

TargetLink

File 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.

© 2004 - 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	7
Static Files	9
Fixed-Point Library Files.....	10
Macros and Functions Provided by the Fixed-Point Library.....	10
Understanding the Names of Fixed-Point Library Macros and Functions in TargetLink Production Code.....	13
Controlling the Use of Fixed-Point Library Macros and Functions.....	15
Generated Files	17
Common Generated Files.....	18
Files Generated During Code Generation.....	18
Files Generated During Frame Generation.....	20
Files Generated by the Host MEX Compiler.....	21
Files Generated by the Make Procedure of the Target Compiler.....	22
Files Generated During Code Size Information Generation.....	22
Files Generated by the Message Browser.....	23
Files Generated During Documentation Generation.....	23
Files Containing Input Data for the TargetLink Data Server.....	24
Files Generated During A2L File Generation.....	24
AUTOSAR-Specific Generated Files.....	26
Files Related to Code Generation.....	26
Customization Files	27
System Preparation, Synchronization, and Clearance.....	28
Files Related to System Preparation, Synchronization, and Clearance.....	28
Structure of Libmap Files.....	34
slblock Field.....	34
tlblock Field.....	35
mapSL2TL Field.....	36
mapTL2SL Field.....	36
funcSL2TL Field.....	37
funcTL2SL Field.....	37

filterSL2TL Field.....	38
filterTL2SL Field.....	38
Documentation Generation.....	39
Files Related to Documentation Generation Customization.....	39
Hook Scripts.....	41
Overview of Hook Scripts.....	41
Alphabetical List of Hook Scripts.....	41
Build Hook Scripts.....	44
tl_post_compile_host_hook.....	45
tl_post_compile_target_hook.....	46
tl_post_download_hook.....	48
tl_pre_compare_creator_options_hook.....	49
tl_pre_compile_host_hook.....	50
tl_pre_compile_target_hook.....	53
tl_pre_download_hook.....	55
Code Generation Hook Scripts.....	56
tl_post_codegen_error_hook.....	57
tl_post_codegen_finished_hook.....	58
tl_post_codegen_hook.....	60
tl_post_framegen_hook.....	62
tl_pre_codegen_hook.....	63
tl_pre_codegen_init_hook.....	65
Documentation Hook Scripts.....	67
tl_autodoc_hook.....	67
tl_post_autodoc_filter_hook.....	71
Export Hook Scripts.....	73
tl_post_export_files_hook.....	74
tl_pre_export_files_hook.....	75
Frame Model Generation Hook Scripts (AUTOSAR).....	77
tl_post_add_arportblock_hook.....	77
tl_post_add_comspecblock_hook.....	84
tl_post_add_operationcallsubsystem_hook.....	85
tl_post_add_operationresultprovidersubsystem_hook.....	87
tl_post_add_runnablesubsystem_hook.....	89
tl_post_add_swcsubsystem_hook.....	90
tl_post_swcmodelegen_hook.....	93
tl_pre_add_arportblock_hook.....	95
tl_pre_add_comspecblock_hook.....	102

Function Subsystem Generation Hook Scripts.....	103
Data Dictionary to Model.....	103
tl_post_add_systemsignatureport_hook.....	104
tl_post_sync_systemsignatureport_hook.....	106
tl_post_syncsystemsignature_hook.....	108
tl_pre_add_systemsignatureport_hook.....	111
tl_pre_sync_systemsignatureport_hook.....	113
tl_pre_syncsystemsignature_hook.....	115
Model to Data Dictionary.....	116
tl_post_add_ddsignatureport_hook.....	117
tl_pre_add_ddsignatureport_hook.....	119
Stimulus Creation.....	121
tl_post_create_stimuli_hook.....	121
tl_pre_create_stimuli_hook.....	123
Import/Export Hook Scripts (AUTOSAR).....	125
tl_post_arexport_hook.....	125
tl_post_arimport_hook.....	127
tl_pre_arexport_hook.....	128
tl_pre_arimport_hook.....	131
tl_pre_containerexport_hook.....	132
Model Processing Hook Scripts.....	133
tl_post_load_hook.....	134
tl_pre_save_hook.....	135
Production Code Simulation Hook Scripts.....	136
tl_pre_prodcode_sim_hook.....	136
System Preparation Hook Scripts.....	138
tl_post_clear_hook.....	138
tl_post_preparation_hook.....	139
tl_pre_clear_hook.....	141
tl_pre_preparation_hook.....	142
Upgrade Hook Scripts.....	143
tl_post_upgrade_hook.....	144
tl_pre_upgrade_hook.....	145
Look-Up Table Functions.....	146
Look-Up Table Functions Customization Files.....	146
Code Formatting.....	147
Code Formatting Customization Files.....	147

Bus Initialization.....	148
tlGetBusStructMapping.....	148
MATLAB Import Export API (MLIE).....	150
Customization Files For The MATLAB Import Export API (MLIE).....	150
M Files for Customizing MATLAB.....	152
dsfinish.m.....	152
dspoststartup.m.....	153
dsstartup.m.....	154
 Glossary	 155
 Index	 185

About This Reference

Contents

This reference provides information on:









- Files used by TargetLink
- Files generated by TargetLink

TargetLink

TargetLink, dSPACE's production code generation software, lets you generate highly efficient C code for microcontrollers in electronic control units (ECUs). It also helps you implement control systems that have been modeled graphically in a Simulink/Stateflow diagram on a production ECU.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

Static Files

Introduction

The static files are used to build a production code target application.

Fixed-Point Library Files

Introduction

To provide information on files belonging to the Fixed-Point Library.

Where to go from here

Information in this section

Macros and Functions Provided by the Fixed-Point Library.....	10
Understanding the Names of Fixed-Point Library Macros and Functions in TargetLink Production Code.....	13
Controlling the Use of Fixed-Point Library Macros and Functions.....	15

Macros and Functions Provided by the Fixed-Point Library

Introduction

By default, TargetLink internally manages the macros and functions provided by the Fixed-Point Library.

Note

- Target-specific macros and functions that belong to the TargetLink Target Optimization Modules (TOMs) are non-ANSI C and can contain assembler instructions. If you are not allowed to use assembler instructions in code, you have to configure the TargetLink Code Generator accordingly. For more information, refer to the Code Generation page of the TargetLink Main Dialog Block.
- The target-specific libraries also contain all the generic elements. This lets you link against a single file.

Operations implemented by generic macros and functions

TargetLink provides several generic macros and functions that implement different operations. The following table lists the operations and the locations of the macro definitions and function declarations.

Operation	Description	File
ACOS	Arccosine	trig.h
ADD	Addition (+)	sum.h sumprot.h ¹⁾
ASIN	Arcsine	trig.h
ATAN	Arctangent	trig.h
ATAN2	Four-quadrant inverse tangent	trig.h

Operation	Description	File
COPY	Copies a value from one variable to another.	copy.h
COS	Cosine	trig.h
DIV	Division (/)	div.h
EQ	Equal ²⁾	cmp.h
FIR	FIR filter	fir.h
FIT	Fit of a type into a type with a smaller width ³⁾	fit.h
GE	Greater than or equal to ²⁾	cmp.h
GT	Greater than ²⁾	cmp.h
HYPOT	Hypotenuse	tl_math.h
LE	Less than or equal to ²⁾	cmp.h
LT	Less than ²⁾	cmp.h
MUL	Multiplication (*)	mul.h
NE	Not equal ²⁾	cmp.h
NEG	Unary minus (== -value)	neg.h
SAT	Saturation	sat.h
SHL	Shift left (<<)	shl.h
SHR	Shift right (>>)	shr.h
SIN	Sine	trig.h
SQRT	Integer square root	sqrt.h
SUB	Subtraction (-)	sum.h sumprot.h ¹⁾
TAN	Tangent	trig.h

¹⁾ For calculations protected against overflows. Used only if ExploitComputeThroughOverflow code generator option is set to 1 = **Never**.

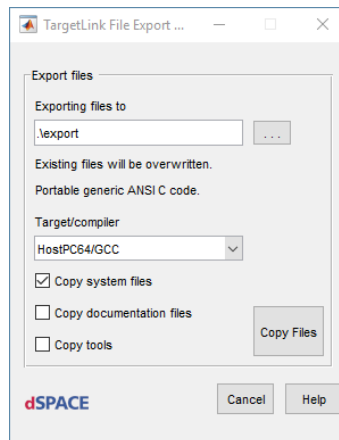
²⁾ Input width: 64-bit

³⁾ Always with saturation.

Code integration

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

TargetLink offers a File Export Utility, which lets you export generated files from the TargetLink environment to a selected destination folder. If you select the **Copy system files** checkbox in the TargetLink File Export Utility dialog, all header files and source files of the Fixed-Point Library required for compiling the code are exported as well.



For more information, refer to [Integrating TargetLink Code in External Applications](#) ([TargetLink Interoperation and Exchange Guide](#)).

If you want to manage the macros and functions provided by the Fixed-Point Library externally, refer to [Specifying the Location of the Sources and Binaries of the TargetLink Fixed-Point Library](#) ([TargetLink Customization and Optimization Guide](#)).

Related topics

Basics

Basics on Identifying Calls to Macros and Functions in Production Code ([TargetLink Preparation and Simulation Guide](#))
 Integrating TargetLink Code in External Applications ([TargetLink Interoperation and Exchange Guide](#))
 Specifying the Location of the Sources and Binaries of the TargetLink Fixed-Point Library ([TargetLink Customization and Optimization Guide](#))

Examples

Example of Identifying a Macro Call in Production Code ([TargetLink Preparation and Simulation Guide](#))

References

Controlling the Use of Fixed-Point Library Macros and Functions..... 15
 ExploitComputeThroughOverflow ([TargetLink Model Element Reference](#))
 File Export Utility ([TargetLink Tool and Utility Reference](#))
 Preferences Editor ([TargetLink Tool and Utility Reference](#))
 TargetLink Main Dialog Block ([TargetLink Model Element Reference](#))
 tl_pref ([TargetLink API Reference](#))
 Understanding the Names of Fixed-Point Library Macros and Functions in TargetLink
 Production Code..... 13

Understanding the Names of Fixed-Point Library Macros and Functions in TargetLink Production Code

Introduction

To understand the naming schema for macros and functions called in the production code. This gives you a better understanding of production code generated by TargetLink.

Naming schema

Consider the following illustration:

```
5.
6.      /* Sum: pidcontroller/e */
7.      Aux_S32 = (Int32) (((UInt32) (Int32) (((Int32) Aux_Sa1 REF a) << 7)) - ((UInt32) (Int32)
8.      (((Int32) Aux_Sa1 U_X a) << 7)));
9.      Sa1_e = C_I16FITI32_SAT Aux_S32, 32767 /* 0.1249961853 */;
10.     LOG_VAR(a, _Sa1_e, Sa1_e);
11.
```

To understand the highlighted macro call, you have to understand how macros and functions provided by TargetLink are named.

