matrix elements are AUTOSAR array types, i.e., they have the following AUTOSAR-specific properties: <ul style="list-style-type: none"> ▪ AutosarArrayWidth ▪ AutosarArrayElementRef
No port-defined argument values in operation call subsystems	TargetLink does not support port-defined argument values in an operation call with runnable implementation subsystem .
No access to activation vector for operation call subsystems	TargetLink does not support the runnable's access to its activation vector if the runnable is modeled as an operation call with runnable implementation subsystem .
Incorrect buffer for implicit communication modeled via data store blocks	<p>TargetLink's stub RTE might use incorrect buffering for the implicit communication of runnables that are modeled as follows:</p> <ul style="list-style-type: none"> ▪ A runnable subsystem accesses a Data Store Memory block that models implicit NvData or interrunnable communication. ▪ The runnable subsystem contains an operation call subsystem or operation result provider subsystem that accesses the same Data Store Memory block. <p>If the stub RTE contains no buffer, the return value of the <code>Rte_IRead</code> and <code>Rte_IrvIRead</code> API function might be changed by the call of the <code>Rte_Call</code> or <code>Rte_Result</code> API functions during the runnable's execution in SIL simulation mode.</p> <p>If the stub RTE contains a buffer, differences between MIL and SIL simulation are possible if the <code>Rte_Call</code> frame function writes directly to the frame variable.</p> <p>This also applies to operation call with runnable implementation subsystems if the server runnable belongs to the same SWC as the client runnable.</p>
No transformer error in operation call with runnable implementation subsystems	You cannot model transformer error logic in operation call subsystems whose Role is set to <code>Operation call with Runnable implementation</code> .

No transformer error for unidirectional get or set operations modeled via ports

You cannot model access to transformer errors when modeling unidirectional get or set operations via port blocks whose Kind is set to **Client-Server**.

Unsupported calls to RTE API functions

TargetLink does not provide native support for the following RTE API functions:

- Rte_SwitchAck
 - Rte_DRead
 - Rte_Trigger
 - Rte_IrTrigger
 - Rte_IFeedback
 - Rte_PBCon
-

AUTOSAR 4.x support

The limitations for the AUTOSAR 4.x support of this version of the TargetLink AUTOSAR Module are as follows:

- No support for variant management as defined in AUTOSAR 4.x. This includes no support for the formula language and system constant objects.
- TargetLink does not support TriggerInterfaces.
- For Runnables, the following AccessPoints are not supported: ExternalTriggeringPoint, InternalTriggeringPoint, and DataReceivePointByValue.

For a detailed list of limitations concerning the AUTOSAR support of TargetLink, contact dSPACE Support (www.dspace.com/go/supportrequest).

Classic AUTOSAR communication via Data Stores and Custom Code Type II

- The optimized, index-based write access for NvData communication via Data Store Write blocks is not supported for Data Store Write access via Custom Code (type II) block.
 - For Custom Code (type II) blocks whose property **Guaranteed complete write** is set to **off**, TargetLink does not support *write-only* access to data stores that are specified as follows:
 - Data stores with explicit AUTOSAR communication
 - Data stores with implicit AUTOSAR specification (IWrite)
-

Block-Specific Limitations

Introduction

The following block-specific limitations apply to this release of TargetLink.

Inheritance of block properties

Inheriting data types and [scalings](#) from preceding blocks is supported for only a subset of block types that do not perform any arithmetic operation:


- Assignment block (Y output inherits from Y0 input)

- Delay block
- D Flip-Flop block (output Q)
- D Latch block (output Q)
- Bus Inport/Bus Outport and InPort/OutPort blocks that are not used for interface specification of:
 - TargetLink subsystem
 - Incremental subsystem
 - External subsystem
 - Reused subsystem
 - Scaling-invariant subsystems
- Merge block
- MinMax block
- Multiport Switch block
- Rescaler block
- Saturation block
- Signal Conversion block
- Switch block
- Unit Delay block
- Unit Delay Reset Enabled block

The output of any of the following blocks inherits its data type and scaling from the preceding block if no DD Variable or Replaceable Data Item object is specified as an output variable:

- Bit Set
- Bit Clear
- Bitwise Operator
- Shift Arithmetic

For each of the above four blocks, the output block variable does not inherit Min/Max limits from a preceding block. If any input subsignal of any one of the above blocks inherits a user-defined data type with Min/Max limits, the corresponding output subsignal inherits the base data type of the user-defined type. Additionally, if a DD variable or a Replaceable Data Item object is specified as output variable of any of the above blocks or the Extract Bits block, then Min/Max limits specified for the DD object are propagated to any succeeding block via the Inherit properties option.

For details, refer to [Inheriting Signal Properties from Preceding Blocks](#) ( [TargetLink Preparation and Simulation Guide](#)).

Limitations for inheritance

Limitation for inheritance at Stateflow ports Stateflow inports can have the Stateflow properties Minimum and Maximum. Therefore, a Stateflow inport that is to inherit properties from the source block will not inherit the specifications for constrained limits (Max, Min).

No inheritance of constrained limits (Max, Min) if there is signal feedback If a block has several inputs at which it can inherit the constrained

limits (for example, the Merge or Multipoint Switch block), and if a subsignal at these inputs is fed back from the block's output signal, the constrained limits are not inherited for that subsignal, if the feedback signal line consists exclusively of blocks that also inherit or pass on their constrained limits.

Only scalar values for parameters when scaling information is inherited

- If the signal properties for the output variable of the Switch block are inherited from upstream blocks, only a scalar threshold can be set. When signal properties are inherited, the scalar property is automatically set for the parameter variable (threshold).
- If the signal properties for the output variable of the Saturation block are inherited from upstream blocks, only scalar limits can be set. When signal properties are inherited, the scalar property is automatically set for the parameter variables (upper limit and lower limit).

No overflow detection with property inheritance (Inherit properties) If a block has the Inherit properties checkbox selected, no scaling data (data type, LSB, offset, Min, Max) is available in MIL simulation mode. It is therefore not possible to perform overflow detection.

Inherit properties flag is not set during library preparation The Inherit properties checkbox is not set when a library is prepared for TargetLink, regardless of whether all the input signals (except for control input) of a Simulink block which is replaced are bus signals or not.

No overflow detection for input signals

Overflow detection for input signals of FIR Filter or look-up table blocks are not available for the MIL simulation mode.

Non-default sample time at Simulink blocks

Simulink lets you specify a sample time at most blocks. TargetLink supports sample time specification only at the following blocks:

- Constant
- Data Store Read
- Data Store Write
- Delay
- Discrete Filter
- Discrete State-Space
- Discrete-Time Integrator
- Discrete Transfer Fcn
- FIR Filter
- Sink
- Unit Delay
- Unit Delay Reset Enabled

For all other simulation blocks, you can specify the sample time only in the Simulink block dialog.

Duplicate input ports	Simulink supports duplicate input ports, i.e., a subsystem can contain several instances of the same input block. TargetLink does not support duplicate input ports.
Assignment block	The output variable's elements must have uniform scalings if the 1st/2nd dimension option is set to Index vector (port) or Starting index (port).
Action-port-triggered subsystem	<p>A subsystem that is triggered by an action port and is inside the TargetLink root system or the TargetLink root system itself (with an Action Port block on its topmost level) cannot be triggered from outside the TargetLink root system.</p> <p>A subsystem that is triggered by an Action Port block and is outside the TargetLink root system cannot be triggered from the TargetLink root system.</p>
Addfile block	Includes are not propagated to subsystems below the current subsystem if these subsystems are specified for incremental code generation.
Bit Clear block	The TargetLink Bit Clear block does not support 64-bit Simulink data types.
Bus Inport and Bus Output blocks	Signals must not be composed of data and event signals but of data signals only.
Bus-capable blocks	<p>Bus-capable TargetLink blocks (except Bus Port blocks) are subject to limitations if one of the following conditions is fulfilled:</p> <ul style="list-style-type: none"> ▪ They reference a DD Variable object with an struct type. ▪ They reference a DD Typedef object that is specified as an explicit struct type. ▪ The Inherit Properties checkbox is selected. <p>The following limitations apply:</p> <ul style="list-style-type: none"> ▪ Logging is not possible. ▪ Overflow detection is not possible.
Counter Alarm block	<ul style="list-style-type: none"> ▪ The CounterAlarm block is left untouched when a model is cleared from TargetLink.
Custom Code block (type I and II)	The Init section in the Custom Code file only results in production code if one or both of the following conditions are fulfilled. Otherwise, the missing production

code Init section can cause differences between Simulink and TargetLink simulation behavior (MIL differs from SIL).

- The Custom Code block resides in a conditionally executed subsystem or logically executed subsystem with *States when enabling*, *States when starting*, or *States when action is resumed* set to *reset*.
- The Custom Code block resides inside the TargetLink subsystem or inside a subsystem which is configured for incremental code generation. A TargetLink Function block resides inside the subsystem and has *Force init function* activated.

The search string, i.e., the `replacement.searchfor` property value, is limited to exactly one word, whitespace is not supported. Furthermore, TargetLink replaces only complete words that match the search string.

Custom Code (type I) block

Struct variables, i.e., variables with components, are not supported. However, accessing the leaf components of structs is supported.

As a general rule, variable classes are not taken into account in custom code S-functions.

The Custom Code (type I) block does not support input/output matrix signals.

If a replacement is specified, the **Use production code for floating-point simulation** option must not be enabled. Otherwise, S-function generation fails.

Matrix variables in custom code If matrix variables are used as custom code parameters, states, or work variables, the access to their elements via double indexing, e.g., `myParam[rowIdx][colIdx]`, in the S-function code for the custom code is not supported. Using the double index access leads to a compilation error for the custom code S-function.

To access matrix elements in the custom code S-function code, use the vector notation with linearized matrix indices, i.e., `myParam[idx]`, where $idx = \text{numOfCol} * (\text{rowIdx}) + \text{colIdx}$.

However, using this notation in the production code leads to incorrect code and incorrect simulation results. Therefore, the custom code file must provide separate code for the custom code S-function and for the production code. Additionally, the **Use production code for floating-point simulation** checkbox on the **Code & Logging** page of the Custom Code block must not be checked when building the custom code S-function.

Custom Code (type II) block

The following Simulink blocks cannot be enhanced to a TargetLink Custom Code (type II) block:

- Inport
- Outport
- Argument Inport
- Argument Outport
- Action Port
- Enable Port

- Reset Port
- Trigger Port
- Bus Creator
- Bus Selector

For Custom Code (type II) blocks, compiled dimensions are used if a dimension is not explicitly specified. In consequence, the `ReduceSingletonDimensions` Code Generator option is ignored in order to keep consistency with the custom code specification.

TargetLink supports an initial condition for the State and values for the Parameter variables only if these variables are mapped to their Simulink counterparts. For the variable mapping, refer to [How to Prepare Unsupported Simulink Blocks via Custom Code \(Type II\) Blocks](#) ([TargetLink Preparation and Simulation Guide](#)).

If an explicit struct is specified for the State or Parameter variable but the Simulink data type unknown due to an empty value of the `sldatatypestr` TargetLink property, TargetLink supports only complete Simulink IC structures as initial values. The number, dimensions, and linearized order of the IC struct components must match the number, dimensions and linearized order of the leaf struct components of the explicit struct specified for the State or Parameter variable. Otherwise, TargetLink cannot assign the values of the IC structure to the leaf struct components, which can result in differences between MIL and SIL/PIL simulation. For basic information on IC structures in TargetLink, refer to [Basics on Initializing Buses via Initial Condition Structures](#) ([TargetLink Preparation and Simulation Guide](#)).

Access to data stores specified at Custom Code (type II) blocks

During MIL simulation, TargetLink does not check the following:

- The Custom Code (type II) block accesses data stores which are specified at the Custom Code (type II) block.
- The access to the data stores corresponds to the specified behavior in the custom code.

Global data stores

Access to data stores In the following cases, the access to data stores must fulfill specific conditions:

- To access a data store that is specified by a `Simulink.Signal` object, the `Simulink.Signal` object must be associated with a `DD DataStoreMemoryBlock` object that is specified in the Data Dictionary. To do this, you must use the same name for both objects.
- To access a data store that is specified by a `Data Store Memory` block in the model that is located outside the current code generation unit, the `Data Store Memory` block must reference a `DD DataStoreMemoryBlock` block that is specified in the Data Dictionary.

If the `DD DataStoreMemoryBlock` object in the Data Dictionary does not specify AUTOSAR communication, it has to reference a `DD Variable` object that fulfills the following requirements:

- The `DD Variable` object must reference a dedicated `DD Module` object.
- The object name must not contain any context-specific name macros, e.g., `$S`.

- The variable must reference a DD VariableClass object with global Scope and non-static Storage.

Data Store blocks

If the sequence of reading and writing accesses is not unambiguously defined in the Simulink model, the sequence implemented in the generated code can differ, which leads to differences between MIL and SIL simulations. In addition, TargetLink does not support the use of priorities for defining the sequence of such accesses.

Data Store Memory block

- The data store variable must not be specified by an DD VariableClass object whose Scope property is set to `fcn_return`, `ref_param`, or `value_param`.
- TargetLink does not support the `ShareAcrossModelInstances` Simulink property.
- Access functions for data store variables are not supported if the data store variable is accessed via a Custom Code (type II) block.

Delay block

TargetLink does not support the Columns as channels (frame based) setting of the Simulink Input processing block property.

For a Delay Length > 1, or an Upper Limit >1, the following limitations apply:

- Bus signals are not supported at the input port.
- The `ddv` command is not supported.
- Variable vector width is not supported.

You cannot specify the initial value of bus signal elements by setting the initial condition parameter of blocks that pass a virtual bus signal to a Simulink Initial Condition Structure. Use a single value to initialize all bus elements with this same value (if possible) or a nonvirtual bus in combination with an IC structure.

Direct Look-Up Table (n-D) block

The Inputs select this object from table parameter cannot be set to 2-D Matrix as is possible with the Simulink counterpart block.

TargetLink does not support the combination of the following settings, which can be used with the Simulink counterpart block:

- Number of table dimensions set to 1
- Inputs select this object from table set to Column

Discrete Filter block

The following requirements must be fulfilled:

- The input and output data types must be either both floating-point or both integer types.
- Initial state values must be 0.
- External reset must be set to None.
- Coefficient vectors (numerator and denominator) and initial states must be specified in the block dialog.

- Input processing must be sample-based.
- Input and output must be scalars.
- TargetLink always implements the filter structure as **Direct form I**. The corresponding Simulink block property is ignored.

Discrete-Time Integrator block

- The TargetLink Discrete-Time Integrator block does not support the **Show state port** property of the Simulink Discrete-Time Integrator.
- The TargetLink Discrete-Time Integrator does not support the **level** and **sampld level** values of the **External reset** property of the Simulink Discrete-Time Integrator.
- The TargetLink Discrete-Time Integrator block does not support the **Accumulation** values of the **Integrator Method** property of the Simulink Discrete-Time Integrator.
- The TargetLink Discrete-Time Integrator block does not support the **Gain** value property of the Simulink Discrete-Time Integrator set to a different value than 1.0.
- Using non-scalar signals with the TargetLink Discrete-Time Integrator block is not supported.
- The limit output flag is not automatically set during model update.

Error-prone modeling scenario The block is used as follows:

- The block resides in an incremental subsystem or in a referenced model, and
- the subsystem or model is referenced in an enabled or triggered subsystem with the **States when enabling** option set held.

In consequence, Simulink keeps the value for one time step after reactivation of the conditional subsystem while TargetLink integrates. This leads to differences in MIL and SIL simulations.

Discrete Transfer Fcn block

The following requirements must be fulfilled:

- The input and output data types must be either both floating-point or both integer types.
- Initial state values must be 0.
- External reset must be set to **None**.
- Coefficient vectors (numerator and denominator) and initial states must be specified in the block dialog.
- Input processing must be sample-based.
- Input and output must be scalars.

Display block

TargetLink does not support Simulink's Display blocks whose **Floating display** option is enabled. For models containing such a block, TargetLink does not support the following for signals in the model :

- Signal logging
- Min/Max logging
- Overflow detection

A warning is displayed. Simulation continues normally.

Outports of Enabled and Action-triggered subsystems

TargetLink does not support the following Simulink behavior, if Simulink Initialization Mode is set to Classic:

Suppose the output of an Enable or Action-Trigger subsystem drives another enable output (either directly or via a series of Simulink outports) and these outports have the following settings:

- Driving output: Output when disabled == held
- Driven output: Output when disabled == reset

If the subsystem that includes the reset output is deactivated, Simulink does not only reset the driven output but resets also the driving output. Therefore, production code simulation differs from MIL simulation. To avoid these simulation differences set the **Output when disabled** setting of the driving output and the driven output to the same value.

Note

The outputs of a D Flip-Flop block behave like Simulink outports with the **Output when disabled** parameter set to reset. A triggered Stateflow system behaves like a conditional system with held outports.

Extract Bits block

These blocks are supported by TargetLink only when the **Output scaling mode** parameter in the Simulink block dialog is set to **Treat bitfield as an integer** (default).

Fcn block

If the Fcn block has scaled input signals or the output has a user-specified scaling, the scalings are not considered in the generated code. This means that scaling operations in the expression are missing. The Fcn block behaves like the Custom Code block in this respect. In this case, you have to ensure that the actual LSB and offset values equal the values displayed in the corresponding fields of the dialog. Otherwise, the output signal is invalid and no error message is issued.

FIR Filter block

The TargetLink Code Generator does not evaluate the `SLFilterFunctionReturn` variable class template in the Data Dictionary.

- Initial state values must be 0.
- Coefficients must be specified in the block dialog.
- Input processing must be sample-based.
- Input and output must be scalars.

For Iterator Subsystem block For this supported Simulink block, TargetLink does not support its Set next i (iteration variable) externally option.

From and Goto blocks In Simulink, From block and Goto block connections must not cross the boundaries of atomic subsystems. Unlike Simulink, TargetLink always treats some systems as atomic regardless of the Treat as atomic unit setting. These are systems that contain one of the following blocks:

- Function block
- Task block
- ISR block
- Runnable block (i.e., Function block configured for Classic AUTOSAR)

You can perform a simulation in MIL mode, but you cannot generate code.

Gain block TargetLink cannot avoid assign arithmetic if one of the inputs of the block's inputs has an offset.

TargetLink has two code patterns for matrix multiplications:

Assign Arithmetic	Scalar Product
<pre>for i for j out[i][j] = 0; for k out[i][j] += in1[i][k] * in2[k][j]</pre>	<pre>for i for j out[i][j] = in1[i][0] * in2[0][j] + ... + in1[i][m] * in2[m][j]</pre>

The Scalar Product pattern cannot be created if one of the block's inputs has an offset. In that case, the Assign Arithmetic pattern is created.

If the DoNotUseAssignArithmeticForAccumulation Code Generator option is set to on, error message E19524 is displayed.

Interpolation Using Prelookup block

The block is limited to two table dimensions. Extrapolation and round integer calculation are not supported. In Simulink, you can use a bus to combine index and fraction signals. This is not supported by TargetLink.

TargetLink always requires the Valid index input may reach last index Simulink block property to be enabled (refer to the Main Simulink block dialog page). Otherwise, MIL and SIL/PIL simulations differ. No error message is displayed, if this property is disabled.

If the ImplementLookUpTableFunctionsWithPointerArithmetics code generation option is enabled, the index calculation of boundary elements changes. For example, the index might be calculated to k=3 and f=0.99 instead of k=4 and f=0.

Look-Up Table block and Look-Up Table (2D) block

The following limitations apply for the support of Simulink n-D Lookup Table blocks:

- TargetLink supports only 1 and 2 as Number of table dimensions.
- TargetLink does not support floating-point types with equidistant tables.
- TargetLink does not support the input setting **Use one Input port for all input data**.
- TargetLink supports only the **Explicit values** value for the **Breakpoints** specification option.
- TargetLink supports only the following values for the **Interpolation method** option:
 - **Linear**
 - **Flat**
 - **Nearest**
- TargetLink supports only the value **Dialog** for the **Source** property for **Table Data and Breakpoints** (MATLAB R2020b).

Math block with Mod/Rem operator

- You must not specify an offset for the output, and the inputs do not accept offsets, either.
- Arbitrary LSB values are allowed, but TargetLink can suggest modifying the arbitrary LSB value or applying power-of-two scaling.
- 64-bit fixed-point operations in the denominator are not supported. This can occur, for example, if the denominator has a 32-bit input, a scaling that is more accurate than that of the numerator, and an output of the integer type.

Matrix Concatenate block

TargetLink does not support bus signals as input to this block type.

Merge block

You cannot specify the initial value of bus signal elements by setting the initial condition parameter of blocks that pass a virtual bus signal to a Simulink Initial Condition Structure. Use a single value to initialize all bus elements with this same value (if possible) or a nonvirtual bus in combination with an IC structure.

- The Simulink Merge block property **Allow unequal port widths** is currently not supported by TargetLink's Merge block.
- The input signals of a Merge block cannot be logged in MIL simulation mode.
- For a merge cascade only the last Merge block, or its output signal, respectively, can be logged.
- Missing overflow detection:

If a block's output is connected to an inport of a Merge block, the output signal cannot be checked for overflows.

Output block	<p>MIL simulation error MIL simulation aborts if all of the following conditions are met:</p> <ul style="list-style-type: none"> ▪ An Output or BusOutput block resides on the root level of the top-level model. ▪ The following two Simulink Configuration Parameters of the Data Import/Export section in the Save to workspace group are specified as follows: <ul style="list-style-type: none"> ▪ The Output checkbox is selected. ▪ The Format parameter is set to Dataset.
ParameterWriter block and StateWriter block (Simulink)	TargetLink does not support the Simulink ParameterWriter block and StateWriter block. TargetLink subsystems must not contain these blocks and Simulink ParameterWriter blocks or StateWriter blocks must not initialize any blocks in a TargetLink subsystem.
Permute Dimensions block	<p>TargetLink does not support bus signals as input to this block type.</p> <p>TargetLink supports this block only if the Order block parameter is set to [2 1] or [2;1].</p>
Prelookup block	<p>The output mode Common output variable for index and fraction is not supported. Extrapolation, and round integer calculation are not supported. In Simulink, you can use a bus to combine index and fraction signals. This is not supported by TargetLink.</p> <p>TargetLink always requires the Use last breakpoint for input at or above upper limitValid index input may reach last index Simulink block property to be enabled (refer to the Main Simulink block dialog page). Otherwise, MIL and SIL/PIL simulations differ. No error message is displayed, if this property is disabled.</p> <p>If the ImplementLookUpTableFunctionsWithPointerArithmetics code generation option is enabled, the index calculation of boundary elements changes. For example, the index might be calculated to k=3 and f=0.99 instead of k=4 and f=0.</p>
PreLook-Up Index Search block	<p>The Common output variable for index and fraction output mode is supported only for custom look-up functions. Extrapolation, round integer calculation, and vectorized output are not supported.</p> <p>TargetLink always requires the Use last breakpoint for input at or above upper limitValid index input may reach last index Simulink block property to be enabled (refer to the Main Simulink block dialog page). Otherwise, MIL and SIL/PIL simulations differ. No error message is displayed, if this property is disabled.</p>

Preprocessor IF

The following limitations apply for the Preprocessor IF block:

- Complex logical conditions are not supported for the `#ifdef` and `#if` defined modes, that is, only one symbol with or without a not operation is permitted.
- The symbol definition and its declaration(s) cannot be encapsulated by preprocessor constructs if the symbol is a structure or a component of a structure.
- When a Custom Code block is used in an `#ifdef` frame and the custom code contains not only the output section but also other sections, the definitions of the custom code interface variables do not become part of the `#ifdef` frame.

Product block

Matrix multiplication The following limitations apply to matrix multiplications:

- TargetLink supports only the multiplication of exactly two operands.
If *Matrix(*)* is selected from the block's Multiplication list, Number of inputs must be set to `**`.
- TargetLink cannot avoid assign arithmetic if one of the block's inputs has an offset.

TargetLink has two code patterns for matrix multiplications:

Assign Arithmetic	Scalar Product
<pre>for i for j out[i][j] = 0; for k out[i][j] += in1[i][k] * in2[k][j]</pre>	<pre>for i for j out[i][j] = in1[i][0] * in2[0][j] + ... + in1[i][m] * in2[m][j]</pre>

The Scalar Product pattern cannot be created if one of the block's inputs has an offset. In this case, the Assign Arithmetic pattern is created.

If the `DoNotUseAssignArithmeticForAccumulation` Code Generator option is set to `on`, error message E19524 is displayed.

Element-wise multiplication The following limitations apply to element-wise multiplications:

- More than two inputs and single vectorized inputs with more than two signals are not supported. Call the utility function `tl_replace_multi_inport_product` to fix the problem.
- Code generation for the input specification `//` is not allowed. Code generation for the input specification `/` is not allowed if the input signal has a width greater than one.

Reshape block

TargetLink does not support bus signals as input to this block type.

TargetLink supports this block only if the Output dimensions block parameter evaluates to a vector whose number of elements does not exceed 2 and the Output dimensionality block parameter is to one of the following values:

- 1-D array
- Column vector (2-D)

- Row vector (2-D)
- Customize

The Output dimensions block parameter is available if the Output dimensionality block parameter is set to **Customize**.

Rescaler block

The Rescaler block does not support bus signals.

TargetLink does not generate saturation code for TargetLink Rescaler blocks that process enumeration signals.

Selector block

If an external selection input is used to define the output values, all data input signals must have the same data type and scaling.

TargetLink does not support the **Starting and ending indices (port)** option.

Shift Arithmetic block

These blocks are supported by TargetLink only if the **Binary** points to shift parameter is set to **0** (default).

Sum block

For Sum blocks with exactly one input, TargetLink supports only the **All dimensions** value of the Sum over block parameter.

Trigger block/port

Constrained ranges are ignored if they are applied to signals fed to a Trigger block or the trigger port of a Discrete-Time Integrator or D Flip-Flop block.

For function-call trigger blocks whose option **Treat as Simulink function** is set, TargetLink does not support setting the **Function visibility** Simulink parameter to **scoped**.

Trigonometric Function block with tan

For the output of a **tan** function, any output with an unsigned data type must have an offset to create a symmetrically implemented output range.

Trigonometric function block with atan

The fixed-point implementation of **atan** must represent the $1/(\text{LSB}(\text{Input})^2)$ value as a 32-bit integer value. If $\text{LSB}(\text{Input})^2$, i.e., the LSB of the input in power-of-two notation, is too small, the $1/(\text{LSB}(\text{Input})^2)$ value cannot be represented correctly. Code generation stops with an error message.

Trigonometric Function block with atan2

The following rules apply to fixed-point implementation:

- TargetLink does not support 32-bit integer values as input for the fixed-point implementation of the **atan2** function.
- TargetLink does not support unsigned inputs and outputs for the fixed-point implementation of the **atan2** function.

- To calculate the atan2 , the second input (y) has to be squared and represented with the square of the LSB of the first input (x). The result must fit into a 32-bit integer. Otherwise, the code generation stops with an error message.

Unit Delay block

You cannot specify the initial value of bus signal elements by setting the initial condition parameter of blocks that pass a virtual bus signal to a Simulink Initial Condition Structure. Use a single value to initialize all bus elements with this same value (if possible) or a nonvirtual bus in combination with an IC structure.