The macros and functions provided by TargetLink are named according to the following schema:

Naming Schema

(A__|AC__|AF__|C__|F__) + <output_type> + <OPERATION> + <input_type> {+ <input_type>} [(+ <_SAT>|_PROT)]

This schema consists of several naming elements, which are shown in the following table:

Naming Element	Description	Replaced by
A__	Assembler macro ¹⁾	Not replaced
AC__	Non-ANSI C macro ¹⁾	
AF__	Non-ANSI C function ¹⁾	
C__	ANSI C macro ²⁾	
F__	ANSI C function ²⁾	
<output_type>	The output data type.	One of the following: U8, U16, U32, U32s U64, U64s I8, I16, I32, I32s I64, I64s ³⁾
<OPERATION>	The implemented operation.	For example: <ul style="list-style-type: none"> MUL = multiplication (*) SHR = shift right (>>) FIT = fitting ... For a complete list of supported operations, refer to Operations implemented by generic macros and functions on page 10.
<input_type> ⁴⁾	The input data type.	One of the following: U8, U16, U32, U32s U64, U64s I8, I16, I32, I32s I64, I64s ³⁾

Naming Element	Description	Replaced by
<_SAT> ⁵⁾	Saturation of the operation's output.	Depending on the mode of saturation: <ul style="list-style-type: none"> ▪ _SAT/SATb = saturation at the lower and upper limits ▪ _SATl = saturation at the lower limit only ▪ _SATu = saturation at the upper limit only
_PROT ⁶⁾	Performed calculations are protected against overflows. Used only if the ExploitComputeThroughOverflow code generator option is set to 1 = Never .	Not replaced

¹⁾ Target-specific, belongs to one of TargetLink's Target Optimization Modules (TOMs). Can contain inlined assembler instructions.

²⁾ Generic.

³⁾ U = unsigned, I = signed, <8, 16, 32, 64> = number of bits, s = struct

⁴⁾ For operations taking two arguments, e.g., divisions or fittings, two input types must be provided.

⁵⁾ Only appended if saturation is applied. Mutually exclusive with _PROT.

⁶⁾ Only appended if calculations protected against overflows are performed. Mutually exclusive with <_SAT>.

Naming example

As a simple example, consider the generic macro used for fitting a signed 32-bit integer variable into an unsigned 16-bit one. Because fitting necessitates saturation, the macro has one of the following names:


- C__U16FITI32_SAT
- C__U16FITI32_SATu
- C__U16FITI32_SATl

Which saturation suffix is appended depends on the limits at which saturation has to be applied.


To find the operation implemented by the macro or function, you can parse the macro or function name. For more information, refer to [Macros and Functions Provided by the Fixed-Point Library](#) on page 10.

Related topics



Basics

Basics on Identifying Calls to Macros and Functions in Production Code
( TargetLink Preparation and Simulation Guide)

Examples

Example of Identifying a Macro Call in Production Code ( TargetLink Preparation and Simulation Guide)

References

Code Generator Options ( TargetLink Model Element Reference)
Controlling the Use of Fixed-Point Library Macros and Functions..... 15
ExploitComputeThroughOverflow ( TargetLink Model Element Reference)
Macros and Functions Provided by the Fixed-Point Library..... 10

Controlling the Use of Fixed-Point Library Macros and Functions

Introduction

To a certain extent, you can control whether TargetLink uses Fixed-Point Library macros or functions.

Floating-point arithmetic

All the macros and functions of the TargetLink Fixed-Point Library that relate to floating-point arithmetic are encapsulated by the `TL_NO_FLOATS` or `TL_NO_FLOAT64` preprocessor directive.

This lets you control the use or removal of floating point code from the library during compilation:

Directive	Purpose
<code>TL_NO_FLOATS</code>	Suppresses all the floating-point related macros and functions of the Fixed-Point Library (32-bit and 64-bit).
<code>TL_NO_FLOAT64</code>	Suppresses only 64-bit floating-point related macros and functions of the Fixed-Point Library.

You can define the macros in the code or use additional compiler options.

FIR Filter block

Whether a Fixed-Point Library macro or function is used for a FIR Filter block depends on the settings made on the Delay Line page of the block dialog:



Inline code Checkbox	Generate loops Checkbox	Method Used
Selected	Selected	Macro
Cleared	Selected	Function

64-bit multiplication

You can control whether TargetLink uses functions for 64-bit multiplication via the **Functions for 64-bit Operations** checkbox, located on the **Advanced** page of the TargetLink Main Dialog.

Related topics


Basics

Basics on Conditional Code Compilation via Preprocessor Directives ( TargetLink Customization and Optimization Guide)
 Basics on Identifying Calls to Macros and Functions in Production Code
 ( TargetLink Preparation and Simulation Guide)

Examples

Example of Identifying a Macro Call in Production Code ( TargetLink Preparation and Simulation Guide)

References

Macros and Functions Provided by the Fixed-Point Library..... 10
 TargetLink Main Dialog Block ( TargetLink Model Element Reference)
 Understanding the Names of Fixed-Point Library Macros and Functions in TargetLink
 Production Code..... 13

Generated Files

Where to go from here

Information in this section

Common Generated Files.....	18
AUTOSAR-Specific Generated Files.....	26

Common Generated Files

Where to go from here

Information in this section






Files Generated During Code Generation.....	18
Files Generated During Frame Generation.....	20
Files Generated by the Host MEX Compiler.....	21
Files Generated by the Make Procedure of the Target Compiler.....	22
Files Generated During Code Size Information Generation.....	22
Files Generated by the Message Browser.....	23
Files Generated During Documentation Generation.....	23
Files Containing Input Data for the TargetLink Data Server.....	24
Files Generated During A2L File Generation.....	24

Files Generated During Code Generation

List of files

The following table shows the production code files generated by the Code Generator. Production code files are located in the folder for production code files defined via the DD ProjectFolder and FolderStructure objects.

File	Description
<cgu>.c	Code file for the TargetLink code generation unit.
<cgu>.h	Header file for the TargetLink code generation unit.
udt_<SubsystemID>.h	Header file containing definitions of all user data types used in the TargetLink code generation unit that are not assigned to a specific module.
t1_defines_<SubsystemID>.h t1_aux_defines_<SubsystemID>.h	Header file containing TargetLink-defined preprocessor macros like FX_GROUND, macros of variable classes with the <i>use name</i> attribute and the log macro. t1_aux_defines_<subsystem>.h contains <i>auxiliary</i> preprocessor macros.

File	Description
<function>.c	Code file that contains function definitions and variable declarations for a Function block. On the Options page of the Function block dialog, you have to specify the name of the module or reference a Module object in the TargetLink Data Dictionary. For details, refer to TargetLink Main Dialog Block. If the Function block resides in a code generation unit (CGU), which can be a TargetLink subsystem, a subsystem configured for incremental code generation, or a referenced model, you must not specify a code file name or reference a Module object in the TargetLink Data Dictionary. In this case, the module has the same name as the CGU in which the Function block resides.
<function>.h	Header file that contains declarations for a Function block.
tl_basetypes.h	Header file that contains TargetLink-defined base types. If desired, you can change the file name of <code>tl_basetypes.h</code> . Refer to Basics on TargetLink Base Types, Typedefs, and Header Files ( TargetLink Customization and Optimization Guide).
<code generation unit>*.*	Module used for the following systems if not otherwise specified: <ul style="list-style-type: none"> TargetLink subsystems Subsystems configured for incremental code generation Referenced models
<user>.c	Code file for program code and variables that are assigned to a separate file named by the user. You can specify file names for variable and function classes in the TargetLink Data Dictionary. In addition, you can use a Function Block to create a function from a subsystem. You can specify a file name in the Codefile field on the Options page of the Function Block's dialog.
<user>.h	Code header file for declarations required for the separate user file <code><user>.c</code> .
tl_dsfxp_types.h	Generated only if the  Code generation unit (CGU) requires the Fixed-Point Library. Contains typedef declarations and typedef names for the basic arithmetic and Boolean types. These typedef names are used for the Fixed-Point Library only and are generated to avoid naming conflicts.
tl_std_types.h	Generated only if the Code Generator option UsePlatformTypeAbstractionLayer ( TargetLink Model Element Reference) is set to on. Contains central typedef declarations and typedef names for the basic arithmetic and Boolean types. Refer to Basics on TargetLink Base Types, Typedefs, and Header Files ( TargetLink Customization and Optimization Guide).
tl_limits.h	Generated only if the production code uses the maximum or minimum value of the floating-point data types <code>Float32</code> or <code>Float64</code> . Contains definitions of macros for these values. Refer to Basics on Floating-Point Limits in TargetLink ( TargetLink Preparation and Simulation Guide).
tl_dsfxp_limits.h	Generated only if the production code requires the Fixed-Point Library. Contains macro definitions that are used in the Fixed-Point Library for minimal and maximal values of all TargetLink base types.



The following table shows the files generated by the Code Generator for simulation purposes. Simulation frame files are located in the folder for simulation frame code files defined via the DD ProjectFolder and FolderStructure objects..

File	Description
<cgu>_fri.h <cgu>_fri.c	Contains defines, external variables, and function declarations for simulation.


For simulation specific files generated during frame generation, refer to [Files Generated During Frame Generation](#) on page 20.

Related topics

Basics

Basics on Floating-Point Limits in TargetLink ( [TargetLink Preparation and Simulation Guide](#))
 Basics on the Code Generation Process ( [TargetLink Preparation and Simulation Guide](#))


References

[Files Generated During Frame Generation](#)..... 20
[TargetLink Main Dialog Block](#) ( [TargetLink Model Element Reference](#))

Files Generated During Frame Generation

List of files

This table shows the files generated during frame generation. Simulation frame files are located in the folder for simulation frame code files defined via the DD ProjectFolder and FolderStructure objects.

File	Description
<system>_frm.h	Holds the definitions of the log structure, the definitions of the log macros, and the code coverage structure. During the simulation, the log macro copies the selected variables to this structure.
<system>_sf.c	Implements the simulation S-function frame file, which interfaces the simulation application with Simulink via the TargetLink Simulation Router (TL SR). For details, refer to Basics on Simulation Modes and Preconditions ( TargetLink Preparation and Simulation Guide). A single simulation S-function frame is generated for each TargetLink subsystem. The generated S-functions are used for both SIL and PIL simulations.
<system>_pcf.c	Simulation frame code for the target; the simulation frame code is specific for a TargetLink subsystem, a subsystem configured for incremental code generation, or a referenced model.
tl_clean.bat	Batch file used to clean the current working directory by removing files generated by the make procedure of the target compiler, model-specific files, and documentation files.

File	Description
<code>tl_sim_limits.h</code>	Contains macro definitions that are used in the simulation frame for minimum and maximum values of all TargetLink base types.

Note

The simulation files described here depend on the `tl_sim_types.h` file. This file is generated during the build processes into the simulation frame folder according to the selected MEX compiler and into the build folder according to the selected simulation target. It contains typedef declarations and typedef names for the basic arithmetic and Boolean types. These typedef names are used for simulation purposes only and are generated to avoid naming conflicts.

Related topics**Basics**

Basics on Simulation Modes and Preconditions ( [TargetLink Preparation and Simulation Guide](#))

References

TargetLink Main Dialog Block ( [TargetLink Model Element Reference](#))

Files Generated by the Host MEX Compiler

List of files

The following table shows the files generated by the host MEX compiler. The files are located in the working directory.

File	Description
<code><subsystem>_sf.mexw64</code>	S-function frame DLL for production code application.
<code><customcode>_f1p.mexw64</code>	S-function custom code DLL for floating-point host application.