Unit Delay Reset Enabled block

A conditional subsystem's output cannot be connected to a Merge block if all the following conditions are fulfilled:

- A Unit Delay Reset Enable block is located in the conditional subsystem
- The output of the conditional subsystem is connected to any output of the Unit Delay Reset Enable block
- The initialization mode of the model is set to Classic.

Tip

Use a Delay block instead.

Vector Concatenate block

TargetLink does not support bus signals as input to this block type.

Blocks not supporting matrix signals

The following blocks of TargetLink's blockset library do not support matrix signals:

- All the blocks having an Action port, Enable port, or Trigger port
- Bit Clear
- Bit Set
- Bitwise Operator
- Custom Code (type I)
- Discrete-Time Integrator (scalar signals only)
- Discrete State-Space
- Discrete Transfer Fcn (scalar signals only)
- Discrete Filter (scalar signals only)
- Extract Bits
- Fcn
- FIR Filter (scalar signals only)
- IF
- J-K Flip-Flop
- Preprocessor IF
- All the Sample blocks
- S-R Flip-Flop

- Shift Arithmetic
- Unit Delay Reset Enabled

Applied matrix signals

- of dimensions [1 x 1] will be reduced to scalars. Code generation is possible.
- of dimensions [M x N] will lead to error E03004 or E03061 when generating code. Code generation will fail.

Blocks not supporting the `ddv` command

The following blocks do not support the `ddv` command (refer to [How to Reference Values from DD Variable Objects to Block Variables](#) ([TargetLink Preparation and Simulation Guide](#))):

- **Fcn Block** ([TargetLink Model Element Reference](#))
- Preprocessor IF
- IF (supported Simulink block)

Simulink Rate-Transition block

TargetLink does not support Ensure data integrity during data transfer set to on.

TargetLink can generate special [simulation code](#) to ensure data integrity from Simulink's Rate Transition block if all of the following conditions are met:

- The block is placed between the externally triggered function-call subsystems.
- The block is directly connected to these subsystems
- The subsystems are not configured for function reuse.
- The interface of these subsystems is modeled via TargetLink port blocks that meet the following conditions:
 - They are not used for RTOS communication.
 - Their block variable is of global scope.
 - For floats, their block variable must be of the same data type.

Note

You can model SIC runnable functions that are time-triggered or asynchronous in the same TargetLink subsystem. However, the blocks for which a time-triggered SIC runnable function will be generated must be moved to a Function Call subsystem. Refer to [Details on Generating SIC Files for ConfigurationDesk That Contain Several SIC Runnable Functions](#) ([TargetLink Interoperation and Exchange Guide](#)).

This code is for simulation purposes only. This means TargetLink does not generate production code for the communication as specified via the Rate Transition block.


Probe block

TargetLink does not support the following outputs of the Probe block:


- Complex
- SampleTime


Related topics

Basics

[Basics on Inheriting Signal Properties](#) ( [TargetLink Preparation and Simulation Guide](#))

References

[DoNotUseAssignArithmeticForAccumulation](#) ( [TargetLink Model Element Reference](#))

[The TargetLink Simulation Blocks and Their TargetLink Properties](#) ( [TargetLink Model Element Reference](#))

Code Generation Limitations

Introduction

The following code generation limitations apply to this Release of TargetLink.

No `const` qualifier for function input

TargetLink does not support `const` qualification for function inputs that are implemented as function parameters, i.e., that are specified in one of the following ways:

- TargetLink InPort blocks with variables that reference DD VariableClass objects whose Scope property is set to `value_param` or `ref_param`.
- TargetLink Bus InPort blocks whose variables are specified as explicit structs and reference DD VariableClass objects whose Scope property is set to `ref_param`.

Const qualification of block output variables is possible only in the following cases:

- For the output variable of the TargetLink Constant block
- For TargetLink InPorts that are specified for indirect reuse, i.e., the InPort block references a DD VariableClass object that is specified as follows:


Property	Value
Scope	<code>ref_param</code>
FunctionReuse	<code>InstanceSpecific</code>
FunctionReuseAccessByPointer	<code>On</code>

CombineControlFlowStatements may interfere with formation of struct assignments

The CombineControlFlowStatements Code Generator option can interfere with the formation of struct assignments from leaf struct component assignments. This occurs if the Code Generator option merges loops with certain conditions, such as read or write accesses that cannot be rearranged into continuous struct assignments.


In these cases, the effect of the Code Generator option on code size and run time can be reduced, non-existent, or negative.

Unsupported Simulink configuration parameter	TargetLink does not support the Use algorithms optimized for row-major array layout Simulink configuration parameter.
Value copy access functions cannot be implemented as macros	TargetLink does not support value copy AFs that are implemented as macros (Macro AF).
N-Dimensional Variables	<p>The following limitations apply to the use of n-dimensional variables:</p> <ul style="list-style-type: none"> TargetLink does not support n-dimensional variables with a dimension greater than five. You can use variables with a dimension greater than two only for code generation from Data Dictionary. These variables can be used as Table variables of TL_Interpolation blocks whose TableDataSource property is set to Dialog. TargetLink does not provide a dialog for displaying or editing the value of a Variable object in the Data Dictionary with more than two dimensions. However, you can specify the width of variables with up to five dimensions and set their values by using MATLAB workspace variables.
Access to model parts	TargetLink does not support subsystems whose Permission attribute is set to NoReadOrWrite . Exception: The subsystem is a TL_BlackBox or resides in a TL_BlackBox.
Interblock optimization	<p>Interblock optimization does not move blocks into conditional branches if they contain a block variable whose Optimization property values contain MERGEABLE, even if MOVABLE is set.</p> <p>You can switch off this limitation with the ConsiderMergeableForBlockDiagramBasedSwitchOptimization Code Generator option at your own risk.</p>
Boolean data types at TargetLink inports	If a TargetLink subsystem inport has a Boolean data type, TargetLink's simulation frame performs a cast which can lead to differences between MIL and SIL/PIL simulations if the Simulink data type is not Boolean. For example, an input value of 8 does not change to 1 but remains 8.
Support of variable vector widths	TargetLink does not support a width variant set to 1. This also applies to the default variant.

For some blocks, propagation of `ExchangeableWidth` objects is supported only if specific settings are applied. For details, refer to [Details on Variable Vector Width Implementation](#) ( [TargetLink Customization and Optimization Guide](#)).

- **Assignment block:**
Variable vector width is supported only if the 1st dimension parameter is set to `Index vector (port)` or `Starting index (port)`. The 2nd dimension parameter must be set to `not used`.
- **Delay block:**
The `Delay length` parameter must be ≤ 1 . If set, the `Upper Limit` must also be ≤ 1 .

Compatibility of buses and predefined structs for Custom Code (type II) blocks

For the parameters, states, and work variables of Custom Code (type II) blocks, the Simulink data type of the bus must be specified via the `sldatatypestr` Simulink property. Otherwise, TargetLink cannot check the compatibility of bus and predefined struct. For details, refer to [Basics on the Compatibility of Buses and Predefined Structs](#) ( [TargetLink Customization and Optimization Guide](#)).

Support of Simulink enumerations

For the specification and implementation of enumeration data types in TargetLink, the following conditions have to be fulfilled:

- TargetLink does not support Simulink enumeration data types based on built-in integer data types.
- Unique integer values must be assigned to each enumeration constant. TargetLink does not support ambiguous integer values.
- TargetLink enumeration data types must have a corresponding Simulink enumeration data type.

Note

For explicit TargetLink enumeration data types and DD-based code generation, TargetLink does not check the correspondence of Simulink and TargetLink enumeration data types.

- If no DD enumeration template object exists: Block variables of enumeration type can reference an initial value from the TargetLink Data Dictionary only by casting its integer value to an enumeration. This is possible with the `ddv` command, for example, by entering `BasicColors(ddv('MyInitialDDEnumVar'))` in the `Value` field.

TargetLink does not support the following:

- Enumerated variables in mathematical calculations, e.g., multiplications.
- Saturation of enumerated variables.
- Enumerated variables at AUTOSAR interfaces. You can use an integer type with `Verbal Conversion Table` instead, for example.
- Enumerated message variables for intertask communication in RTOS code generation mode.

- Generation of saturation code for Simulink Data Type Conversion blocks processing enumeration signals. This limitation also applies to TargetLink Rescaler blocks, which are masked Simulink Data Type Conversion blocks.
- Grounds having a Simulink enumeration data type whose default enumeration constant value is not 0.
- Enumeration data types for the following table parameters:
 - Axis parameter (Look-Up Table block)
 - Axis#1 and Axis#2 parameters (Look-Up Table (2D) block)
 - Axis parameter (Prelookup block)

The following limitation applies to DD Variable objects that reference the IMPLICIT_ENUM typedef object:

- Initial values are not allowed.

TargetLink does not allow using IMPLICIT_ENUM typedef objects for:

- Components of structured variables
- Variables referencing a DD-based module (code generation straight from the Data Dictionary)
- Interface variables of incremental and external systems
- DD ReplaceableDataItem objects
- Variables referenced in DD DataItemMapping objects

For Custom Code blocks (Type II) the following limitations apply if the parameters, states, and work variables meet both of these conditions:

- An explicit struct is specified.
- No Simulink data type is specified via `sldatatypestr`.

In consequence, TargetLink does not check the validity of the TargetLink data types and scalings of the leaf bus elements with respect to enumerations.

This also applies for work variables that have a plain variable specified.

For the Assignment block, the following limitation applies:

- If the Element indices are specified via enumeration constants, only the constants' integer values are taken into account by the TargetLink Code Generator.

Function reuse

Parameters of reused incremental code generation units cannot be used for values that are not instance-specific.

Note

No warning is generated.

Unsigned integer constants

For unsigned integer constants smaller than INT32MAX, TargetLink does not append the suffix "u" hence the compiler applies a signed data type. In

consequence, the signed data type may be larger than actually required, for example, `Int32` instead of `UInt16`.

Inconsistent behavior of the ConformingFreestandingImplementation option

Activation of this option avoids standard `math.h` functions in the generated code, and a TargetLink-specific implementation is used instead. However, a TargetLink-specific implementation exists only for the `fabs` function.

Missing function interface variables

Suppose that a TargetLink block's output variable has a struct data type and the variable is fed to a Simulink input or output port block of an atomic subsystem. If the struct variable does not arrive as a whole but only as a subset of components (e.g., due to an interposed Bus Selector block), the Simulink port cannot inherit the variable's struct data type.

Access functions

Interface variables of custom code Custom code cannot be modified by replacing occurrences of variables or components of struct variables defined to be implemented using access functions. If custom code is optimized by eliminating its input and output variables, the custom code can contain access function calls.

Name macro not supported for components of DD structs

If a structure variable is specified in the Data Dictionary, none of its components must use the `$S` and `$B` name macros. The only permitted name macros are `$M`, `$T`, `$(Type)`, `$(BaseType)`, `$D`, `$(Group)` and `$I`.

No name macros specific to a model or code generation unit

If you use the code generation straight from the Data Dictionary feature, not all the available name macros can be used. As an example, you must not use `$M` as it does not work properly for DD-based modules. If you do so nevertheless, different model-based [code generation units](#) would result in different names. When several model-based code generation units are simulated, this might lead to code that cannot be compiled or to varying simulation results.

\$(FunctionName) name macro not supported for DD module objects

TargetLink does not support the `$(FunctionName)` name macro for the `NameTemplate` property in DD module objects. You can use one of these workarounds to specify the module that is referenced by a DD `LookupFunctionTemplate` object, refer to [Reducing Code Size via Table Function Reuse](#) ([TargetLink Preparation and Simulation Guide](#)).

No pointer-to-struct components in map structures

You cannot use a pointer-to-struct component (component of a DD structure whose root has the `ref_param` variable class scope) as an indirectly referenced map structure element. If you specify this in the block diagram, TargetLink aborts code generation and displays an error message about this limitation.

Map structure and pointer-to-struct components	If you use templates to have map structures generated, and if at least one of the elements is a pointer-to-struct component or references one (component of a DD structure whose root has the ref_param), code generation is aborted with an error message.
Pointer as return value	Calling a function with a return value of pointer type is not supported.
Structure as return value or function argument	A function with a return value or formal function argument of structure type is not supported.
Arithmetic operations with bitfields	TargetLink does not support arithmetic operations the result of which is a bitfield type. However, it is possible to use it for logical and relational operations.
Bit width of bitfields	Code generation is currently supported for operands that are mapped to a full C type variable (8-, 16-, 32-, and 64-bit variables) and single bits only. Variables consisting of a different number of bits (for example, 5 bits) must not be used.
Data type definitions	Type definitions in the Pool area of the TargetLink Data Dictionary must be scalar, which means the width property must be empty (Width=[]). You can create vector or matrix variables by using a scalar data type and setting the Width property of the variable as needed.
SIL/PIL simulation with RESTART function	A RESTART function cannot be called during SIL/PIL simulation if no root step function has been generated.
FCN_RETURN for system outputs only	Only system outputs can be defined with a variable class of the FCN_RETURN type. This can be in either a TargetLink OutPort block or a block that is connected directly to a Simulink Outport block.
Data variants for reused subsystems	Parameters that are reused (part of the reuse structure) are not supported in combination with data variants .
Implicit merging of functions	Implicit merging of subsystems with identical function names is not supported. Using the same function for a subsystem that has one instance with a global specification, and one or more instances with an external function specification, might result in incorrect code or code generation problems. This also applies to subsystems all of whose function specifications are external. To merge subsystems whose function specifications are all external, you can use the EnableExternFunctionMerge Code Generator option. For details, refer to Basics

on Merging External Functions ( TargetLink Customization and Optimization Guide).

Code differences in reused instances

Some settings lead to different code for the different instances, for example, scaling, operations, data types, size of variables, or names of the variables/functions, and TargetLink generates an error message. However, if two instances in different code generation units (TargetLink subsystem, incremental subsystem) differ, no error message is displayed. This might also happen if differing values for certain Code Generator options are used for code generation units.

For example, different Code Generator option settings can lead to different type specifications for the reused instances. Another more detailed example is that the code of reused functions can differ if the function provides outputs that are terminated and whose Optimization property is set to ERASABLE.

Note

The code generated for reuse structures can vary with different TargetLink versions and might not be compatible across versions.

There are two ways to avoid problems:

- Use the same TargetLink version to generate code for models with reused functions.
- Use a separate version of the reused library subsystem for each TargetLink version (e.g., with different subsystem and function names).

Reuse of custom code

Custom code variables cannot become part of reuse structures. For implicit reuse, they become shared variables, and a warning is issued only for state variables. If they are explicitly defined to be reused, an error message is displayed during code generation.

Including external code

TargetLink does not support external code in the S-functions of Custom Code (Type I) blocks if the code is included via external DD modules. The build process aborts. You might use an Addfile block instead.

Building S-functions that link external code files is only possible when these files are linked via an Addfile block that is placed on the same level as the Custom Code block. Therefore, linking the external files via DD FileSpecification objects is not supported for the S-functions.


Width macros in custom code

TargetLink does not support width macros in Custom Code (type I) blocks, only in Custom Code (type II) blocks.

No indirect reuse of 2-D matrices

If instance-specific entries represent 2-D matrices, they cannot be implemented as pointers in reuse structures.

Incremental code generation	<p>The following restrictions apply to incrementally generated subsystems and referenced models:</p> <ul style="list-style-type: none"> ▪ Function calls into incrementally generated subsystems or referenced models are not possible.
Custom look-up table functions	<p>The following features are not supported when you create variables for look-up table data:</p> <ul style="list-style-type: none"> ▪ Arrays with more than two dimensions ▪ More than two levels of nested structs ▪ Vectors of pointers ▪ Vectors of structures ▪ Pointers to pointers ▪ Initializations of pointers with other than the first elements of nonscalar variables, for example: <pre>int b[2]; int *a = &b[1]; /* unsupported */ but int *a = &b[0] /* supported */</pre> ▪ Unions ▪ Structures as formal parameters of functions (pointers to structures are supported)
Sorting of dynamic components	<p>Dynamic components can be sorted by different criteria such as type, size, and name. However, when the sort algorithm calculates the size of a pointer component, it does not take into account the specification as NEAR or FAR of the object the pointer points to.</p>
Offsets	<p>In some cases the combination of an offset and a very small value for the least significant bit (LSB) can lead to a constant that is wider than 32 bit. These constants cannot be handled properly by the Code Generator and should be avoided. This problem tends to occur when 32-bit block outputs are involved.</p>
Identifier of structs	<p>Both structure variables and their components defined in the TargetLink Data Dictionary must be specified by different identifiers, so that the resulting struct variable/component name in the code is unique. This is independent of the variable's scope.</p> <p>The identifier of a struct must be globally unique. The identifier of a struct component must be locally unique, i.e., it must be unique in the struct. The identifiers of structs that are implicit and themselves struct components must be globally unique.</p>

Union types	Union types are not supported by TargetLink, although they can be defined in the TargetLink Data Dictionary.
Page switching	<ul style="list-style-type: none"> ▪ The maximum number of generated data pages is 256. ▪ Structs generated by a Discrete Filter, Discrete State-Space, or Discrete Transfer Fcn block cannot be moved to data pages.
Requirement information of reused systems	<p>If subsystems or charts are reused multiple times, you can link every instance of them to different requirements. TargetLink includes the requirement information in the function header comment of the generated function. However, only the requirements of those instances that are part of the current code generation unit are included. Requirement information of all other instances are dropped without displaying a warning or note.</p> <p>To avoid the loss of requirement information of a reused system, ensure that all instances of the reused system are linked to the same requirements.</p>
Validity of compiler options	TargetLink does not check whether the compiler options specified by the user are supported by the host or target compiler.
No RDI objects in table maps	TargetLink does not support ReplaceableDataItem (RDI) objects in table maps. For details on RDI objects, refer to Basics on Component-Based Development Using Abstract Interfaces (RDI Objects) ( TargetLink Customization and Optimization Guide).
Compute through overflow and 64 bit	When using 64 bit operands with the ExploitComputeThroughOverflow Code Generator option set to never , TargetLink cannot avoid overflows in production code in all cases.
No code generation for matrices of dynamic size	Only 2-D matrices whose size is fixed during code generation time are supported. TargetLink does not support 2-D matrices whose size changes at compile time or run time.
Function calls to reused systems from outside the TargetLink subsystem are not supported	Reused systems cannot be called via a function call from outside the TargetLink subsystem in which they reside.

Saturation of pointer-to-struct components

If a pointer-to-struct DD variable meets both of the following conditions:

- the Scope property of its variable class is set to `ref_param`,
- a leaf component is referenced in the model multiple times,

all the references of the leaf component must have the same saturation setting, that is, the **Saturate** checkbox in the block dialog. Otherwise, code generation terminates.

Workaround Do not specify saturation in blocks referencing leaf components of pointer-to-struct DD variables. Instead, place **Rescaler** blocks directly after these blocks, and specify saturation there.

State reset in function-call triggered subsystems

If iterator subsystems meet the following condition:

- The iterator subsystem itself or one of its child subsystems calls a Function-Call subsystem by a Function-Call Generator block or a Stateflow chart with a function-call output event
- The For/While iterator has its **States when starting** property set to **reset**

TargetLink then resets the states of the called subsystems, when resetting the states of the iterator subsystem. This is a different to Simulink, because Simulink does not reset the states of the called subsystems in this situation.

Related topics

Basics

[Decomposing Models Logically \(Incremental Code Generation\)](#) (📖 TargetLink Customization and Optimization Guide)

[Managing Variant Coding and Conditional Code Compilation](#) (📖 TargetLink Customization and Optimization Guide)

[Reusing C Functions and Variables for Identical Model Parts \(Function Reuse\)](#) (📖 TargetLink Customization and Optimization Guide)

References

[ConsiderMergeableForBlockDiagramBasedSwitchOptimization](#) (📖 TargetLink Model Element Reference)

[EnableExternFunctionMerge](#) (📖 TargetLink Model Element Reference)

[ExploitComputeThroughOverflow](#) (📖 TargetLink Model Element Reference)

[Function Block](#) (📖 TargetLink Model Element Reference)

Component-Based Development Limitations

Introduction

The following limitations apply when you work with abstract interfaces or with referenced models:

Note

TargetLink treats referenced models as subsystems configured for incremental code generation, i.e., the same TargetLink limitations apply as for the incremental code generation (refer to [Code Generation Limitations](#) on page 40).

Export of code files

TargetLink does not support the export of code files from different code generation units using one unique file name. A just recently generated code file overwrites an existing one and TargetLink displays W03522 to inform you about the overwrite.

Generation of INIT variables

TargetLink cannot identify different initialization values, which is the precondition for generating INIT variables, across different code generation units but only within the same code generation unit.

Changes to Global Data Stores after Code Generation

TargetLink does not detect all changes made to DD objects that are associated to global data stores after code was successfully generated.

If you change DD objects associated to a global data store after code was successfully generated, you must generate code again for all the code generation units below the subsystem that contains the data store.

Variant subsystems

TargetLink does not support the Specify output when source is unconnected property of Simulink Outport blocks in Variant Subsystems.

Referenced models

TargetLink does not support the following properties of the Simulink Model block:

- ShowModelInitializePort
- ShowModelTerminatePort
- ShowModelPeriodicEventPorts
- Argument for InstanceParameters

TargetLink does not support default values for model arguments.

Sample time of reused subsystems	<p>For the sample times of reused and reusing systems, all of the following conditions must be met:</p> <ul style="list-style-type: none"> ▪ The sample times in a reused system instance must not differ. ▪ If at least one instance of a reused system A is not part of another reused system instance, one of the following conditions must be fulfilled: <ul style="list-style-type: none"> ▪ All the instances of system A have sample times that <i>equal</i> the sample time of their reusing system. ▪ All the instances of system A have sample times that <i>differ</i> from the sample time of their reusing system. ▪ If a reused system contains a TargetLink Discrete-Time Integrator block, all its instances must have the same sample time.
Read-only model workspace	<p>Due to the Simulink autosave, it can happen that model initialization in combination with referenced models aborts when TargetLink tries to access a read-only model workspace. To solve this problem, you must deactivate the Save before updating and simulating the model Simulink option.</p>
ReplaceableDataItem objects	<p>For the RDI objects themselves, TargetLink does not support struct types, struct components, bitfields, or pointers. If you assign one of these data types, an error message is displayed during code generation.</p>
DataItemMapping objects	<p>RDI objects cannot be mapped (via the VariableRef property of the DiM object) to a variable of struct or pointer type, but to global variables or components of global structures.</p>
Mask workspace variables of converted atomic subsystems	<p>For making a subsystem configured for incremental code generation a referenced model the mask workspace variables of the atomic subsystem are ignored, and vice versa.</p>
Name macros in function interfaces	<p>The \$M, \$N and \$B name macros in the interface of a referenced model evaluate into strings different from those of not-referenced systems. Therefore, the \$M, \$N and \$B name macros must not be used in the interface of referenced systems.</p> <p>The interface of a referenced system also includes:</p> <ul style="list-style-type: none"> ▪ Step, Init, Restart, and Term function ▪ The input and output variables (\$B is allowed) and the file names in which the Step, Init, Restart, and Term functions are generated
Autoscaling	<p>Suppose the Global logging option is set to Log min/max-values. For blocks that belong to a referenced model, no simulated ranges are available for further processing when the referencing model is simulated. In consequence,</p>

autoscaling the referenced model based on simulated ranges does not work.
One possible workaround is:

1. Disable the model reference.
2. Simulate the root model, and autoscale the formerly referenced model.
3. Restore the model reference.

Overflow detection

Overflow detection is supported only for referenced models in MIL mode if the signal history is logged.

Write-protected model files

[🔗 MIL simulation](#) with referenced models fails if the directory that contains the model file is write-protected.

Simulink's Run button

If you use Simulink's Run button on the model toolbar to start a simulation in MIL mode, signals in the referenced models are not logged. To ensure that signals inside referenced models are logged, use the `tl_sim` API function, or the Start simulation button of one of the following TargetLink windows:

- TargetLink Main Dialog
- TargetLink Plot Overview Window
- TargetLink Detailed Plot Window
- TargetLink Signal Comparison Window

Standard code generation mode only

RTOS code generation is supported only for the TargetLink subsystem but not for incremental code generation units for which code is always generated in the Standard code generation mode, if RTOS code generation is used for the TargetLink Subsystem. Hence, do not use blocks from the TargetLink RTOS library in referenced models and subsystems configured for incremental code generation.

AUTOSAR

Modeling is restricted in [🔗 model components](#) that contain functional subparts of runnables and are placed in [🔗 runnable subsystems](#) or [🔗 operation call with runnable implementation subsystems](#). Only the following modeling styles for AUTOSAR communication are possible:

- Modeling client-server communication via [🔗 operation subsystems](#).
- Modeling sender-receiver, data element updated, NvData, and IRV communication via data store blocks.
- Modeling PIMs, SharedCalPrms and PerInstanceCalPrms via predefined variable classes and access functions.

Converting referenced models into incremental subsystems

TargetLink cannot convert a referenced model into an incremental subsystem, if variant conditions are specified for it.

Modeling operation call and operation result provider subsystems

Modeling AUTOSAR operation calls or operation result providers via a subsystem is not supported for subsystems that are configured as incremental code generation units.

Data Dictionary Limitations

Introduction

When working with the TargetLink Data Dictionary, you must take certain limitations into account:

MATLAB import/export

TargetLink lets you synchronize MATLAB variable objects with the value properties of exported DD variable objects, and in the other direction, the value properties of existing DD variable objects with the values of imported MATLAB variable objects. However, MATLAB structures are not clearly defined, i.e., the value property is not clear without additional user input. Therefore, MATLAB structures are not supported for value synchronization.

Code generation Validation Limitation

The DD validation validates all DD objects before generating code, even if the DD object meets one or more of the following criteria:

- Their ExcludeFromCodeGeneration property is set to on.
- They are not used in any model or DD-based code generation.

Therefore, the complete Config and Pool area of the Data Dictionary must validate without errors before code generation can be started.

Stand-alone mode

Some utilities that are called from the TargetLink Data Dictionary Manager are written as MATLAB M scripts. The following utilities cannot be used with the TargetLink Data Dictionary Manager in stand-alone mode:

- Opening the code file associated with a DD object.
- Showing the block associated with a DD object in a Simulink model.
- Creating a custom output view.
- Creating an HTML Comparison report.
- Menu Extensions specified via XML files.
- The following commands from the Tools menu:
 - Manage Build
 - Import MATLAB Variable Objects
 - Export DD Variable Objects
- Some AUTOSAR functionalities.

Custom command objects and custom command group objects cannot be executed by the Data Dictionary Manager if the Data Dictionary Manager was started in stand-alone mode.