Files Generated by the Make Procedure of the Target Compiler

List of files

The following table shows the files generated by the make procedure of the target compiler. These files are placed in the **TLBuild** folder and below.

File	Description
<model>.mk	Makefile fragment containing a list of files which are necessary to build a simulation application.
tlsim_<model>_globals.c	Contains definitions of symbols needed for the simulation (as interface variables).
<application>.abs	The absolute file that must be downloaded to the target board
<application>.map	Linker/locator-generated map file; contains information about the sizes and positions of the code segments in the target's memory
*.a, *.o, *.obj, *.xao	Compiler/assembler-generated object files
*.L, *.lis, *.ls, *.lst	Compiler/assembler-generated list files; evaluated during the code size measurement procedure
*.asm, *.src, *.s	Compiler-generated assembler code files
*.elc, *.elk, *.err, *.ers	Error files containing messages that appear during the build process

Files Generated During Code Size Information Generation

List of files

The following table shows the files generated during the generation of code size information. Simulation frame files are located in the folder for simulation frame code files defined via the DD ProjectFolder and FolderStructure objects.

File	Description
<application>.csi	Code size information file for the <application> model
dummy.map	Linker/locator-generated map file for a dummy linkage process without any target frame files; generated to get detailed code size information about the code generated during the generation of the code size information file

Files Generated by the Message Browser

List of files

This table shows the file generated when you click the Log File button of the Message Browser. The file is located in the working directory.

File	Description
tl_messages.log	This file contains all the messages shown in the Message Browser.

Files Generated During Documentation Generation

List of files

The following table shows the files generated when the documentation is generated. These files are located in the **doc** subfolder of the folder for production code files defined via the DD ProjectFolder and FolderStructure objects.

Note

The documentation generation is controlled by template files, refer to [Files Related to Documentation Generation Customization](#) on page 39.

File	Description
<model>_main.html <model>_frame.html <model>_link.html	HTML documentation files generated for the <model> model
<model>_main.pdf	PDF documentation files generated for the <model> model
<model>.png	Screenshot of the main Simulink Editor of the <model> model
<model>_<tlsubsystem>.png	Screenshot of the Simulink Editor for the subsystem
<model>_<tlsubsystem>_<function>.png	Screenshot of the Simulink Editor for the subsystem the function is generated for
<model>_<simulation>.png	Screenshot of a simulation plot. <simulation> is the name of the simulation in TargetLink's Data Server.

Related topics

References


[Files Related to Documentation Generation Customization](#)..... 39

Files Containing Input Data for the TargetLink Data Server

List of files

This table shows the files that contain input data for the TargetLink Data Server:

File	Description	Location
<plot>.mat	File containing the saved data of a TargetLink simulation plot	You can specify the location of the file via the Save button on the Simulation page of the TargetLink Main Dialog. Refer to TargetLink Main Dialog Block.

For more information on the Data Server, refer to [Example of Specifying Signal-Specific Logging Options](#) ( [TargetLink Preparation and Simulation Guide](#)).

Related topics

Examples

[Example of Specifying Signal-Specific Logging Options](#) ( [TargetLink Preparation and Simulation Guide](#))

References

[TargetLink Main Dialog Block](#) ( [TargetLink Model Element Reference](#))


Files Generated During A2L File Generation

List of files

This table shows the ASAM MCD-2 MC files that are generated via the **Generate ASAP2 file** button on the **Tools** page of the TargetLink Main Dialog. A2L files are located in the folder for production code files defined via the DD ProjectFolder and FolderStructure objects.

File	Description
<\$(CGU)>.a2l	Provides a description of the calibratable and measurable variables of the ECU application.
<div> Tip </div> <div> This path contains artifact-location-specific name macros. Refer to Macro expansion for artifact locations. </div>	

Note

If you generate an A2L file using the A2L export module of the TargetLink Data Dictionary, you can specify the name of the A2L file and the export destination folder. Refer to [Exchanging A2L Files](#) ( [TargetLink Interoperation and Exchange Guide](#)).

AUTOSAR-Specific Generated Files

Files Related to Code Generation

Production code

The following table shows the production code files generated by the Code Generator. Production code files are located in the folder for production code files defined via the DD ProjectFolder and FolderStructure objects.

Name	Description
Software component code	
<SWC/Runnable>.c/.h	Contains the definition/implementation of a runnable. The module name is: <ul style="list-style-type: none"> ▪ The specified module of the DD runnable object ▪ The specified module of the DD software component object if the module of the runnable is not specified
Other code	
<TargetLink subsystem>.c/.h	Contains code for the TargetLink subsystem that is not part of other modules such as software component modules.

AUTOSAR files

The following table shows AUTOSAR files that you can export after production code generation:

Name	Description	Location
AUTOSAR files		
<>.arxml	One or more AUTOSAR files that contain a description of the software component(s) you have generated code for.	User-defined

Simulation files

The following table shows the files generated by the Code Generator for simulation purposes. Simulation frame files are located in the folder for simulation frame code files defined via the DD ProjectFolder and FolderStructure objects.

Name	Description
Stub RTE code	
Rte_Type.h	Contains all the user-defined types.
Rte_<SWC>.h/.c	Contains RTE API function and macro definitions/implementations for the software component.

Customization Files

Where to go from here

Information in this section

System Preparation, Synchronization, and Clearance.....	28
Structure of Libmap Files.....	34
Documentation Generation.....	39
Hook Scripts.....	41
Look-Up Table Functions.....	146
Code Formatting.....	147
Bus Initialization.....	148
MATLAB Import Export API (MLIE).....	150
M Files for Customizing MATLAB.....	152

System Preparation, Synchronization, and Clearance

Files Related to System Preparation, Synchronization, and Clearance


Purpose To customize the preparation of Simulink systems (subsystem, library, model) for TargetLink, and vice versa.

Access You can access TargetLink's customization files via the `tlCustomizationFiles('Create',...)` API function.

Description The following files concern system preparation, synchronization, and clearance:

Filename	Description
tl_map_block_rtw_parameters_config tl_map_subsystem_rtw_parameters_config	The customization file derived from this template lets you define the mapping of Simulink RTW data on TargetLink production code options. TargetLink's default mapping is ignored.
tl_map_constraints_config tl_remap_constraints_config	The customization file derived from this template lets you define the mapping of Simulink <code>OutMin</code> and <code>OutMax</code> parameters on TargetLink's <code>output.min</code> and <code>output.max</code> properties. TargetLink's default mapping is ignored.
tl_map_lock_flag_config tl_remap_lock_flag_config	The customization file derived from this template lets you define the mapping of the Simulink <code>LockScale</code> parameter on TargetLink's <code>scalingvalid</code> property. TargetLink's default mapping is ignored.
tl_map_output_scaling_config tl_remap_output_scaling_config	The customization file derived from this template lets you define the mapping of Simulink output scaling data on TargetLink production code options. TargetLink's default mapping is ignored.
tl_map_inheritance_config	The customization file derived from this template lets you define custom rules for setting the <code>output.inheritscaling</code> property for a block. This script is executed when Simulink local output scaling data is mapped to TargetLink's.
tl_map_param_scaling_config tl_remap_param_scaling_config	The customization file derived from this template lets you define the mapping of Simulink parameter scaling data on TargetLink production code options. TargetLink's default mapping is ignored.
tl_map_saturation_flag_config tl_remap_saturation_flag_config	The customization file derived from this template lets you define the mapping of Simulink <code>SaturateOnIntegerOverflow</code> parameter on TargetLink's <code>output.checkmin</code> and <code>output.checkmax</code> properties. TargetLink's default mapping is ignored.

Filename	Description
<code>tl_map_sf_compiled_scaling_config</code>	The customization file derived from this template lets you define the mapping of Stateflow compiled scaling data on TargetLink production code options. TargetLink's default mapping is ignored.
<code>tl_map_sf_scaling_config</code> <code>tl_remap_sf_scaling_config</code>	The customization file derived from this template lets you define the mapping of Stateflow scaling data on TargetLink production code options. TargetLink's default mapping is ignored.
<code>tl_map_signal_scaling_config</code>	The customization file derived from this template lets you define the mapping of Simulink signal data on TargetLink production code options. TargetLink's default mapping is ignored.
<code>tl_prepare_system.log</code>	Log file that contains all messages concerning system preparation. The file is stored in the current MATLAB working folder. New messages are always appended to the existing file content if the same log file (name) is used. You can change the file name.
<code>tl_clear_system.log</code>	Log file that contains all messages concerning system clearance. The file is stored in the current MATLAB working folder. New messages are always appended to the existing file content if the same log file (name) is used. You can change the file name.
<code>tl_sync_system.log</code>	Log file that contains all messages concerning system synchronization. The file is stored in the current MATLAB working folder. New messages are always appended to the existing file content if the same log file (name) is used. You can change the file name.

Deriving customization files from their templates Use TargetLink's `tlCustomizationFiles('Create',...)` API function or the Create Customization Files dialog to derive customization files from their templates. You can open the dialog by executing the `tlCustomizationFiles` command without arguments. Refer to [Deriving Customization Files From Their Templates](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook scripts You can further customize the system preparation process using hook scripts. For information on hook scripts which are invoked during system preparation (Simulink to TargetLink) or system clearance (TargetLink to Simulink), refer to [System Preparation Hook Scripts](#) on page 138.

Examples of libmaps

In the following, some libmap specifications are shown.

Mapping user-defined blocks The following code maps the `userlib/picontroller` block to the `tl_ext/tl_picontroller` block, and vice versa. As the mask variables are identical for both blocks, the `mapSL2TL` and `mapTL2SL` fields contain the same values.

```

i = i + 1;
libmap(i).slblock      = 'userlib/picontroller';
libmap(i).tlblock      = 'tllib_ext/tl_picontroller';
libmap(i).mapSL2TL     = {'Kp', 'Kp'; 'Ki', 'Ki'; 'dt', 'dt'};
libmap(i).mapTL2SL     = {'Kp', 'Kp'; 'Ki', 'Ki'; 'dt', 'dt'};
libmap(i).funcSL2TL    = '';
libmap(i).funcTL2SL    = '';
libmap(i).filterSL2TL  = {'MaskType', 'PICONROLLER'};
libmap(i).filterTL2SL  = {'MaskType', 'TL_PICONROLLER'};

```

This libmap specifies that during system preparation, all blocks whose **MaskType** parameter is **PICONROLLER** should be replaced by the **tl_picontroller** block from the **tllib_ext** library. During replacement, the block parameters **Kp**, **Ki**, and **dt** are written to the replacing block, which has the same parameter set. When a system is cleared from TargetLink, the libmap specifies that all blocks whose **MaskType** parameter is **TL_PICONROLLER** should be replaced by **picontroller** blocks from the **userlib** library.

Note

If not empty, the **mapSL2TL** and **mapTL2SL** fields must be (n,2) cell arrays.