No downgrading of Data Dictionaries	<ul style="list-style-type: none"> ▪ The Data Dictionary APIs do not support downgrading Data Dictionaries to older Data Model versions. For example, you are working with the newest TargetLink version and want to downgrade a Data Dictionary to use it in an older TargetLink version that uses an older Data Model. ▪ In addition, the DD APIs do not support downgrading of DDs of newer TargetLink versions to the current Data Model version. For example, you are working with an older TargetLink version and want to downgrade a Data Dictionary from a newer TargetLink version that uses a newer Data Model.
“default” name	The <i>default</i> string is a keyword in TargetLink that instructs the Code Generator to choose the best implementation according to internal rules. You must not choose <i>default</i> as a name for DD objects.
Number of variants	The number of variants for a property is limited to 255.
GetAll command	When the <code>dsdd('GetAll',...)</code> command is used to read all the properties of a DD object, special characters in property names are replaced by underscores to get valid MATLAB struct field names. If this happens, the properties cannot be restored with <code>dsdd('SetAll',...)</code> .
Non-scalar values of type Int64 and UInt64	The Data Dictionary APIs do not support non-scalar values of the Int64 or UInt64 type.
64-bit integers	<p>The following TargetLink Data Dictionary Import and Export modules do not support 64-bit integer data types:</p> <ul style="list-style-type: none"> ▪ OIL ▪ A2L
Product-specific limitations	The Data Dictionary supports specification, import and export of objects, and settings that are ignored or not supported by the TargetLink code generator. For example, although the Data Dictionary can represent UNION types, the production code generator does not support them. It is therefore recommended to also check the code generator-specific limitations.
No central database server	The Data Dictionary is not a central database server. It can be accessed from only one process. For example, if MATLAB and another application are open at the same time, they both work with independent instances of the Data Dictionary. Any changes in the Data Dictionary workspace in MATLAB are propagated to the other application only if they are saved in the DD project file and loaded by the other application.

Different DD project files	Multiple DD project files can be open at a time but only the file that is loaded to DD0 can be used with the model and for Code Generation. You cannot open two Simulink models and work with two different DD project files simultaneously.
File I/O	<p>When DD files are loaded into the current Data Dictionary workspace, objects and properties might be overwritten without notification. To make sure that valuable data is not lost, save the current Data Dictionary to file before loading a file.</p> <p>DD files represent a DD object tree/subtree. The object kind of the root object of a file must match the object kind of the object the file is to be loaded to.</p>
DD objects without read access right	If an administrator removes users' read access rights to a DD object, i.e., <code>Access="rw--"</code> or <code>"r---"</code> , users cannot get and set the object properties. Although read access is denied, the object is still visible in the DD object tree in the Data Dictionary Manager. The existence of the object can be checked with DD API functions, such as <code>dsdd('Exist',...)</code> , <code>dsdd('GetChildren',...)</code> , <code>dsdd('GetAttribute')</code> , etc.
Admin password for DD files	<p>If the current DD project file is not password-protected and another password-protected DD project file is loaded into the current DD workspace, this password is inherited by the current DD project file. This means a user cannot get access rights for protected objects from the loaded file without knowing the admin password.</p> <div data-bbox="592 1190 659 1222" data-label="Section-Header">Note</div> <div data-bbox="598 1245 1374 1341" data-label="Text"> <p>Without knowing the admin password, you cannot delete write-protected DD objects. The only way to delete them is to clear the entire data dictionary.</p> </div>
Data Dictionary Manager started via MATLAB	When opened via the MATLAB Command Window, the windows and messageboxes of the Data Dictionary Manager, such as the Save Data Dictionary or Reference selection dialog box, sometimes open in the background, and you have to activate them.
Points of inclusion	The Config, the / Config / DDIncludeFiles, and the Pool objects cannot be made points of inclusion. Files whose file root object is a Config, Pool, or DD root object cannot be included.

MATLAB does not react after Ctrl+C

Do not use Ctrl+C to copy, e.g., DD objects in the Data Dictionary Manager running under MATLAB. This may interrupt scripts running in background. For details, refer to MathWorks® support.

Support of 8 bit encodings in the Data Dictionary

The Data Dictionary supports only 8 bit encodings for property names, DD object names and DD paths (Multibyte Character Set/UTF-8 is not supported). Characters that cannot be displayed in the local code page are lost during the conversion to other code pages. It is recommended to use ASCII characters only.

Documentation Generation Limitations

Introduction

The following limitations apply to the generation of documents.

HTML-to-PDF conversion

If HTML content was inserted into the generated documentation by means of the Autodoc Customization block, the HTML-to-PDF conversion might fail or the inserted part might not be displayed correctly in the PDF documentation if the HTML uses features that are not supported by the PDF generator.

Missing SVG images

If you use Internet Explorer, the generated HTML documentation does not display images of SVG format if the following two conditions are fulfilled at the same time:

- The generated documentation is stored on a network drive
- The Internet Explorer's compatibility view is enabled. Refer to the Tools - Compatibility View settings - Display Intranet sites in Compatibility View option.

Related topics**Basics**

Generating Documentation on Model Characteristics ( [TargetLink Interoperation and Exchange Guide](#))

FMU Generation Limitations

Introduction

The following limitations apply to FMU generation.

No TOM support

TargetLink cannot generate FMUs for TargetLink subsystems whose code contains target-specific optimizations (TOM).

No RTOS support

TargetLink cannot generate FMUs from TargetLink subsystems whose production code was generated in RTOS mode.

No asynchronously triggered root functions

TargetLink cannot generate FMUs for TargetLink subsystems that are triggered asynchronously.


No support for matrix interface variables

TargetLink cannot create FMUs for TargetLink subsystems that require matrix variables at their interface.

Only one step function

TargetLink cannot create FMUs for TargetLink subsystems that have more than one root step function.


A TargetLink subsystem can have more than one root step function if it contains one or more externally triggered function call subsystems.

Downsampling as described in [Generic Multirate Code Generation](#) ( [TargetLink Preparation and Simulation Guide](#)) is supported.

Related topics

Basics

[Basics on Exporting FMUs from TargetLink](#) ( [TargetLink Interoperation and Exchange Guide](#))


[Definition of the FMI Standard and FMUs](#) ( [TargetLink Interoperation and Exchange Guide](#))

HowTos

[How to Generate an FMU to use in an FMI-Compliant Tool](#) ( [TargetLink Interoperation and Exchange Guide](#))

References

[TargetLink FMU Manager](#) ( [TargetLink Tool and Utility Reference](#))

[tl_generate_fmu](#) ( [TargetLink API Reference](#))

Generic Multirate Limitations

Introduction	The following limitations refer to multirate code generation in the Standard and AUTOSAR code generation modes.
Enabled subsystems	<p>TargetLink does not support the following modeling, otherwise differences between the MIL simulation and SIL simulation/PIL simulations occur:</p> <ul style="list-style-type: none"> ▪ If a code generation unit (CGU) is enabled, it must not contain embedded systems with sample times different from the sample time of the code generation unit. ▪ If a code generation unit is part of an enabled subsystem, it must not contain embedded systems with sample times different from the sample time of the code generation unit. ▪ If a Runnable, an Adaptive AUTOSAR function, or a Method Behavior subsystem is enabled or is part of an enabled subsystem, it must not contain embedded systems with sample times different from the sample time of the code generation unit.

Processing of Non-Scalar Signal Limitations

Introduction	The following limitations apply to the processing of non-scalar signals.
Blocks not supporting loop generation	<p>The following blocks never support the generation of loops for vector/matrix signals:</p> <ul style="list-style-type: none"> ▪ Custom Code block (except for the updates of the input variables) Reason: TargetLink never changes custom code which is associated with a Custom Code block. ▪ Fcn block Reason: You can use vector/matrix signals in the mathematical expression for the block. However, TargetLink does not analyze or change the expression.
No Direct Memory Access logging for matrix variables	TargetLink does not support Direct Memory Access logging for matrix variables that have different scalings.
Blocks with limitations under specific conditions	<p>No loops are generated for vector/matrix signals using the <i>Sum block</i>, if all of the following conditions are met:</p> <ul style="list-style-type: none"> ▪ The block has exactly one inport. ▪ The data type of the output is integer.

- And one of the following conditions:
 - Saturation for the block's output to the lower and/or upper limits of the implemented range is enabled.
 - or –
 - The scaling of the vector/matrix signals at the inport is more precise than the scaling of the block's outport.

No loops are generated for vector/matrix signals using the *Product block*, if all of the following conditions are met:


- The block has exactly one inport.
- The data type of the outport is integer.

The *Trigger block*, *D Flip-Flop block* and *Stateflow charts* have limitations with regard to processing their trigger logics: Loops are generated for vectorized signals at the block's trigger inputs (at the D Flip-Flop block it is the CLK input) only if these signals are well-defined. Well-defined means that the signals are not multiplexed and that they have equal scalings for each element. No slice identification is performed for the trigger logic code.

Limitations for blocks with vectorized block parameters

For blocks with vector/matrix block parameters, loop generation is only possible if:

- All vector/matrix elements are constants and equal, or
- You have specified a variable class for the block parameters that is unequal to default to force TargetLink to create a new variable for them.

In this case, loop generation is also performed according to the principles of slice identification as described in [Basics on Processing Vectors and Matrices](#) ( [TargetLink Preparation and Simulation Guide](#)). For example, all vector/matrix elements that are processed in a loop must have the same characteristics (same data types, uniform elements, same user-defined range limits and same saturation settings).

Tip

Keep in mind that introducing variables can influence stack consumption, RAM/ROM consumption and run-time performance.

Merging consecutive loops

TargetLink can generate **for**, **while**, and **do...while** loops, but only consecutive **for** loops are merged if possible. Consecutive **while** and **do...while** loops as well as any kind of mixed consecutive loops are left untouched.

RTOS Multirate Limitations

Introduction

The following limitations refer to the RTOS [code generation mode](#).

Counter Alarm Block

The following limitations apply to the Counter Alarm block

- The CounterValue output must be connected to a Terminator block.
- The CounterTrigger input can be triggered only from outside the TargetLink subsystem.

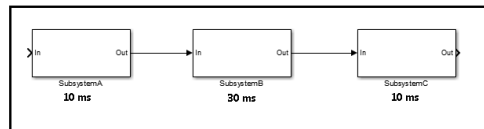
Standard code generation mode only

RTOS code generation is supported only for the TargetLink subsystem but not for incremental code generation units for which code is always generated in the Standard code generation mode, if RTOS code generation is used for the TargetLink Subsystem. Hence, do not use blocks from the TargetLink RTOS library in referenced models and subsystems configured for incremental code generation.

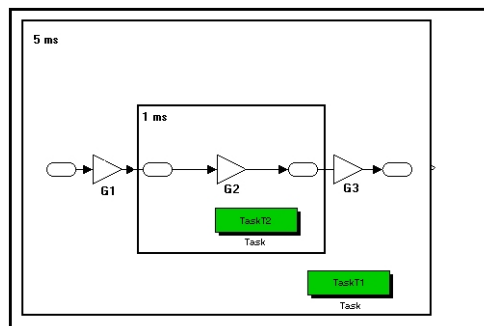
Interrupted signal flow in specific subsystems

If awkward subsystem sample times are chosen, the data flow between the subsystems may be interrupted and ignored by TargetLink.

Consider the following example models. In both models, the data flow is interrupted due to the subsystem's sample times.



In the example model below, the data flow changes depending on the chosen simulation mode.



If you start a Simulink simulation, the calculation order of the Gain blocks is G1, G2, G3. In TargetLink [MIL simulation mode](#), there is the same execution order. In the generated code, G1 and G3 reside in task T1, G2 resides in task T2. Depending on the task priorities, T1 is executed before T2 or vice versa. If T1 is executed before T2, the execution order of the blocks is G1, G3, G2. If T2 is

executed before T1, the execution order is G2, G1, G3. Both variants differ from the Simulink and MIL simulation modes.

Same sample time for real blocks in a subsystem

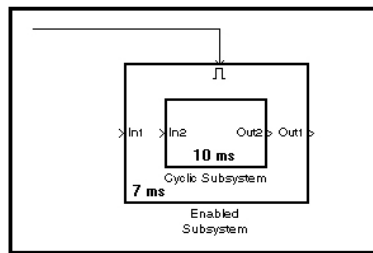
In Simulink you can mix real blocks with different sample times in the same atomic subsystem. This is not possible in TargetLink.

No reset for action-port-triggered subsystem

The states and outputs of an action-port-triggered subsystem cannot be reset if the subsystem becomes a separate task.

Reset outputs when disabled in nested subsystems

Consider the following model:



For both subsystems, the Reset outputs when disabled option is set to ON.

Simulink behavior: The reset is performed with the greatest common divisor of the involved sample times.

TargetLink behavior: The reset is performed with the sample time of the embedded cyclic subsystem.

This also applies in the following cases:

- The embedded subsystem is an enabled subsystem.
- The sample time of an enabled subsystem differs from the sample time of its enabling signal.

Multiple edge-triggered subsystems behavior

If the two trigger sources of an edge-triggered subsystem reside in different tasks, it is not always possible to map Simulink behavior in the production code precisely.

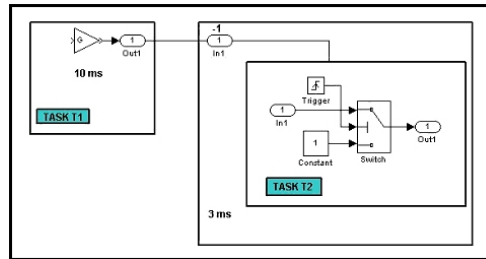
Simulink behavior: If the two trigger signals are activated at the same time step, the triggered subsystem is calculated once.

TargetLink behavior: If the two trigger signals are activated at the same time step, the triggered subsystem is calculated twice.

Show output port for Trigger blocks is not allowed in special cases

The Show output port option is not allowed at trigger blocks if the trigger signal crosses task borders. The signal at the trigger inport is a standard data input of the [root step function](#) containing the triggered subsystem. This signal cannot be used in the triggered subsystem.