Mapping of S-functions The following code replaces all S-functions whose function is **iofun_s** with the **tl_iofun_s** block from the **tllib_plus** user library, which must be compliant with TargetLink. The **tl_iofun_s** block could be, for example, a TargetLink Custom Code block which runs the same pieces of code as the **iofun_s** S-function.

```

libmap(i).slblock      = 'built-in/S-function';
libmap(i).tlblock      = 'tllib_plus/tl_iofun_s';
libmap(i).mapSL2TL     = {'FunctionName', 'iofun_s'};
libmap(i).mapTL2SL     = {};
libmap(i).funcSL2TL    = '';
libmap(i).funcTL2SL    = '';
libmap(i).filterSL2TL  = {'FunctionName', 'iofun_s'};
libmap(i).filterTL2SL  = {'ReferenceBlock', 'tllib_plus/tl_iofun_s'};

```

Instead of using the **FunctionName** property to filter S-functions, you could also use the block's mask type if it is masked, and have the S-function in a user-defined library:

```

libmap(i).slblock      = 'userlib/iofun_s';
libmap(i).tlblock      = 'tllib_ext/tl_iofun_s';
libmap(i).mapSL2TL     = {};
libmap(i).mapTL2SL     = {};
libmap(i).funcSL2TL    = '';
libmap(i).funcTL2SL    = '';
libmap(i).filterSL2TL  = {'MaskType', 'iofun_mask'};
libmap(i).filterTL2SL  = {'ReferenceBlock', 'tllib_ext/tl_iofun_s'};

```

Parameter-dependent block mapping The **filterSL2TL** and **filterTL2SL** libmap fields can also be used to achieve parameter-dependent mapping. Consider the following preparation demo libmap (refer to <DocumentsFolder>\dSPACE\TargetLink\<Version>\Demos\preparation\tllib_ext_libmap.m):

```

i = i + 1;
libmap(i).slblock      = 'built-in/DeadZone';
libmap(i).tlblock      = 'tllib_ext/poshalfwave';
libmap(i).mapSL2TL     = {};
libmap(i).mapTL2SL     = {'LowerValue', '-Inf'; 'UpperValue', '0'};
libmap(i).funcSL2TL    = '';
libmap(i).funcTL2SL    = '';
libmap(i).filterSL2TL  =
{'RegExp', 'on', 'BlockType', 'DeadZone', 'LowerValue', '-Inf|-'
inf', 'UpperValue', '0'};
libmap(i).filterTL2SL  = {'MaskType', 'DEADZONE_POSHALFWAVE'};

```

The `filterSL2TL` and `filterTL2SL` fields can specify all parameter `name/value` pairs that are supported for the `find_system` Simulink API function. In this example, the `filterSL2TL` field works with the `RegExp` parameter which tells `find_system` to apply regular expressions. During system preparation, the libmaps thus applies to all blocks that are not supported by TargetLink and

- whose `BlockType` parameter is `DeadZone`,
- whose `LowerValue` parameter is `-Inf` or `-inf`,
- whose `UpperValue` parameter is 0.

All blocks that match these conditions are replaced by `poshalfwave` blocks from the `tllib_ext` library. Reversely, when a system is cleared from TargetLink, all blocks whose `MaskType` property is `DEADZONE_POSHALFWAVE` are replaced by Simulink's built-in `DeadZone` blocks, and the `LowerValue` parameter is set to `-Inf` and `UpperValue` parameter to 0.

Consider another preparation demo libmap (refer to `<DocumentsFolder>\dSPACE\TargetLink\<Version>\Demos\preparation\tllib_ext_libmap.m`):

```

i = i + 1;
libmap(i).slblock      = 'built-in/DeadZone';
libmap(i).tlblock      = 'tllib_ext/generic';
libmap(i).mapSL2TL     =
{'start_of_deadzone', 'LowerValue'; 'end_of_deadzone', 'UpperValue'};
libmap(i).mapTL2SL     =
{'LowerValue', 'start_of_deadzone'; 'UpperValue', 'end_of_deadzone'};
libmap(i).funcSL2TL    = '';
libmap(i).funcTL2SL    = '';
libmap(i).filterSL2TL  = {'BlockType', 'DeadZone'};
libmap(i).filterTL2SL  = {'MaskType', 'DEADZONE_GENERIC'};

```

During system preparation, this libmap applies to all `DeadZone` blocks, including those which match the libmap above. This ambiguity is resolved by the rule that libmaps with more filter rules preempt.

Note

According to Simulink API rules, parameter names are case-insensitive while parameter values are case-sensitive.

Type `help find_system` in the MATLAB Command Window to get more information about Simulink's API and the `find_system` command, or consult the Simulink Guide.

Customizing the mapping process via callback functions The following `libmap` customizes the mapping of `io_fun_s` S-function blocks. The `set_io_params` callback function is called up during system preparation when both blocks exist, these are the replacing block and the block that is being replaced.

```
libmap(i).slblock    = 'userlib/iofun_s';
libmap(i).tlblock    = 'tllib_ext/tl_iofun_s';
libmap(i).mapSL2TL   = {};
libmap(i).mapTL2SL   = {};
libmap(i).funcSL2TL  = 'set_io_params';
libmap(i).funcTL2SL  = '';
libmap(i).filterSL2TL = {'MaskType', 'iofun_mask'};
libmap(i).filterTL2SL = {'ReferenceBlock', 'tllib_ext/tl_iofun_s'};
```

The `set_io_params` callback function might be specified as an M-script as follows:

```
function set_io_params(hSLBlock, hTLBlock)

% get name of block that is being replaced
blockName = get_param(hSLBlock, 'Name');

% filter literals that specify an integer
number_of_bits = blockName(isstrprop(blockName, 'digit'));

% write to replacing block
tl_set(hTLBlock, 'param(2).value', number_of_bits);
```

The callback sets the `number_of_bits` parameter of the replacing block to a value that depends of the name of the block that is being replaced. The replacing block is a TargetLink Custom Code block whose 2nd parameter specifies the number of bits.

Tip

With callbacks, you can establish arbitrarily complex data processing.

Redirection of library links A library block whose link points into a specific library is modified so that its link points into another library. This allows to work with TargetLink-compliant libraries while the original, TargetLink-incompliant libraries are preserved.


```
libmap.slblock = 'customlib';
libmap.tlblock = 'customlib_tl';

libmap.filterSL2TL = { ...
'Regexp', 'on', ...
'LookUnderMasks', 'on', ...
'ReferenceBlock', '^customlib/'};
libmap.filterTL2SL = { ...
'Regexp', 'on', ...
'LookUnderMasks', 'on', ...
'ReferenceBlock', '^customlib_tl/'};
```

As opposed to libmaps for block replacement rules, the `slblock` and `tlblock` fields specify library names instead of block paths. The filter rules are designed to find all blocks whose library link points to `customlib` or `customlib_tl`, respectively.

Note

Using library redirection implies that blocks in both libraries match, which is ensured when associated libmaps are generated.

Tip

Libmaps for library link redirection are generated automatically when a library is prepared for TargetLink and saved to another file. Refer to the `Userlib_2` preparation demo.

Related topics

Basics

[Deriving Customization Files From Their Templates](#) (📖 TargetLink Customization and Optimization Guide)

[Details on Synchronizing Simulink and TargetLink Data](#) (📖 TargetLink Preparation and Simulation Guide)

References

[Create Customization Files Dialog Description](#) (📖 TargetLink Tool and Utility Reference)

[System Preparation Hook Scripts](#)..... 138

[tlCustomizationFiles](#) (📖 TargetLink API Reference)

Structure of Libmap Files

Where to go from here

Information in this section

slblock Field.....	34
tlblock Field.....	35
mapSL2TL Field.....	36
mapTL2SL Field.....	36
funcSL2TL Field.....	37
funcTL2SL Field.....	37
filterSL2TL Field.....	38
filterTL2SL Field.....	38

slblock Field

This field is located in libmap files.

Purpose

To specify the Simulink block that will be replaced.

Description

This field must be a valid Simulink identifier. Permissible values are strings that identify the corresponding Simulink block.

Example

The first example shows the identifier of Simulink's continuous Integrator block:`libmap(i).slblock = 'built-in/Integrator';`

The second example identifies a block called 2D Table in a library userlib. This library must reside on the MATLAB search path:`libmap(i).slblock = 'userlib/2D Table';`

The third example shows that string functions must be used if the block identifier contains newline characters: `libmap(i).slblock = sprintf('userlib/Smith\nPredictor');`

Description(cont.)**Note**

Simulink identifiers are case-sensitive.

tlblock Field

This field is located in libmap files.

Purpose

To specify the TargetLink block that will replace Simulink blocks of the type specified by the `slblock` property (see [slblock Field](#) on page 34).

Description

This field must be a valid Simulink identifier. Permissible values are strings that identify the corresponding TargetLink block.

Example

The first example shows the identifier of the in the TargetLink library `tlbib`:
`libmap(i).tlblock = 'tlbib/Gain';`

The second example identifies a PI controller block in the user-supplied library `tlbib_ext`. This block could be a subsystem with TargetLink blocks that realizes a PI controller. The library must reside on the MATLAB search path:
`libmap(i).tlblock = 'tlbib_ext/picontroller';`

The third example shows that string functions must be used if the block identifier contains newline characters: `libmap(i).tlblock = sprintf('tlbib/Look-Up\nTable');`

Description(cont.)**Note**

Simulink identifiers are case-sensitive.

Related topics**References**

[slblock Field](#)..... 34

mapSL2TL Field

This field is located in libmap files.

Purpose To set the identifiers of properties to be mapped during conversion from Simulink to TargetLink.

Description The first column of the `mapSL2TL` field contains the TargetLink properties, the second the corresponding Simulink properties. If there are no parameters to be mapped, `mapSL2TL` must be an empty cell array.

Permissible values are (n, 2) cell arrays of strings, or an empty cell array.

Example

```
mapSL2TL= { ...
    'denom.value' 'Denominator'
    'num.value'   'Numerator'
    'dt'          'SampleTime' };
```

This is the standard parameter mapping for a Discrete Filter Block.

Remarks Property identifiers are not case-sensitive.

Related topics

References

[Discrete Filter Block](#) ( [TargetLink Model Element Reference](#))

mapTL2SL Field

This field is located in libmap files.

Purpose To set the identifiers of properties to be mapped during conversion from TargetLink to Simulink.

Description The first column of the `mapTL2SL` field contains the Simulink properties, the second the corresponding TargetLink properties. If there are no parameters to be mapped, `mapTL2SL` must be an empty cell array.

Permissible values are (n, 2) cell arrays of strings, or an empty cell array.

Example

```
mapTL2SL = ...
    {'RowIndex'      'row.value',
     'ColumnIndex'   'col.value'
     'OutputValues'  'table.value'};
```

This is the standard parameter mapping for a [Look-Up Table \(2D\) Block](#) ([TargetLink Model Element Reference](#)).

Remarks

Property identifiers are not case-sensitive.

funcSL2TL Field

This field is located in libmap files.

Purpose

To apply further processing to the newly inserted TargetLink block.

Description

The **funcSL2TL** function is called when the TargetLink block was inserted, but the Simulink block not yet deleted. The function has two input parameters: The handle of the Simulink block to be replaced, and the handle of the TargetLink block replacing it. This allows you to apply further processing to the inserted TargetLink block depending on properties of the block which is being replaced.

Permissible values are strings denoting the user-supplied function to be called after the Simulink block is replaced by the corresponding TargetLink block, or an empty string if no function is specified.

funcTL2SL Field

This field is located in libmap files.

Purpose

To apply further processing to a newly inserted Simulink block.

Description

The **funcTL2SL** function is called when the Simulink block was inserted, but the TargetLink block not yet deleted. The function has two input parameters: The handle of the TargetLink block to be replaced, and the handle of the replacing

Simulink block. This allows you to apply further processing to the inserted Simulink block depending on properties of the block which is being replaced.

Permissible values are strings denoting the user-supplied function to be called after the TargetLink block is replaced by the corresponding Simulink block, or an empty string if no function is specified.

filterSL2TL Field

Purpose	To specify for which blocks a libmap should apply when a system is prepared for TargetLink.
Description	Specifies a filter rule. These are propertyname/propertyvalue pairs which specify for which blocks the libmap should apply during system preparation. All propertyname/value pairs can be used that are supported for the Simulink API find_system command.

filterTL2SL Field

Purpose	To specify for which blocks a libmap should apply when a system is cleared from TargetLink.
Description	Specifies a filter rule. These are propertyname/propertyvalue pairs which specify for which blocks the libmap should apply when a system is cleared from TargetLink. All propertyname/value pairs can be used that are supported for the Simulink API find_system command.

Documentation Generation


Files Related to Documentation Generation Customization

Purpose To customize the generation of automatic documentation of the Simulink model and the production code generated by TargetLink.

Access You can access TargetLink's customization files via the `tlCustomizationFiles('Create',...)` API function.

Description The table below describes the customization file templates related to the automatic documentation of the Simulink model and the production code generated by TargetLink:

Template	Description
<code>tldoc_html_customized</code>	Template file for customizing the HTML documentation generation.
<code>tldoc_layout</code>	Template file for customizing the language and table layout of generated documentation.
<code>tldoc_pdf_customized</code>	Template file for customizing the PDF documentation generation.

Use TargetLink's `tlCustomizationFiles('Create',...)` API function or the Create Customization Files dialog to derive customization files from their templates. You can open the dialog by executing the `tlCustomizationFiles` command without arguments. Refer to [Deriving Customization Files From Their Templates](#) ( [TargetLink Customization and Optimization Guide](#)).


Hook scripts You can further customize the documentation generation process using the

- [tl_autodoc_hook](#) on page 67
- [tl_post_autodoc_filter_hook](#) on page 71

hook scripts, which can be invoked during automatic document generation.


Related topics

Basics

[Deriving Customization Files From Their Templates](#) ( TargetLink Customization and Optimization Guide)

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

References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[Document Generation Utility](#) ( TargetLink Tool and Utility Reference)

[tlCustomizationFiles](#) ( TargetLink API Reference)

Hook Scripts

Where to go from here

Information in this section

Overview of Hook Scripts.....	41
Build Hook Scripts.....	44
Code Generation Hook Scripts.....	56
Documentation Hook Scripts.....	67
Export Hook Scripts.....	73
Frame Model Generation Hook Scripts (AUTOSAR).....	77
Function Subsystem Generation Hook Scripts.....	103
Import/Export Hook Scripts (AUTOSAR).....	125
Model Processing Hook Scripts.....	133
Production Code Simulation Hook Scripts.....	136
System Preparation Hook Scripts.....	138
Upgrade Hook Scripts.....	143

Overview of Hook Scripts

Alphabetical List of Hook Scripts

Name	Purpose
tl_autodoc_hook on page 67	Lets you insert your own text and images into the generated HTML documentation.
tl_post_add_arportblock_hook on page 77	Lets you customize AUTOSAR frame model generation.
tl_post_add_comspecblock_hook on page 84	Lets you customize AUTOSAR frame model generation/update.
tl_post_add_ddsignatureport_hook on page 117	Lets you customize the generation of a DD Signature object from a function subsystem's signature.

Name	Purpose
tl_post_add_operationcallsubsystem_hook on page 85	Lets you customize AUTOSAR frame model generation.
tl_post_add_operationresultprovidersubsystem_hook on page 87	Lets you customize AUTOSAR frame model generation.
tl_post_add_runnablesubsystem_hook on page 89	Lets you customize AUTOSAR frame model generation.
tl_post_add_swcsubsystem_hook on page 90	Lets you customize AUTOSAR frame model generation.
tl_post_add_systemsignaturereport_hook on page 104	Lets you customize the synchronization of DD Signature and Function Block objects.
tl_post_arexport_hook on page 125	Lets you customize the AUTOSAR file export from the TargetLink Data Dictionary.
tl_post_arimport_hook on page 127	Lets you customize the AUTOSAR file import to the TargetLink Data Dictionary.
tl_post_autodoc_filter_hook on page 71	Lets you enter your own commands, e.g., to filter the list of the found Autodoc Customization blocks.
tl_post_clear_hook on page 138	Lets you customize the phase when a TargetLink system is being cleared from TargetLink data.
tl_post_codegen_error_hook on page 57	Lets you customize the phase after code generation finished with errors.
tl_post_codegen_finished_hook on page 58	Lets you customize the phase after code generation finished or aborted with an error.
tl_post_codegen_hook on page 60	Lets you customize the code generation phase.
tl_post_compile_host_hook on page 45	Lets you customize the compilation phase for SIL simulation applications.
tl_post_compile_target_hook on page 46	Lets you customize the compilation phase for PIL simulation applications.
tl_post_create_stimuli_hook on page 121	Lets you customize AUTOSAR frame model generation and the synchronization of DD Signature and Function Block objects.
tl_post_download_hook on page 48	Lets you customize the download phase for production code target simulation applications.
tl_post_export_files_hook on page 74	Lets you customize the file export.
tl_post_framegen_hook on page 62	Lets you customize the simulation frame generation phase.
tl_post_load_hook on page 134	Lets you execute commands after loading a model from file.

Name	Purpose
tl_post_preparation_hook on page 139	Lets you customize the phase when a Simulink system is being prepared for TargetLink.
tl_post_swcmodelegen_hook on page 93	Lets you customize AUTOSAR frame model generation.
tl_post_sync_systemsignaturereport_hook on page 106	Lets you customize the synchronization of DD Signature and Function Block objects.
tl_post_syncsystemsignature_hook on page 108	Lets you customize the synchronization of DD Signature and Function Block objects.
tl_post_upgrade_hook on page 144	Lets you customize the system upgrade.
tl_pre_add_arportblock_hook on page 95	Lets you customize AUTOSAR frame model generation.
tl_pre_add_comspecblock_hook on page 102	Lets you customize AUTOSAR frame model generation/update.
tl_pre_add_ddsignaturereport_hook on page 119	Lets you customize the generation of a DD Signature object from a function subsystem's signature.
tl_pre_add_systemsignaturereport_hook on page 111	Lets you customize the synchronization of DD Signature and Function Block objects.
tl_pre_arexport_hook on page 128	Lets you customize the AUTOSAR file export from the TargetLink Data Dictionary.
tl_pre_arimport_hook on page 131	Lets you customize the AUTOSAR file import to the Data Dictionary.
tl_pre_clear_hook on page 141	Lets you customize the phase when a TargetLink system is being cleared from TargetLink data.
tl_pre_codegen_hook on page 63	Lets you customize the code generation phase.
tl_pre_codegen_init_hook on page 65	Lets you customize the code generation phase.
tl_pre_compare_creator_options_hook on page 49	Lets you customize the build process.
tl_pre_compile_host_hook on page 50	Lets you customize the compilation phase for SIL simulation applications.
tl_pre_compile_target_hook on page 53	Lets you customize the compilation phase for PIL simulation applications.
tl_pre_containerexport_hook on page 132	Lets you customize the container export.
tl_pre_create_stimuli_hook on page 123	Lets you customize AUTOSAR frame model generation and the synchronization of DD Signature and Function Block objects.
tl_pre_download_hook on page 55	Lets you customize the download phase for PIL simulation applications.

Name	Purpose
tl_pre_export_files_hook on page 75	Lets you customize the file export.
tl_pre_preparation_hook on page 142	Lets you customize the phase when a Simulink system is being prepared for TargetLink.
tl_pre_prodcodsim_hook on page 136	Lets you modify the values of the production code variables, e.g., parameters and data variant switch variables.
tl_pre_save_hook on page 135	Lets you execute commands before saving a model to file.
tl_pre_sync_systemsignaturereport_hook on page 113	Lets you customize the synchronization of DD Signature and Function Block objects.
tl_pre_syncsystemsignature_hook on page 115	Lets you customize the synchronization of DD Signature and Function Block objects.
tl_pre_upgrade_hook on page 145	Lets you customize the system upgrade.

Build Hook Scripts

Where to go from here

Information in this section

tl_post_compile_host_hook	45
tl_post_compile_target_hook	46
tl_post_download_hook	48
tl_pre_compare_creator_options_hook	49
tl_pre_compile_host_hook	50
tl_pre_compile_target_hook	53
tl_pre_download_hook	55

tl_post_compile_host_hook

Purpose Lets you customize the compilation phase for SIL simulation applications.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_compile_host.m** immediately after the production code DLL was compiled.



Description You can enter your own commands to be executed after the SIL simulation application was compiled.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model.
<code>modelData</code>	Yes	Options entered in the TargetLink Main Dialog.
<code>mexCompiler</code>	Yes	Selected MEX compiler.
<code>makeCmd</code>	Yes	Make command that was used for compiling the SIL simulation application.
<code>buildDir</code>	Yes	Build directory.
<code>makeFileName</code>	Yes	Name of the generated make file fragment containing the list of files needed to build a simulation application.
<code>silCompiler</code>	Yes	Selected SIL compiler.
<code>cmdResult</code>	Yes	Result of the make command.

Related topics

Basics

Basics on the Code Compilation Process ( [TargetLink Preparation and Simulation Guide](#))
 Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

HowTos

How to Compile Production Code in SIL Simulation Mode ( [TargetLink Preparation and Simulation Guide](#))

References


Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))
 tl_pre_compile_host_hook..... 50

tl_post_compile_target_hook

Purpose

Lets you customize the compilation phase for PIL simulation applications.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_compile_target.m** immediately after the PIL simulation application was compiled.



Description You can enter your own commands to be executed after the PIL simulation application was compiled.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model.
<code>modelData</code>	Yes	Options entered in the TargetLink Main Dialog.
<code>simConfig</code>	Yes	Simulation configuration.
<code>makeCmd</code>	Yes	Make command that was used for compiling the PIL simulation application.
<code>buildDir</code>	Yes	Build directory.
<code>makeFileName</code>	Yes	Name of the generated make file fragment containing the list of files needed to build a simulation application.
<code>cmdResult</code>	Yes	Result of the make command.

Related topics

Basics

Basics on the Code Compilation Process ( TargetLink Preparation and Simulation Guide)
 Using the TargetLink M-Script Interface (API) ( TargetLink Interoperation and Exchange Guide)

HowTos

How to Compile Production Code in PIL Simulation Mode ( TargetLink Preparation and Simulation Guide)

References

Create Customization Files Dialog Description ( TargetLink Tool and Utility Reference)
 tl_pre_compile_target_hook..... 53

tl_post_download_hook

Purpose Lets you customize the download phase for production code target simulation applications.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_download.m** immediately after the PIL simulation application was downloaded to the evaluation board.


Description You can enter your own commands to be executed after the download of the PIL simulation application.

Predefined variables The following predefined variables are available:


Predefined Variable	Read Only	Description
model	Yes	Name of the Simulink model.
simConfig	Yes	Simulation configuration.