Consider the following example model with tasks T1 and T2. The subsystem surrounding T2 is a separate task in this model:

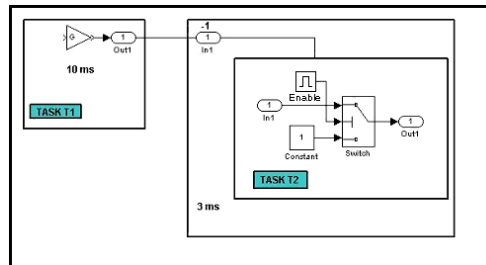


You cannot specify that the trigger flag used in T2 must be consistent with the other data transferred from T1 to T2.

Show output port for Enable blocks is not allowed in special cases

The Show output port option is not allowed at enable blocks if the enable signal crosses task borders. The signal at the enable inport is a standard data input of the root step function containing the triggered subsystem. This signal cannot be used in the enabled subsystem.

Consider the following example model with tasks T1 and T2. The subsystem surrounding T2 is a separate task in this model:



You cannot specify that the enable flag used in T2 must be consistent with the other data transferred from T1 to T2.

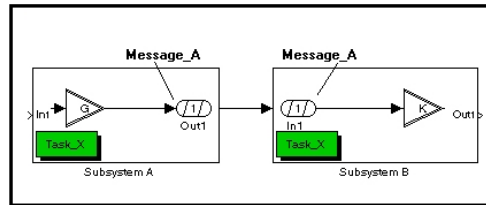
No ITC within a root step function

You cannot specify [intertask communication](#) for subsystems belonging to the same root step function.

The alignment of the code receiving or sending a message is fixed for each root step function. The code is aligned according to the following rules:

- All operations that receive a message are placed at the beginning of a root step function.
- The application code generated from the subsystems and blocks is placed in the center of the root step function.
- All operations sending a message are placed at the end of the root step function.

Consider the following example model:



In this example, the bounds of subsystem A and subsystem B are root step function borders.

Due to the above limitation, the signal flow between subsystem A and subsystem B is not implemented in the right order. This leads to several delays.

Linked resources

You cannot specify linked resources in your TargetLink model. However, you can store them in the LinkedResourceRef property in the **Pool1/RTOS/Resources** subnode in the TargetLink Data Dictionary so that you can import and export OIL files. For more information on linked resources, refer to the OSEK OS specification.

Unit Delay block used for sampling rate transition

If a Unit Delay block is used for sampling rate transition, you cannot switch logging on for it. Additionally, you must not specify Saturation.

Enable and Trigger blocks on the root level of multirate models

In the RTOS code generation mode, Enable and Trigger blocks are not permissible at the root level of your TargetLink model. As a workaround, copy a subsystem to the root level TargetLink subsystem and handle it as the root level TargetLink subsystem. All input signals can be specified. If a root level TargetLink subsystem is empty or contains no real blocks, TargetLink does not generate a function from it. Therefore, no superfluous functions are generated.

Simulink fixed-step solver with multitasking mode

The Simulink fixed-step solver with multitasking mode issues an error message if it detects an illegal sampling rate transition between blocks, that is, a direct connection between blocks operating at different sample rates. For [SIL simulation](#)/[PIL simulation](#), the complete TargetLink subsystem is replaced by an S-function to integrate the generated code into the Simulink simulation. In MIL simulation, all the blocks like Gain, Sum and Unit Delay remain in the TargetLink subsystem and are used for simulation.

The following example illustrates the problem:

The TargetLink subsystem contains two atomic subsystems with different sample times, 90 ms and 100 ms. The 100-ms subsystem receives data from a 50-ms subsystem outside the TargetLink subsystem. If the fixed-step solver and the multitasking mode are selected, Simulink forces you to place a Zero Order Hold block with a sample time of 100 ms between the 50-ms and the 100-ms subsystem in MIL simulation. In SIL and PIL simulation, the 90-ms and 100-ms

subsystems are replaced by the S-function. This S-function is executed cyclically with the greatest common divisor of 90 ms and 100 ms = 10 ms. In this case, Simulink expects a Unit Delay block with 50 ms instead of the Zero Order Hold block with 100 ms between the 50-ms and the 100-ms subsystem.

To avoid the problem use a Rate Transition block instead of a Zero Order Hold/Unit Delay block between blocks operating at different sample rates.

Unit Delay block to simulate real-time behavior at rate transitions

Simulink always assumes rate monotonic scheduling. This means faster tasks have higher priorities. A Unit Delay block between the slow and the fast task simulates the delay caused by the lower priority of the slow task. Simulink uses a delay of the slow rate, which is the worst-case assumption. This is also the result in MIL simulation mode. The TargetLink code executed in SIL and PIL mode does not include these worst-case delays. The rate monotonic scheduling now results in a delay with only the fast sampling rate. The result is that MIL and SIL/PIL can differ if a Unit Delay block is used to simulate such real-time effects.

No message called default

It is not allowed to create a message called default, because a default entry already exists in the messages list of the TargetLink InPort and OutPort block.

Execution time and stack consumption of multirate models

If you perform simulations in the RTOS code generation mode, both the execution time and stack consumption measured include the time and stack required by the OSEK emulator. As a result, the time and stack consumption measurements are higher than actually required.

\$S name macro

The \$S name macro is neither allowed for task names nor for the names of root step functions.

Sample time specification

A large sample time specification might lead to an error message reporting downsampling problems.

No macros for ISR

TargetLink does not support interrupt service routines specified as macros, e.g., with their function class set to GLOBAL_MACRO.

2-D signals

For RTOS code generation mode, 2-D signals are not allowed to pass task boundaries. However, in tasks 2-D signals are supported.

No resettable subsystems

For RTOS code generation mode, TargetLink does not support resettable subsystems in the TargetLink subsystem.

Function-call-triggered subsystems

For function-call-triggered subsystems in RTOS code generation mode, TargetLink only supports the States when enabling parameter set to Inherit.

Stateflow Limitations

Introduction

TargetLink supports the MathWorks® Stateflow Toolbox. However, a few conventions are different to what is supported in Simulink.

Pass-by-reference of Stateflow functions

TargetLink does not support the pass-by-reference semantics of Stateflow functions (graphical functions, truth table functions, MATLAB functions). Pass-by-reference semantics use the same data as input and output, which is modeled by giving identical names to input and output data.

This Stateflow feature was introduced with MATLAB R2018b.

String functions

The TargetLink Code Generator does not support string functions in Stateflow Charts.

TargetLink/Stateflow differences

If an operation of variables with a Stateflow and TargetLink integer type is assigned to a variable with Stateflow type double and TargetLink type Float32 or Integer (LSB≠1 or Offset≠0), TargetLink and Stateflow return different results. Solution: Assign the result to an integer variable.

TargetLink and Stateflow perform division operations differently. This can lead to differences in MIL/SIL simulations.

TargetLink does not apply saturation to auxiliary results as Stateflow does via the SaturateOnIntegerOverflow chart property.

C-based Stateflow derives the data type of a bitwise operation from the data types of the operands. Without scaling, TargetLink also uses information from the context of the bitwise operation to determine the data type.

This can lead to differences between MIL and SIL.

C-based Stateflow derives the data type of a shift operation from the data type of the left operand. Without scaling, TargetLink also uses information from the context of the shift operation to determine the data type.

This can lead to differences between MIL and SIL.

Stateflow messages

TargetLink does not support Stateflow messages. Stateflow messages are Stateflow objects that carry data and can be queued. They became available as of Stateflow R2015b.

rand function is not supported	TargetLink does not support the rand function in Stateflow charts' action language.
Simulink functions are not supported	TargetLink does not support Simulink functions defined in Stateflow charts.
Unicode	TargetLink supports Unicode in Stateflow charts only for descriptions and annotations.
General Stateflow limitations	<ul style="list-style-type: none"> ▪ Implicit data change events are not supported if the data is defined at the level of the Stateflow machine. ▪ Elements of Stateflow variables (for example, vectors, matrices) must have the uniform elements for each element. ▪ Complex Stateflow data is not supported. In consequence, the complex, imag and real functions are not supported. <p>Setting the complexity to auto or inherited is supported only if the complexity can be determined.</p>
Calls to Stateflow functions with side-effects	TargetLink always calls functions first before evaluating the rest of the expression containing the call. Therefore, if a function has a side effect, it affects the evaluation of the expression containing the call in the Stateflow action language. This can lead to differences between MIL and SIL/PIL simulations. No warning or error occurs.
Action language and name resolution	TargetLink does not support the old, i.e., R2008a and earlier, name resolution scheme if there is more than one match in the chart.
Using State Variables as State Flags	<p>Only integer data types are allowed. In addition, mandatory settings are LSB=1 and offset=0.</p> <p>The following limitations apply when working with active state data:</p> <ul style="list-style-type: none"> ▪ TargetLink does not support the Leaf State Activity state activity mode. ▪ TargetLink does not support non-scalar data elements that are connected with a state (active state data). ▪ TargetLink supports only active state data with the initial value of 0.
Customer integer types are increased to 8-, 16- or 32-bit length	Fixed-point data types specified in MATLAB/Simulink with customer bit length (for example 5 bit) are increased to the next possible length. If this semantic is used, a warning occurs.

Stateflow data logging limitations	<p>Stateflow test points Because Stateflow test points allow only one signal value to be stored during a simulation step, it is not possible to determine how often a Stateflow data object value was changed or assigned during a simulation step in MIL simulation mode. This could be never, once, or many times.</p> <p>As a consequence, the TargetLink logging mechanism logs only the last value of the Stateflow data object during each simulation step. This affects the graphical representation of the signal histories and can easily lead to visual differences between logging during MIL simulation and that in both SIL and PIL simulation.</p> <p>Loggable objects Only Stateflow Data Local and Data Output objects can be logged during MIL simulation.</p> <p>Stateflow data objects MIL logging of data objects is possible only for data</p> <ul style="list-style-type: none"> ▪ with Local scope if the data objects reside on chart level or below ▪ with Output scope if the data objects reside on chart level (meaning that outputs of Stateflow functions cannot be logged during MIL simulation) <p>You cannot perform data logging of Stateflow Data objects if you use a const or macro variable class.</p>
External source code	Literals (code between \$ characters) are not supported in Stateflow.
Embedding MATLAB/Simulink functionality	<p>TargetLink does not support:</p> <ul style="list-style-type: none"> ▪ The use of MATLAB functions (see <i>ml()</i> Functions for details). ▪ The use of MATLAB actions (see <i>MATLAB Name Space Operator</i> in the <i>Stateflow User's Guide</i> for details). ▪ The use of the time symbol <i>t</i>, which represents the absolute simulation time inherited from Simulink (see <i>Time Symbol</i> in the <i>Stateflow User's Guide</i> for details).
Tan function in Stateflow actions	<p>In the generated production code, the tan function can be called with a float data type. If you want to use fixed-point or integer data types, use an assignment of the function call to a fixed-point variable:</p> <pre>fixedPtVariable = tan(...)</pre>
Early Return Logic semantic	<p>To achieve higher production code efficiency, TargetLink does not support the rarely used modeling technique described below:</p> <p>The Early Return Logic after event broadcasts (see Semantics in the Stateflow User's Guide for details) is currently not supported. If a state <i>p</i> broadcasts an event <i>e</i>, TargetLink will always execute the remaining code of the state action after the broadcast, even if the state <i>p</i> is deactivated while event <i>e</i> is being processed. Do not use any event that might deactivate its sender during the broadcast. If necessary, you can introduce an additional state to control the activity of the sender state. In this case, the TargetLink semantics differ from Stateflow. You should not use this technique with TargetLink.</p>

Casts of data types

TargetLink does not support casts to the int64 and uint64 data types.

Reserved names

The keywords below are reserved by Stateflow and must not be used as names for data, events, states, or graphical functions:

abs	change	exp	log10	sqrt
acos	cos	exit	max	t
after	cosh	fabs	min	tan
asin	du	floor	matlab	tanh
at	during	fmod	ml	tc<digits>
atan	en	hasChanged	pow	trigger_id
atan2	enter	hasChangedFrom	send	
before	entry	hasChangedTo	sgn	
ceil	every	in	sin	
chg	ex	log	sinh	

If a Stateflow chart contains a graphical function that has exactly the same name as a valid change detection operator (*hasChanged*, *hasChangedFrom* or *hasChangedTo*), the graphical function's name will conceal the name of the change detection operator, and the graphical function will be used instead of the change detection operator. This will lead to simulation differences between MIL and SIL/PIL simulations.

TargetLink does not support change detection for structures and substructures. However, for leaf components change detection is supported.

TargetLink does not support change detection for data that is nested in states.

Temporal Logic

TargetLink does not support the Stateflow Temporal Logic operators *elapsed* and *duration*.

Exported Stateflow functions

Stateflow functions include graphical functions, truth tables, and MATLAB functions.

For an exported Stateflow function, you have to ensure correct simulation modes of the [TargetLink subsystem](#) containing the defining chart and the TargetLink subsystems containing the using charts.

Simulation fails (linker error) or simulation differences occur if the simulation modes of the subsystems are not identical.

Bind actions and function-call-triggered subsystems

When Bind actions as introduced by Stateflow 6.0 are used, the limitations concerning state and output reset described in [Subsystem Creation Limitations](#)

on page 71 also apply to function-call-triggered subsystems called by bound events. For details, refer to the following limitations:

- States in function-call-triggered subsystems
- State reset in conditionally executed subsystems
- Output reset in function-call-triggered subsystems

History junctions

Default transitions or default paths connected to history junctions are not supported.

Initialize Outputs Every Time Chart Wakes Up

TargetLink does not support the Initialize Outputs Every Time Chart Wakes Up chart property. Any setting of this chart property is ignored and TargetLink generates a warning message.

Update method

TargetLink does not support the Update method chart property set to Continuous. This setting is ignored and TargetLink generates a warning message.