Related topics

Basics

Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

References


Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))
 tl_pre_download_hook..... 55

tl_pre_compare_creator_options_hook

Purpose

Lets you customize the build process.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated during the build process of the simulation application, before the comparison of the Code Generator options of the code generation units whose production code is to be compiled.

Description

You can enter your own commands to be executed during the build process, before the Code Generator options are compared.

Predefined variables


The following predefined variables are available:

Predefined Variable	Read Only	Description
excludedOptions	No	List of Code Generator options to be excluded from comparison.


Example

```
% Exclude CodeCoverageLevel Code Generator option from comparison.
excludedOptions = {'CodeCoverageLevel'};
```

Related topics**Basics**

[Customizing Production Code Generation via Hook Scripts](#) ( [TargetLink Customization and Optimization Guide](#))

References


[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

tl_pre_compile_host_hook

Purpose

Lets you customize the compilation phase for SIL simulation applications.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by `tl_compile_host.m` immediately before the production code DLL is compiled.

Description You can enter your own commands to be executed before the SIL simulation application is compiled.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model.
<code>modelData</code>	Yes	Options entered in the TargetLink Main Dialog.
<code>mexCompiler</code>	Yes	Selected MEX compiler.
<code>makeCmd</code>	No	Make command for compiling the SIL simulation application. You can modify this string to customize the compilation of the SIL simulation application.
<code>buildDir</code>	Yes	Build directory.
<code>makeFileName</code>	Yes	Name of the generated make file fragment containing the list of files needed to build a simulation application.
<code>silCompiler</code>	Yes	Selected SIL compiler.

Example



```
% Enable verbose mode of MAKE utility.
makeCmd = strrep(makeCmd,'dsmake.exe','dsmake.exe VERBOSE=ON');

% Add libraries to be linked. The libraries must be specified as an
Nx1 cell array of strings.
% NOTE: To add OBJ files, use APPL_OBJ_FILES instead of
MODEL_LIB_FILES as the macro name.
additionalFiles = {'MyUserLib_First.lib \';
'MyUserLib_Second.lib \'};
macroToBeChange = 'MODEL_LIB_FILES';
makeFileFullName = fullfile(buildDir, makeFileName);
fid = fopen(makeFileFullName, 'rt');
makeFileContent =
textscan(fid,'%s','delimiter','\n','whitespace','');
fclose(fid);
idxInLineList = strfind(makeFileContent{1},macroToBeChange);
macroFileLine = find(cellfun('isempty',idxInLineList)==0);
makeFileContent = [makeFileContent{1}(1:macroFileLine);
additionalFiles; makeFileContent{1}(macroFileLine+1:end)];
fid = fopen(makeFileFullName, 'wt');
fprintf(fid, '%s\n', makeFileContent{:});
fclose(fid);

% Disable parallel compilation by setting the
EnableParallelCompiling option to 0 (default 1).
% To do it,
% - read the contents of the makefile fragment
% - find the line containing the TL_MAX_SIM_VARS macro
% - add the new option below this line
makeFileOptionToBeAdded = 'EnableParallelCompiling=0';
macroToBeFind = 'TL_MAX_SIM_VARS';
makeFileFullName = fullfile(buildDir, makeFileName);
fid = fopen(makeFileFullName, 'rt');
makeFileContent =
textscan(fid,'%s','delimiter','\n','whitespace','');
fclose(fid);
idxInLineList = strfind(makeFileContent{1},macroToBeFind);
macroFileLine = find(cellfun('isempty',idxInLineList)==0);
makeFileContent = [makeFileContent{1}(1:macroFileLine);...
    makeFileOptionToBeAdded;...
    makeFileContent{1}(macroFileLine+1:end)];
fid = fopen(makeFileFullName, 'wt');
fprintf(fid, '%s\n', makeFileContent{:});
fclose(fid);
```

Related topics


Basics

Basics on the Code Compilation Process ( [TargetLink Preparation and Simulation Guide](#))
 Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

HowTos

How to Compile Production Code in SIL Simulation Mode ( [TargetLink Preparation and Simulation Guide](#))

References


Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))
 tl_post_compile_host_hook..... 45

tl_pre_compile_target_hook

Purpose

Lets you customize the compilation phase for PIL simulation applications.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_compile_target.m** immediately before the PIL simulation application is compiled.

Description You can enter your own commands to be executed before the PIL simulation application is compiled.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
model	Yes	Name of the Simulink model.
modelData	Yes	Options entered in the TargetLink Main Dialog.
simConfig	Yes	Simulation configuration.
makeCmd	No	Make command for compiling the PIL simulation application. You can modify this string to customize the compilation of the PIL simulation application.
buildDir	Yes	Build directory.
makeFileName	Yes	Name of the generated make file fragment containing the list of files needed to build a simulation application.



Example

```
% Enable verbose mode of MAKE utility.
makeCmd = strrep(makeCmd,'dsmake.exe','dsmake.exe VERBOSE=ON');

% Add libraries to be linked. The libraries must be specified as an
Nx1 cell array of strings.
% NOTE: To add OBJ files, use APPL_OBJ_FILES instead of
MODEL_LIB_FILES as the macro name.
additionalFiles = {'MyUserLib_First.lib \';
'MyUserLib_Second.lib \'};
macroToBeChange = 'MODEL_LIB_FILES';
makeFileFullName = fullfile(buildDir, makeFileName);
fid = fopen(makeFileFullName, 'rt');
makeFileContent =
textscan(fid,'%s','delimiter','\n','whitespace','');
fclose(fid);
idxInLineList = strfind(makeFileContent{1},macroToBeChange);
macroFileLine = find(cellfun('isempty',idxInLineList)==0);
makeFileContent = [makeFileContent{1}(1:macroFileLine);
additionalFiles; makeFileContent{1}(macroFileLine+1:end)];
fid = fopen(makeFileFullName, 'wt');
fprintf(fid, '%s\n', makeFileContent{:});
fclose(fid);
```

Related topics


Basics

Basics on the Code Compilation Process ( [TargetLink Preparation and Simulation Guide](#))
 Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

HowTos

How to Compile Production Code in PIL Simulation Mode ( [TargetLink Preparation and Simulation Guide](#))

References


Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))
 tl_post_compile_target_hook..... 46

tl_pre_download_hook

Purpose

Lets you customize the download phase for PIL simulation applications.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_download.m` immediately before the PIL simulation application is downloaded to the evaluation board.

Description

You can enter your own commands to be executed before the download of the PIL simulation application.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
model	Yes	Name of the Simulink model.
simConfig	Yes	Simulation configuration.

Related topics**Basics**

[Using the TargetLink M-Script Interface \(API\)](#) (📖 TargetLink Interoperation and Exchange Guide)

References

[Create Customization Files Dialog Description](#) (📖 TargetLink Tool and Utility Reference)

tl_post_download_hook..... 48

Code Generation Hook Scripts

Where to go from here**Information in this section**

tl_post_codegen_error_hook.....	57
tl_post_codegen_finished_hook.....	58
tl_post_codegen_hook.....	60
tl_post_framegen_hook.....	62
tl_pre_codegen_hook.....	63
tl_pre_codegen_init_hook.....	65

tl_post_codegen_error_hook

Purpose Lets you customize the phase after code generation finished with errors.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is called by tl_generate_code.m immediately after the production code generation finished with errors for any of the specified code generation units.


Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
model	Yes	Name of the Simulink model (only for model-based code generation units).
tlSystem	Yes	Simulink path of the model-based code generation unit for which production code was generated.
modelData	Yes	Options entered in the TargetLink Main Dialog (only for model-based code generation units).
ddCodeGenUnit	Yes	Data Dictionary path of the CodeGenerationUnit object for which production code was generated.
simMode	Yes	Intended simulation mode: <ul style="list-style-type: none"> ▪ none - Pure code generation, no simulation. ▪ TL_CODE_HOST - Software-in-the-loop (SIL) simulation. ▪ TL_CODE_TARGET - Processor-in-the-loop (PIL) simulation. ▪ TL_CODE_SFCN - Simulation with stand-alone S-Function.


Predefined Variable	Read Only	Description
<code>tlcgOptions</code>	Yes	Code Generator options used for code generation. To get a list of supported Code Generator options, enter <code>tl_get_supported_cgoptions</code> in the MATLAB Command Window.
<code>bFirst</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If code was generated for the first code generation unit. ▪ <code>0</code> - If code was not generated for the first code generation unit.
<code>bLast</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If code was generated for the last code generation unit. ▪ <code>0</code> - If code was not generated for the last code generation unit.

Related topics

Basics

[Basics on Using Hook Scripts](#) ( [TargetLink Customization and Optimization Guide](#))

References


[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

tl_post_codegen_finished_hook

Purpose

Lets you customize the phase after code generation finished or aborted with an error.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter `tlCustomizationFiles` ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_generate_code.m` one time after production code generation finished for all of the specified model-based code generation units and one time after production code generation finished for all of the specified DD-based code generation units. It is also evaluated if the production code generation aborts with an error.

Description

You can enter your own commands to be executed after code generation.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model (only for model-based code generation units).
<code>tlSubsystemList</code>	Yes	Cell array with the Simulink paths of the TargetLink subsystems selected for production code generation.
<code>incrSubsystemList</code>	Yes	Cell array with the Simulink paths of the incremental subsystems selected for production code generation.
<code>refModelList</code>	Yes	Cell array with the names of the models selected for production code generation.
<code>ddCodeGenUnitList</code>	Yes	Cell array with the DD paths of the <code>CodeGenerationUnit</code> objects selected for production code generation.
<code>bCodeGenFromDD</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If the hook script was called for DD-based code generation. ▪ <code>0</code> - If the hook script was called for code generation for the model-based code generation unit.
<code>bCodeGenSucceeded</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If code generation was successful for all listed code generation units. ▪ <code>0</code> - If code generation was not successful for all listed code generation units.

Related topics

Basics

Basics on the Code Generation Process ([📖 TargetLink Preparation and Simulation Guide](#))
 Basics on Using Hook Scripts ([📖 TargetLink Customization and Optimization Guide](#))
 Using the TargetLink M-Script Interface (API) ([📖 TargetLink Interoperation and Exchange Guide](#))

References

Create Customization Files Dialog Description ([📖 TargetLink Tool and Utility Reference](#))
 tl_generate_code ([📖 TargetLink API Reference](#))

tl_post_codegen_hook

Purpose

Lets you customize the code generation phase.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([📖 TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([📖 TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_generate_code.m** immediately after the production code generation finished without errors for one of the specified code generation units.

Description


You can enter your own commands to be executed after code generation.