Mealy and Moore

TargetLink does not support Mealy and Moore semantics and generates a warning message.

MATLAB data type

TargetLink does not support the MATLAB-specific data type `m1`.

Stateflow boxes data

Data can be defined in Stateflow box objects. TargetLink does not support data defined in box objects. Instead of defining data in Stateflow boxes, define them on a higher level, for example, in the surrounding Stateflow chart.

Calling a Stateflow function

You cannot call a Stateflow function, e.g., a graphical function or truth table, that has scalar input and output by using a vector argument. If you do call such a function, the following error message is displayed:

E20967 Scalar function cannot be called with vector argument.

Using temporal logic

The use of temporal logic in Stateflow is limited by TargetLink as follows:

- The use of absolute simulation time periods in temporal logic with the `sec`, `msec`, and `usec` keywords is not supported.
- The use of the `temporalCount` function is not supported, which returns the number of events occurred or the number of seconds elapsed since the activation of the state.
- Temporal logic conditions on transitions originating from junctions are not supported.

Using MATLAB action language in truth tables	<p>MATLAB provides an option that allows you to use the MATLAB action language in Stateflow truth tables. This option is not supported by TargetLink. If you use the MATLAB action language, the following error message will be displayed:</p> <p>E20885 Evaluating MATLAB expressions from within Stateflow diagrams is not supported.</p>
Unsupported charts	<p>Truth Table and State Transition Table TargetLink does not support Truth Table and State Transition Table blocks. If you want to use truth table or state transition table functions with TargetLink, include a Stateflow chart with a truth table or a state transition table instead.</p> <p>Atomic subcharts Atomic subcharts and atomic boxes are not supported.</p>
Accessing components of external Custom Structures	<p>TargetLink does not support dots and appended component names for external C variables in Stateflow (Custom Structures). For example, the <code>mystructvar.comp1</code> expression is not allowed. As a workaround, define macros to access struct components of external struct variables. For example, use <code>MYSTRUCTVAR_COMP1</code> in the chart, and add the following macro to the header file:</p> <pre>#define MYSTRUCTVAR_COMP1 mystructvar.comp1</pre>
Pointer operations	<p>TargetLink does not support the <code>-></code> pointer operation for Stateflow.</p>
Action language	<p>TargetLink does not support Stateflow charts whose action language is M. TargetLink supports only C.</p>
Related topics	<p>Basics</p> <p>Working with Stateflow (📖 TargetLink Preparation and Simulation Guide)</p> <p>References</p> <p>Subsystem Creation Limitations..... 71</p>

MATLAB Language Limitations

Introduction	The following limitations apply to the TargetLink support of MATLAB Language.
Varying variable types	TargetLink does not support different types for the same variable in a MATLAB code script.
Types of local variables	TargetLink does not support Struct types for local variables in MATLAB code.
64-bit integer literals	TargetLink does not support binary/hexadecimal 64-bit integer literals in MATLAB code.
Custom code in Stateflow MATLAB functions	TargetLink does not support custom code in Stateflow MATLAB functions. Custom code in Stateflow is supported.

Subsystem Creation Limitations

Introduction	The most important limitations that you have to take into account when creating a TargetLink subsystem are described below.
Subsystems called from extern	A function-call triggered subsystem that is called from outside the TargetLink subsystem might require additional update operations before and/or after its call. These update operations are not part of the production code but must be provided by the code of the call (TargetLink simulation frame or the calling environment of the target).
Outputs of TargetLink subsystems	For TargetLink subsystems, the outputs at model initialization time always emit the default values that are related to the data type, e.g., false for Boolean and 0 for numeric data types.
Inheritance of struct output variables	Suppose the following blocks make up a signal line: <ul style="list-style-type: none"> ▪ Block A: TargetLink block whose output is a struct type with Scope set to Local. ▪ Block B: Simulink Inport or Outport block. ▪ Block C: TargetLink block whose Inherit properties property is enabled.

Then, the struct type of block A is not inherited by block B but by block C, i.e., block B is skipped.

Logging of outputs of enabled subsystem with Output when disabled = reset

There are MIL/SIL differences in the signal plot for the output of an enabled or action triggered subsystem, if the **Output when disabled** property of the output is set to reset: During the time the enabled or action triggered subsystem is disabled, the plot of the MIL simulation shows the initial output value specified at the output. This is the external behavior of the enabled or action triggered subsystem. In contrast, the plot of the SIL simulation shows the last value of the output before the enabled or action triggered subsystem has been disabled. This is the internal behavior of the enabled or action triggered subsystem.

Models with event buses not initializable

Models containing a bus that consists only of event signals are not initializable and simulation in SIL mode is not possible if both of the following conditions are fulfilled:

- The bus crosses the boundary of the TargetLink subsystem on the input side.
- Inside the TargetLink subsystem, single bus signals are selected via a BusSelector block.

To avoid this problem, use the following workaround:

- Outside the TargetLink subsystem:
Convert the bus signal to a multiplexed signal by using Bus Selector and Mux blocks.
- Inside the TargetLink subsystem:
Select event signals by using Selector or Demux blocks (no Bus Selector block).

Merging external functions

TargetLink does not allow for external functions to be merged if one of the following conditions is met:


- At least one interface variable is a struct.
- At least one interface variable is a Pointer-To-Struct parameter.

Merging external functions is permitted only for **Step**, **Init**, **Term**, and **Restart** functions of subsystems that contain a Function block, and Charts.

For details, refer to [Basics on Merging External Functions](#) ( [TargetLink Customization and Optimization Guide](#)).

Compiled data types of output signals

If the signals that leave a TargetLink subsystem have Simulink fixed-point data types, the Simulink Fixed-Point software license is required during [SIL simulation](#)/[PIL simulation](#). [MIL simulation](#) and production code generation might work without the Simulink Fixed-Point software license.


Mixed signal bus inputs for atomic subsystems	If the input of an atomic subsystem is specified to become a function argument, the input signal is restricted to real signals. The signals must not be composed of data and event signals but of data signals only. This also applies to bus inputs.
Triggered and function-call subsystems	A triggered TargetLink subsystem can reside anywhere in the model. Function call connections, however, must not run from inside a TargetLink subsystem to a block outside the TargetLink subsystem. Bus signals that run into a TargetLink subsystem must not contain both data signals and function call signals.
Initial output handling	<p>The following applies only to the Classic initialization mode but not to the Simplified initialization mode:</p> <p>With conditionally executed subsystems, possible differences between the Simulink and TargetLink initial output handling might result in deviations of the SIL simulation from the MIL simulation. To avoid these differences, specify initial values at the Outport blocks of conditionally executed subsystems.</p> <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>If a TargetLink subsystem itself is triggered, the Simulink outport block dialog field that contains the initial value might be disabled or not shown. However, TargetLink evaluates this field.</p> </div>
State reset in conditionally executed subsystems	The TargetLink state reset behavior in conditionally executed subsystems differs from the Simulink state reset behavior in conditionally executed subsystems in a few special cases, refer to the limitations described below and also State Reset in Conditionally Executed Subsystems: Differences of Simulink and TargetLink ( TargetLink Customization and Optimization Guide)
State reset in TargetLink Subsystems	<p>TargetLink does not perform a state reset if both of the following two conditions apply to the TargetLink subsystem:</p> <ul style="list-style-type: none"> ▪ The TargetLink subsystem is an Enabled Subsystem or Function-Call-Triggered Subsystem whose States when enabling parameter set to Held. ▪ The TargetLink subsystem calls a Function-Call-Triggered Subsystem whose States when enabling parameter is set to Reset. <p>TargetLink does not perform a state reset in subsystem B if all the following three conditions apply:</p> <ul style="list-style-type: none"> ▪ The TargetLink subsystem is an Enabled Subsystem or Function-Call-Triggered Subsystem whose States when enabling parameter is set to Reset. ▪ The TargetLink subsystem calls an Enabled Subsystem or Function-Call-Triggered Subsystem (subsystem A) whose States when enabling parameter is set to Held.

- Subsystem A calls an Enabled Subsystem or Function-Call-Triggered Subsystem (subsystem B) whose States when enabling parameter is set to Reset.

Reset of actual parameters

TargetLink does not reset a state variable of a block that resides in a subsystem that has its state reset enabled if this state variable uses a variable class that has the Scope property set to ref_param.

Resettable subsystems configured for incremental code generation

TargetLink does not generate code for Resettable Subsystems that are configured for incremental code generation. For information on configuring subsystems for incremental code generation, refer to [Basics on Incremental Code Generation](#) ( [TargetLink Customization and Optimization Guide](#)).

No code generation for resettable TargetLink subsystems

TargetLink does not generate code for a TargetLink subsystem that is also a resettable subsystem.

No code generation for resettable BlackBox subsystems

TargetLink does not generate code for a resettable subsystem that is masked with the BlackBox mask type.

Output reset in Triggered/Enabled subsystems in Simplified Initialization Mode

TargetLink does not support Simulink's output reset behavior if all the following modeling conditions are met:

- Simulink's Initialization Mode is set to Simplified.
- The model contains an Enabled and Triggered Subsystem. An output of the subsystem has its Output when disabled option set to Reset.
- The above mentioned subsystem is called by an Enabled Subsystem or Action Subsystem.

Output reset of D-FlipFlop blocks in Simplified Initialization Mode

TargetLink does not support the Simulink output reset behavior if all the following modeling conditions are met:


- The Simulink Initialization Mode is set to Simplified.
- The model contains a D-FlipFlop. The block is located in one of the following subsystems:
 - Enabled Subsystem or Action Subsystem
 - a subsystem that is called by an Enabled Subsystem or Action Subsystem

Output reset in function-call-triggered subsystems

TargetLink does not support the reset setting of the Output when disabled Simulink parameter for Outport blocks in function-call-triggered subsystems.

To avoid differences of the generated production code with the model's behavior, set the **Output when disabled** property of the Output block in the function-call-triggered subsystem to held (default).

Frame updating


If you add, move, or remove a Simulink Inport, Output, Trigger, or Enable block to or from the topmost level of a TargetLink subsystem, this interface modification will not be immediately displayed on the level of the parent Simulink system. This behavior is different from standard Simulink behavior because the TargetLink simulation frame involves two more levels between the TargetLink subsystem and the parent Simulink system. These levels allow you to switch between simulation modes. For details, refer to [Basics on Simulation Modes and Preconditions](#) ( [TargetLink Preparation and Simulation Guide](#)).

The interface is updated when the simulation mode is switched or reactivated, or when production code is generated.

Enabled or triggered subsystems

If the Simulink Classic initialization mode is active, the following limitation applies: If the initial values at the outputs of triggered or enabled subsystems are not specified, TargetLink might use initial values different from the initial values used by Simulink for these outputs. Hence a MIL simulation can differ from the SIL/PIL simulations.

Externally triggered function calls

Differences in MIL and SIL/PIL simulation might occur in an externally triggered function-call subsystem or Stateflow chart. The reason is the following: For each externally triggered function-call subsystem or Stateflow chart a  [root function](#) is generated, which is called during SIL/PIL simulation instead of the corresponding subsystem or Stateflow chart. Additionally, a root function is generated for the comprising TargetLink subsystem. In SIL/PIL simulation mode, first the root function generated for the TargetLink subsystem is called. Subsequently, the remaining root functions are called in the order determined by the external function-call signals. If the root function <A> requires variables that are up-to-date when the root function is called, and the function <A> is called before the function is called, MIL and SIL/PIL simulation results might differ (one sample delay).

Show output port for Enable blocks

The **Show output port** option is not supported for Enable blocks that are at the root level of the TargetLink root system or of an incremental, scaling-invariant, external, or function-reused system.

Show output port for Trigger blocks

The **Show output port** option is not supported for Trigger blocks that are at the root level of the TargetLink root system or of an incremental, scaling-invariant, external, or function-reused system.

Scaling-invariant systems

TargetLink does not support bus signals that cross the boundaries of scaling-invariant systems (e.g., subsystems, charts, and referenced models that are marked as scaling-invariant in TargetLink).

For scaling-invariant Stateflow functions, only the following actuals are allowed:

- Stateflow data objects (optionally providing index access)
- Calls of a Stateflow function
- Enumeration constants

Scaling-invariant subsystems


Scaling invariance for in-/outputs Scaling invariance is supported for fully specified inports only. If a subsystem's inport is selected to be scaling-invariant, a TargetLink InPort block must specify the scaling of the signal to be used inside the system. This limitation also applies to TargetLink Outport blocks.

System preparation

For system preparation the following TargetLink limitations apply:

- No checks are made whether a replacing block realizes identical algorithms and is compliant with the targeted environment (TargetLink or non-TargetLink).
- Simulink scaling parameters that specify integer word lengths other than 8, 16, or 32 bit cannot be mapped. In this case, a warning message is produced. By customization, you can introduce equivalents to non-(8, 16, 32)-bit data types which are then mapped to corresponding TargetLink scaling parameter sets.
- Subsystems or models can be prepared only if they can be initialized. This limitation does not apply to libraries.
- Subsystems or models can be prepared only if there is read/write access to all blocks.

Parameterized links

Generally, you are recommended to avoid parameterized links, use them only temporarily during model creation. There are other ways for assigning different values to instance-specific parameters, for example, by using self-modifying masks or mask variables. For details on this ways, refer to [How to Specify Instance-Specific Values for Library Subsystems](#) ( [TargetLink Customization and Optimization Guide](#)) or the Simulink Help.

When working with parameterized links the following TargetLink limitations apply:

- If an instance of a reused system has a parameterized library link (i.e., some data in the system instance differs from data of the linked library system), this difference makes function reuse impossible if it leads to code differences in the reused function.



The TargetLink Code Generator might not detect all the possible differences.

TargetLink subsystems in subsystems with state reset


TargetLink does not support state reset if the model has the following characteristics:

- The TargetLink subsystem is not a conditional subsystem but is part of a conditional subsystem.
- The conditional subsystem resets its state.
- The TargetLink subsystem does not contain a TargetLink Function block with Force init function set to on.
- The TargetLink subsystem, or one of its non-conditional child subsystems, contains a block with a state variable, e.g., a Unit Delay block.





Related topics**Basics**

Getting Started ( TargetLink Orientation and Overview Guide)
 Transforming Floating-Point Code to Fixed-Point Code (Scaling Variables)
 ( TargetLink Preparation and Simulation Guide)

HowTos

How to Create a Model from Scratch ( TargetLink Orientation and Overview Guide)

References

Function Block ( TargetLink Model Element Reference)
 InPort Block ( TargetLink Model Element Reference)
 OutPort Block ( TargetLink Model Element Reference)
 Subsystem Block ( TargetLink Model Element Reference)

User Interface Limitations

Introduction

The following limitations apply to the graphical user interface.

MLIE filter and mapping mechanisms are not supported by the GUI

The GUI for MATLAB Import Export (MLIE) API does not support the filter and mapping mechanisms, as MLIE is for advanced users who usually run it directly via its API functions.

Delayed display update

Due to the fact that TargetLink is based on MATLAB, the state of the controls reflects only the last user input. Therefore, the display cannot respond when, for example, the current system (system preparation) is closed, or deleted.

V-ECU Implementation Generation Limitations

Introduction

The following limitations apply to V-ECU implementation generation.

V-ECU generation

TargetLink cannot export V-ECU implementations for the following TargetLink subsystems:

- With target optimized (TOM) production code
- With production code generated in RTOS mode
- That contain asynchronous root functions

Tip

The VECU_EVENT_SIMULATION demo model shows a workaround.

- That contain root functions with matrix interface variables

SIC Generation Limitations

Introduction

The following limitations apply to the generation of [Simulink implementation container files](#).

No TOM support

TargetLink cannot generate a [Simulink implementation container \(SIC\)](#) file from a TargetLink subsystem if its production code or the production code of a nested [code generation unit \(CGU\)](#) is optimized for a specific target (TOM).

No RTOS support

TargetLink cannot generate a [Simulink implementation container \(SIC\)](#) file from a TargetLink subsystem if its production code or the production code of a nested [code generation unit \(CGU\)](#) was generated in RTOS code generation mode.

No muxed function-call trigger

TargetLink cannot generate a [Simulink implementation container \(SIC\)](#) file from a TargetLink subsystem if it contains externally triggered function-call subsystems that are triggered by more than one external function-call signal.

No model port variables in reused subsystems

TargetLink does not support [model port variables](#) at blocks that are contained in reusable model parts.

Modeling for data integrity via Rate Transition blocks

TargetLink can generate special [simulation code](#) to ensure data integrity from Simulink's Rate Transition block if all of the following conditions are met:

- The block is placed between the externally triggered function-call subsystems.
- The block is directly connected to these subsystems
- The subsystems are not configured for function reuse.
- The interface of these subsystems is modeled via TargetLink port blocks that meet the following conditions:
 - They are not used for RTOS communication.
 - Their block variable is of global scope.
 - For floats, their block variable must be of the same data type.

Note

You can model SIC runnable functions that are time-triggered or asynchronous in the same TargetLink subsystem. However, the blocks for which a time-triggered SIC runnable function will be generated must be moved to a Function Call subsystem. Refer to [Details on Generating SIC Files for ConfigurationDesk That Contain Several SIC Runnable Functions](#) ([TargetLink Interoperation and Exchange Guide](#)).

TargetLink displays a message if any of these conditions is not met.

Model port variables

The following restrictions apply to [model port variables](#):

- They must not define a 2-D matrix.
- They must not have a scaling.
- They must not have an enumeration data type.
- They must not be used at more than one TargetLink port block within the hierarchy of the TargetLink subsystem.

Assigning multiple model implementations to the same application process

Because you can freely name variables in TargetLink, TargetLink cannot ensure that no naming conflicts occur when assigning several model implementations (SIC files) to the same application process in ConfigurationDesk.

To resolve potential naming conflicts, refer to [Assigning multiple model implementations to the same application process in ConfigurationDesk](#) ([TargetLink Interoperation and Exchange Guide](#)).

Related topics

Basics

[Interoperating with ConfigurationDesk](#) ([TargetLink Interoperation and Exchange Guide](#))

Adaptive AUTOSAR-Related Limitations

One CGU for SIL	TargetLink supports SIL simulation for Adaptive AUTOSAR components only for a single code generation unit (CGU) .
No Stateflow data objects for persistency access	A Data Store Memory block used to model the access to a key-value storage as defined by the functional cluster <code>ara::per</code> must not be used by Stateflow data object whose Scope property is set to Data Store Memory .
Adaptive AUTOSAR communication via Data Stores and Custom Code Type II	For Custom Code (type II) blocks whose property Guaranteed complete write is not activated, TargetLink does not support <i>write-only</i> access to data stores that are specified for Adaptive AUTOSAR communication.
Property Manager	The Property Manager does not support Adaptive AUTOSAR. If you load a model that contains Adaptive AUTOSAR elements, the Adaptive AUTOSAR-specific properties are not visible in the Property Manager and E09558 is displayed.
Function reuse	A subsystem that is specified for function reuse or resides in such a subsystem cannot simultaneously contain a Function block whose Role is set to Method Call .
Code decorations	TargetLink does not decorate ARA adapter code modules with the CodeDecorationSet that is set as default.
Simulation unit export	If the production code for the TargetLink subsystem or incremental CGUs below this particular subsystem was generated with the Code generation mode property set to Adaptive AUTOSAR , TargetLink does not generate the following simulation units for the TargetLink subsystem: <ul style="list-style-type: none"> ▪ SIC files ▪ FMUs ▪ V-ECU implementations ▪ Stand-alone S-functions
PIL simulation	TargetLink does not support PIL simulation if the Code generation mode for at least one TargetLink subsystem or incremental CGU is set to Adaptive AUTOSAR .

Code generation	<p>TargetLink does not support code generation for models with a subsystem whose Function block has its Role set to Adaptive AUTOSAR Function if both of the following conditions apply:</p> <ul style="list-style-type: none"> ▪ The subsystem is modeled directly below the TargetLink subsystem. ▪ The DD AdaptiveAutosarFunction object that is associated with the subsystem references a DD Module object that is influenced by a DD ModuleTemplate object.
AUTOSAR Import/Export of data types	<p>TargetLink does not support the following CppImplementationDataType data types:</p> <ul style="list-style-type: none"> ▪ CustomCppImplementationDataType <p>TargetLink does not support StdCppImplementationDataTypes of the following categories:</p> <ul style="list-style-type: none"> ▪ VARIANT ▪ VECTOR ▪ ASSOCIATIVE_MAP ▪ STRING
AUTOSAR Import/Export of communication specification elements	<p>TargetLink does not support the import or export of ComSpec communication specification elements for port prototypes that are typed by service interfaces.</p>

Limitations for Arrays of Buses and Arrays of Structs

Arrays of buses	<p>The following limitations apply to signals that are arrays of buses:</p> <ul style="list-style-type: none"> ▪ Writing array-of-bus signals to data stores via Data Store Write blocks is not supported. ▪ Reading array-of-bus signals from data stores via Data Store Read blocks is not supported. ▪ Writing non-scalar leaf bus elements via Data Store Write blocks is possible only for the leaf bus elements as a whole. ▪ Reading non-scalar leaf bus elements via Data Store Read blocks is possible only for the leaf bus elements as a whole. ▪ TargetLink does not support signal lines that represent arrays of buses.
Arrays of structs	<p>The following limitations apply to the specification of DD Variable objects for array-of-struct variables:</p> <ul style="list-style-type: none"> ▪ The ArrayOfStructValue property is supported only for the leaf struct components of array-of-struct variables.

- The `ArrayOfStructValue` property and the `Value` property cannot be used in the same array of structs.
- When using the `ArrayOfStructValue` property, you cannot specify initial values for a subset of leaf struct components but only for all the leaf struct components at the same time.
- Data variants are not supported.
- Code variants for the `ArrayOfStructValue` property are not supported.
- The `IMPLICIT_STRUCT` type is not supported.
- For leaf struct components of array-of-struct variables, a [non-standard scaling](#) for signed and unsigned integer data types must be specified via the referenced DD `TypedefComponent` objects. This limitation implies that the scaling must be uniform for all the elements of a leaf struct component.

No action is required for the following data types:

- Bool
- Bitfield
- Float
- Enum

Refer to [How to Scale the Leaf Struct Components of Array-Of-Struct Variables with Non-Standard Scaling](#) ([TargetLink Customization and Optimization Guide](#)).

- If Min/Max values are specified for the leaf struct components of array-of-struct variables, these values must be the same for all the elements of a leaf struct component.
- TargetLink does not support access functions in the following cases:
 - For variables and struct components that are specified as arrays of structs.
 - For subcomponents of variables that are specified as arrays of structs.
 - For subcomponents of struct components that are specified as arrays of structs.

Array-of-struct variables in the model

TargetLink does not support the direct use of [array-of-struct variables](#) in the model, except in Data Store Memory blocks:

- Variables of model elements cannot be specified as array-of-struct variables only via their Struct data type.
- The components of array-of-struct variables specified in the Data Dictionary cannot be referenced in the model.

For instructions on how to access the array elements of array-of-struct variables in the model, refer to [How to Access Specific Elements of Arrays of Buses via Data Store Blocks](#) ([TargetLink Preparation and Simulation Guide](#)), [Accessing Array-of-Struct Interface Variables](#) ([TargetLink Preparation and Simulation Guide](#)), and [Example of Dynamically Accessing Array-of-Struct Variables via Data Store Blocks and Custom Code \(Type II\) Blocks](#) ([TargetLink Preparation and Simulation Guide](#)) .

Array-of-struct variables as function parameters of RTE API functions

TargetLink does not support array-of-struct variables as function parameters for the following RTE API functions:

- Rte_(Irv)Read
- Rte_(Irv)Write
- Rte_Send
- Rte_Receive
- Rte_(Irv)IWrite

Simulink-Functions-Related Limitations

No support for scoped Simulink Function subsystems

TargetLink does not support [Simulink Function subsystems](#) whose Simulink parameter Function visibility is set to `scoped`. Scoped, i.e., non-global, exported Stateflow functions are supported.

Call of Simulink Function subsystems only in same CGU

[Simulink Function subsystems](#) can be called only from within the same [Code generation unit \(CGU\)](#).

Restricted call-callee dependencies

TargetLink does not support the following call-callee dependencies:

- Exported Stateflow functions called by Function Caller blocks
- [Simulink Function subsystems](#) called from Stateflow code
- Simulink Function subsystems called from MATLAB code

No OSEK support

You cannot use TargetLink OSEK support with [Simulink Function subsystems](#).

No support of AUTOSAR step functions

TargetLink does not support the use of [Simulink Function subsystems](#) for the following AUTOSAR modeling elements:

- [Runnable subsystem](#)
- [Operation subsystem](#)
- [Operation call with runnable implementation subsystem](#)
- [Adaptive AUTOSAR Function subsystem](#)
- [Method Call subsystem](#)
- [Method Behavior subsystem](#)

Restrictions for AUTOSAR code

TargetLink generates AUTOSAR code from [Simulink Function subsystems](#) only if the following conditions hold:

- RTE API functions are generated only if the Simulink Function subsystem resides in a [runnable subsystem](#).

The Simulink Function subsystem can be called only in the runnable subsystem that contains the Simulink Function subsystem.

- [ARA Adapter code functions](#) are generated only if the Simulink Function subsystem resides in a [Adaptive AUTOSAR Function subsystem](#) or [Method Behavior subsystem](#).

The Simulink Function subsystem can be called only in the Adaptive AUTOSAR Function subsystem or Method Behavior subsystem that contains the Simulink Function subsystem.

Simulink Function subsystems must not be CGUs

Model-based [code generation units \(CGUs\)](#) must not be Simulink Function subsystems.

No variant Simulink Function subsystems

TargetLink does not support models that contain a [Simulink Function subsystem](#) with an enabled Simulink Parameter Enable variant condition.

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.....	86
A.....	87
B.....	90
C.....	91
D.....	94
E.....	96
F.....	97
G.....	98
I.....	98
L.....	100
M.....	101
N.....	103
O.....	104
P.....	105
R.....	106
S.....	108
T.....	110
U.....	112
V.....	112
W.....	113

Numerics

1-D look-up table

output value (y).

A look-up table that maps one input value (x) to one

2-D look-up table

output value (z).

A look-up table that maps two input values (x,y) to one

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 AUTomotive Open System ARchitecture. 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](#) ([TargetLink Orientation and Overview Guide](#)).

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
- Multipoint 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., variant 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.

DataltemMapping (DIM) A DataltemMapping object is a DD object that references a [ReplaceableDataltem \(RDI\)](#) 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 that can 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.

M

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

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 that 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.

tlilib A TargetLink block library that is the source for creating TargetLink models graphically. Refer to [How to Open the TargetLink Block Library](#) ([TargetLink Orientation and Overview Guide](#)).

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](#).

B

block-specific limitations 24

C

Common Program Data folder 6

CommonProgramDataFolder 6

D

Data Dictionary

limitations 53

Documents folder 6

DocumentsFolder 6

F

frame updating 75

L

limitations

block-specific 24

creating a TargetLink subsystem 71

Data Dictionary 53

frame updating 75

triggered TargetLink subsystems 73

Local Program Data folder 6

LocalProgramDataFolder 6

R

reserved names 68

S

saturation

output signals 13