Predefined variables


The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model (only for model-based code generation units).
<code>tlSystem</code>	Yes	Simulink path of the system for which production code was generated (only for model-based code generation units).
<code>modelData</code>	Yes	Options entered in the TargetLink Main Dialog (only for model-based code generation units).
<code>simMode</code>	Yes	Intended simulation mode: <ul style="list-style-type: none"> ▪ <code>none</code> - Pure code generation, no simulation. ▪ <code>TL_CODE_HOST</code> - Software-in-the-Loop (SIL) simulation. ▪ <code>TL_CODE_TARGET</code> - Processor-in-the-Loop (PIL) simulation. ▪ <code>TL_CODE_SFCN</code> - Simulation with a stand-alone S-Function.
<code>ddCodeGenUnit</code>	Yes	Data Dictionary path of the CodeGenerationUnit object for which production code was generated.
<code>tlcgOptions</code>	Yes	Code Generator options used for the code generation. To get a list of supported Code Generator options, type <code>tl_get_supported_cgoptions</code> in the MATLAB Command Window.
<code>tlGeneratedFiles</code>	Yes	List of C and header files generated for the current code generation unit (as cellarrays, with the full file names).
<code>bFirst</code>	Yes	<ul style="list-style-type: none"> ▪ 1 - If code was generated for the first code generation unit. ▪ 0 - If code was not generated for the first code generation unit.
<code>bLast</code>	Yes	<ul style="list-style-type: none"> ▪ 1 - If code was generated for the last code generation unit. ▪ 0 - If code was not generated for the last code generation unit.


Related topics**Basics**

[Basics on the Code Generation Process](#) ( [TargetLink Preparation and Simulation Guide](#))

[Basics on Using Hook Scripts](#) ( [TargetLink Customization and Optimization Guide](#))

[Using the TargetLink M-Script Interface \(API\)](#) ( [TargetLink Interoperation and Exchange Guide](#))

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

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

[tl_pre_codegen_hook](#)..... 63

tl_post_framegen_hook

Purpose Lets you customize the simulation frame generation phase.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_generate_code.m** immediately after the simulation frame for the production code SIL/PIL simulation was generated.


Description You can enter your own commands to be executed after the simulation frame generation.

Predefined variables The following predefined variables are available:


Predefined Variable	Read Only	Description
model	Yes	Name of the Simulink model.
tlSystems	Yes	Name of the model-based code generation units that a simulation frame was generated for.
simMode	Yes	Intended simulation mode: <ul style="list-style-type: none"> ▪ TL_CODE_HOST - Simulation frame was generated for SIL simulation. ▪ TL_CODE_TARGET - Simulation frame was generated for PIL simulation. ▪ TL_CODE_SFCN - Simulation frame was generated for stand-alone S-function.
modelData	Yes	Options entered in the TargetLink Main Dialog.

Related topics

Basics

Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

References


Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))

tl_pre_codegen_hook

Purpose

Lets you customize the code generation phase.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_generate_code.m** immediately before the production code generator is called for one of the specified code generation untis.

Description

You can enter your own commands to be executed before code generation.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model (only for model-based code generation units).
<code>tlSystem</code>	Yes	Simulink path of the system for which production code is to be generated (only for model-based code generation units).
<code>modelData</code>	Yes	Options entered in the TargetLink Main Dialog.
<code>ddCodeGenUnit</code>	Yes	Data Dictionary path of the <code>CodeGenerationUnit</code> object for which production code is to be generated.
<code>simMode</code>	Yes	Intended simulation mode: <ul style="list-style-type: none"> ▪ <code>none</code> - Only production code will be generated. ▪ <code>TL_CODE_HOST</code> - Frame files for the SIL simulation are also generated. ▪ <code>TL_CODE_TARGET</code> - Frame files for the PIL simulation are also generated. ▪ <code>TL_CODE_SFCN</code> - Stand-alone S-function is also generated.
<code>tlcgOptions</code>	No	Code Generator options used for the code generation. You can modify this cell array to customize the code generation. To get a list of supported Code Generator options, enter <code>tl_get_supported_cgoptions</code> in the MATLAB Command Window.
<code>bFirst</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If code is generated for the first code generation units. ▪ <code>0</code> - If code is not generated for the first code generation unit.
<code>bLast</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If code is generated for the last code generation unit. ▪ <code>0</code> - If code is not generated for the last code generation unit.

Remark

This hook script is called immediately before the start of the Code Generator but after the model was initialized by Simulink. Therefore, using API functions that change the Simulink model or block data can lead to unexpected or even incorrect results when production code is generated. Do not use the `tl_pre_codegen_hook` hook script to make changes to the model:

- Changes to the Simulink structure and to Simulink model properties are not supported (e.g., do not delete or add blocks).
- Changes to the signal widths are not supported.
- Changes to the signal labels are not supported.

To modify the model before code generation, you can derive the `tl_pre_codegen_init_hook` hook script from its template.

Example

```
% Append item to list of Code Generator options.
tlcgOptions = { ...
    tlcgOptions{:} ...
    'OmitZeroInitializationsInRestartFunction','on'};

% Suppress right and left shifts....
tlcgOptions = { ...
    tlcgOptions{:} ...
    'ShiftMode', 3};
```

Related topics**Basics**

Basics on the Code Generation Process ([📖 TargetLink Preparation and Simulation Guide](#))

Basics on Using Hook Scripts ([📖 TargetLink Customization and Optimization Guide](#))

Suppression of Shift Operations ([📖 TargetLink Preparation and Simulation Guide](#))

Using the TargetLink M-Script Interface (API) ([📖 TargetLink Interoperation and Exchange Guide](#))

References

Code Generator Options ([📖 TargetLink Model Element Reference](#))

Create Customization Files Dialog Description ([📖 TargetLink Tool and Utility Reference](#))

tl_post_codegen_hook..... 60

tl_pre_codegen_init_hook

Purpose

Lets you customize the code generation phase.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([📖 TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_generate_code.m` one time before the model data needed for the code generation is read and the production code generator is called for the first of the specified model-based code generation units, and one time before the production code generator is called for the first of the specified DD `CodeGenerationUnit` objects.

Description

You can enter your own commands to be executed before code generation.

Predefined variables

The following predefined variables are available:




Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model (only for model-based code generation units).
<code>tlSubsystemList</code>	Yes	Cell array with the Simulink paths of the TargetLink subsystems selected for production code generation.
<code>incrSubsystemList</code>	Yes	Cell array with the Simulink paths of the incremental subsystems selected for production code generation.
<code>refModelList</code>	Yes	Cell array with the names of the of models selected for production code generation.
<code>ddCodeGenUnitList</code>	Yes	Cell array with the DD paths of the <code>CodeGenerationUnit</code> objects selected for production code generation.
<code>bCodeGenFromDD</code>	Yes	<ul style="list-style-type: none"> ▪ <code>1</code> - If the hook script was called for DD-based code generation. ▪ <code>0</code> - If the hook script was called for model-based code generation.

Remark

Necessary modifications of the model code generation units are possible.

Related topics

Basics

Basics on the Code Generation Process ( [TargetLink Preparation and Simulation Guide](#))
 Basics on Using Hook Scripts ( [TargetLink Customization and Optimization Guide](#))
 Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

References

Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))

Documentation Hook Scripts

Where to go from here

Information in this section


[tl_autodoc_hook](#)..... 67
[tl_post_autodoc_filter_hook](#)..... 71

tl_autodoc_hook

Purpose

Lets you insert your own text and images into the generated HTML documentation.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is called at various points during automatic document generation.

Description

The HTML document is structured according to the entry points as shown below. Each entry point, e.g., `Create(Entry)`, is a potential insertion point for your text. Whenever an entry point is reached during the document generation process, the `tl_autodoc_hook.m` hook script is called with predefined variables that describe the current chapter and section of the documentation. Depending on this data, you can use HTML commands to insert your own text and images into the document. The following list shows the available entry points for hook scripts [chapter(section)]: `Create(Entry)`

```
Overview(Entry)
Overview(ModelImage)
Overview(ModelComment)
Overview(Exit)
TlSubsystems(Entry)
/ TlSubsystems(TlSubsystemEntry) ... for all code generation
units
| TlSubsystems(TlSubsystemImage)
| TlSubsystems(TlSubsystemInfo)
| TlSubsystems(FunctionHierarchy)
| TlSubsystems(SourceFiles)
| TlSubsystems(BlockStatistics)
| TlSubsystems(TypeInfo)
| / TlSubsystems(FunctionEntry) ... for all functions
| | TlSubsystems(FunctionImage)
| | TlSubsystems(FunctionComment)
| | TlSubsystems(FunctionInterface)
| | TlSubsystems(MeasurableVariables)
| | TlSubsystems(CalibratableVariables)
| | TlSubsystems(MacroVariables)
| | TlSubsystems(OtherVariables)
| | TlSubsystems(AtomicSubsystems)
| | TlSubsystems(1DTables)
| | TlSubsystems(2DTables)
| | TlSubsystems(InterpolationBlocks)
| | TlSubsystems(IndexSearchBlocks)
| \ TlSubsystems(FunctionExit)
\ TlSubsystems(TlSubsystemExit)
TlSubsystems(AllStateMachines)
```

```

/ TlSubsystems(StateMachineEntry) ... for all state machines
| TlSubsystems(StateMachineHierarchy)
| / TlSubsystems(StateChartEntry) ... for all state charts
| | TlSubsystems(StateChartInfo)
| \ TlSubsystems(StateChartExit)
\ TlSubsystems(StateMachineExit)
TlSubsystems(Exit)
SimulationResults(Entry)
SimulationResults(ListOfSimulations)
/ SimulationResults(SimulationEntry) ... for each simulation
\ SimulationResults(SimulationExit)
< SimulationResults(Signal) ... for each simulated signal
SimulationResults(Exit)
Close(Exit)

```

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
options	Yes	Data structure containing information about the current state of the documentation generator.
fid	Yes	Returns the file handle of the document.
currentChapter	Yes	Returns the current chapter in the document.
currentSection	Yes	Returns the current section in the document.
currentItem	Yes	Returns the currently documented item.
currentSystem	Yes	Returns the Simulink path of the currently documented system. This can be a cell array for reused functions.

Remark

You can add your own commands, e.g., to insert a link into the generated HTML document.

Example

```
% Suppose tl_autodoc_hook.m contains the following code. When the
% Create(Entry) entry point is reached during document generation, a
% header is added to the document.


topic = [options.currentChapter '(' options.currentSection ')'];
switch (topic)


    case 'Create(Entry)'
        % Print header at the beginning of the document.
        dsdoc(options.fid,'Printf','Bold','on','Color','r',...
            'String',sprintf('Hook script %s: %s\n',mfilename,topic));
        if exist('tldoc_header.m','file')
            tldoc_header(options.fid, options.modelName,'dSPACE
GmbH','dSPACE.bmp');
            if tl_error_check, dsdoc(fid,'Close'); return; end
        end

    otherwise
        % Show where hook scripts are called.
        dsdoc(options.fid,'Printf','Bold','on','Color','r',...
            'String',sprintf('Hook script %s: %s\n',mfilename,topic));
        dsdoc(options.fid,'Printf',...
            'String',sprintf('Item: %s\n',options.currentItem));
        if iscell(options.currentSystem)
            for i = 1:length(options.currentSystem)
                dsdoc(options.fid,'Printf',...
                    'String',sprintf('System%d:
%s\n',i,options.currentSystem{i}));
            end
        else
            dsdoc(options.fid,'Printf',...
                'String',sprintf('System: %s\n',options.currentSystem));
        end
    end

end
```

Related topics**Basics**

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

[Using the TargetLink M-Script Interface \(API\)](#) ( TargetLink Interoperation and Exchange Guide)


References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[tldoc](#) ( TargetLink API Reference)

tl_post_autodoc_filter_hook

Purpose Lets you enter your own commands, e.g., to filter the list of the found Autodoc Customization blocks.


Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Description This M script is called during automatic documentation generation, after TargetLink obtained the list and the data of the TargetLink Autodoc Customization blocks and applied the filter that you specified via the AutoDocFilter property of the dsdoc command. It is called once for the root model and once for each referenced model, if applicable.

You can enter your own commands, e.g., to influence the list of the found Autodoc Customization blocks.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>filteredAutoDocBlocksInfo</code>	No	<p>Structure containing information about the TargetLink Autodoc Customization blocks found in the current model that satisfy the filter criterion specified by the <code>AutoDocFilter</code> property of the <code>dsdoc</code> command. If no filter is specified, it is equal to <code>rawAutoDocBlocksInfo</code>.</p> <p>With the exception of <code>modelName</code>, each component in the structure is a list, where each list element corresponds to one TargetLink Autodoc Customization block. A detailed description of each component is listed below.</p> <p>Note: If you want to remove a TargetLink Autodoc Customization block, you have to remove the corresponding entry in each list.</p>
<code>modelName</code>	Yes	Name of the model where the found TargetLink Autodoc Customization blocks reside.
<code>blockDataList</code>	No	Vector of structures where each vector element contains data of one Autodoc Customization block.
<code>autodocOwnerList</code>	No	<p>Vector of structures where each structure element contains the scope information about one Autodoc Customization block:</p> <ul style="list-style-type: none"> ▪ <code>.fcnSubsystemPath</code> - Simulink path of the function subsystem that the Autodoc Customization block resides in. It is not always the parent system of the Autodoc Customization block, e.g., if it is placed in a virtual subsystem. ▪ <code>.atomicSubsystemPath</code> - Simulink path of the atomic subsystem that the Autodoc Customization block resides in. It is not always the parent system of the Autodoc Customization block, e.g., if it is placed in a virtual subsystem.
<code>hooksLocationList</code>	No	Cell array where each cell contains information about the location in the generated documentation (documentation placement) that is to be influenced by the corresponding Autodoc Customization block.
<code>printedFlagList</code>	No	<p>Double array where each element indicates whether the corresponding Autodoc Customization block was already evaluated. Possible values:</p> <p><code>1</code> - The block was already evaluated.</p> <p><code>0</code> - The block was not evaluated.</p>
<code>rawAutoDocBlocksInfo</code>	Yes	Structure containing information about all TargetLink Autodoc Customization blocks found in the model, regardless of whether they satisfy the specified filter criterion. It contains the same components as the <code>filteredAutoDocBlocksInfo</code> structure, but all of them are read-only.

Example

```
% Example 1: Overwrite existing filter, if applicable.
filteredAutoDocBlocksInfo = rawAutoDocBlocksInfo;

% Example 2: From the list of all Autodoc Customization blocks, take
% only those whose Tag property is set to 'Extern Documentation'.
% Remove all other blocks. Overwrite the filter specified by the
% AutoDocFilter property, if applicable.
selectedTag = 'Extern Documentation';
toBeRemovedBlockIdxList =
find(strcmp({filteredAutoDocBlocksInfo.blockDataList.tag},selectedTa
g) == 0);

% Delete the elements with the index in toBeRemovedBlockIdxList from
% all list components:
filteredAutoDocBlocksInfo.blockDataList(toBeRemovedBlockIdxList) =
[];
filteredAutoDocBlocksInfo.autodocOwnerList(toBeRemovedBlockIdxList)
= [];
filteredAutoDocBlocksInfo.hooksLocationList(toBeRemovedBlockIdxList)
= [];
filteredAutoDocBlocksInfo.printedFlagList(toBeRemovedBlockIdxList) =
[];
```

Related topics**Basics**

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

[Using the TargetLink M-Script Interface \(API\)](#) ([TargetLink Interoperation and Exchange Guide](#))

References

[Create Customization Files Dialog Description](#) ([TargetLink Tool and Utility Reference](#))

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

Export Hook Scripts

Where to go from here**Information in this section**

tl_post_export_files_hook	74
tl_pre_export_files_hook	75

tl_post_export_files_hook

Purpose Lets you customize the file export.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_export_files.m** immediately after the files were copied.

Description You can enter your own commands to be executed after the files were copied.

Predefined variables The following predefined variables are available:


Predefined Variable	Read Only	Description
fileList	Yes	List of all exported files.
srcName	Yes	Name of the source file.
srcDir	Yes	Folder the source file is copied from.
destName	Yes	Name of the destination file.
destDir	Yes	Name of the destination folder.
keyword	Yes	Description of the file contents.

Remark Do not modify the fileList variable in this hook script.

Example

```
% This example displays a list of all the exported files.
fprintf('%s\\%s\\n', fileList.destDir, fileList.destName);
```

Related topics**Basics**

[Using the TargetLink M-Script Interface \(API\)](#) ( [TargetLink Interoperation and Exchange Guide](#))

HowTos

[How to Export Generated Files from the TargetLink Environment](#) ( [TargetLink Interoperation and Exchange Guide](#))

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

[File Export Utility](#) ( [TargetLink Tool and Utility Reference](#))


tl_pre_export_files_hook..... 75

tl_pre_export_files_hook

Purpose

Lets you customize the file export.

Access


You can access customization files via the **Create Customization Files** dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by `tl_export_files.m` immediately before the files are copied.

Description You can enter your own commands to execute before the files are copied.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>fileList</code>	No	List of all files to be exported. You can modify this list as needed by appending files, removing files, or changing destination file names.
<code>srcName</code>	No	Name of the source file.
<code>srcDir</code>	No	Folder the source file is copied from.
<code>destName</code>	No	Name of the destination file.
<code>destDir</code>	No	Name of the destination folder.
<code>keyword</code>	No	Description of the file contents.


Example

```
% To call the
% 'copy .\controller.c .\export\controller.c' command
% specify the following fileList entries:

fileList.srcDir = '.';
fileList.srcName = 'controller.c';
fileList.destDir = '.\export';
fileList.destName = 'controller.c';
fileList.keyword = 'controller';
```

Related topics


Basics

[Using the TargetLink M-Script Interface \(API\)](#) ( TargetLink Interoperation and Exchange Guide)

HowTos

[How to Export Generated Files from the TargetLink Environment](#) ( TargetLink Interoperation and Exchange Guide)

References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[File Export Utility](#) ( TargetLink Tool and Utility Reference)

[tl_post_export_files_hook](#)..... 74

Frame Model Generation Hook Scripts (AUTOSAR)

Where to go from here

Information in this section

tl_post_add_arportblock_hook.....	77
tl_post_add_comspecblock_hook.....	84
tl_post_add_operationcallsubsystem_hook.....	85
tl_post_add_operationresultprovidersubsystem_hook.....	87
tl_post_add_runnablesubsystem_hook.....	89
tl_post_add_swcsubsystem_hook.....	90
tl_post_swcmodelegen_hook.....	93
tl_pre_add_arportblock_hook.....	95
tl_pre_add_comspecblock_hook.....	102

tl_post_add_arportblock_hook

Purpose

Lets you customize AUTOSAR frame model generation.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_generate_sw_model.m` after a TargetLink port block(Inport, Outport, BusInport or BusOutport) block that represents one of the following was added:

- Data read/write access (`runnablePortKind == 1`)
- Data receive/send point (`runnablePortKind == 2`)
- Interrunnable variable (`runnablePortKind == 3`)
- Operation call (`runnablePortKind == 4`)
- Operation argument within a server runnable (`runnablePortKind == 5`, `operationCallSubsystemPath` empty)
- Operation argument within an operation call subsystem (`runnablePortKind == 5`, `operationCallSubsystemPath` set)
- Application error (`runnablePortKind == 6`)
- Mode access/mode switch point (`runnablePortKind == 7`)
- Activation reasons argument (`runnablePortKind == 8`)
- Port-defined argument (`runnablePortKind == 9`)
- NvData read access, NvData write access(`runnablePortKind == 10`)
- Transformer error (`runnablePortKind == 11`)

Description

You can enter your own commands to be executed after one of the following port blocks is added to the frame model:

- Inport
- Outport
- Bus Inport
- Bus Outport

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
runnablePortInfo	Yes	Structure describing the current port.
swcRef	Yes	Reference to the DD SoftwareComponent object.
portRef	Yes	Reference to the DD object that represents the port, e.g., a DD SenderPort or ClientPort object.
dataElementRef	Yes	Reference to the DD DataElement object.
modeElementRef	Yes	Reference to the DD ModeElement object.
variableRef	Yes	Reference to the DD Variable object.
operationRef	Yes	Reference to the DD Operation object.
operationArgumentRef	Yes	Reference to the DD OperationArgument object.
runnablePortKind	Yes	Specifies one of the following runnable port kinds: <ul style="list-style-type: none"> ▪ 1 - Data read access, data write access ▪ 2 - Data receive point, data send point ▪ 3 - Interrunnable variable read access, interrunnable variable write access ▪ 4 - Synchronous server call point ▪ 5 - Server runnable operation argument, operation call subsystem operation argument ▪ 6 - Application error ▪ 7 - Mode switch point, mode access point ▪ 8 - Activation reasons argument ▪ 9 - Port-defined argument ▪ 10 - NvData write access, NvData read access ▪ 11 - Transformer error
runnablePortName	Yes	Name of the TargetLink port that represents the runnable port.
runnableSubsystemPath	Yes	Simulink path of the subsystem that represents the runnable.
operationSubsystemPath	Yes	Simulink path of the operation subsystem.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDPort	Yes	Handle of the DD object that represents the port, e.g., a DD SenderPort or ClientPort object.
hDDDataElement	Yes	Handle of the DD DataElement object.
hDDModeElement	Yes	Handle of the DD ModeElement object.
hDDIrvVariable	Yes	Handle of the DD Variable object.
hDDOperation	Yes	Handle of the DD Operation object.
hDDOperationArgument	Yes	Handle of the DD OperationArgument object.
hDDArgInList	Yes	Vector of handles of the DD OperationArgument objects whose Kind property is set to ARGIN .
hDDArgOutList	Yes	Vector of handles of the DD OperationArgument objects whose Kind property is set to ARGOUT .
t1BlockType	Yes	Type of the TargetLink block that represents the port (TL_Inport, TL_Outport, TL_BusInport, TL_BusOutport).
s1BlockPath	Yes	Simulink path of the inserted TargetLink port.
connectionPort	Yes	Runnable subsystem port that matches the port block inserted by TargetLink.
hDDPortDefinedArgument	Yes	Handle of the DD PortDefinedArgument object.

Predefined Variable	Read Only	Description
portDefinedArgumentRef	Yes	Reference to the DD PortDefinedArgument object.
hDDActivationReasonList	Yes	Vector of handles of the DD ActivationReason objects.
hDDNvDataElement	Yes	Handle of the DD NvDataElement object.
nvDataElementRef	Yes	Reference to the DD NvDataElement object.

Conditions

Condition	Related Variables	Description
runnablePortKind = 1	<ul style="list-style-type: none"> ▪ runnablePortInfo ▪ swcRef ▪ runnablePortKind ▪ runnablePortName ▪ runnableSubsystemPath ▪ tlBlockType ▪ slBlockPath ▪ connectionPort ▪ portRef ▪ dataElementRef ▪ hDDPort ▪ hDDDataElement ▪ hDDSwc 	Data read access, data write access.
runnablePortKind = 2	<ul style="list-style-type: none"> ▪ runnablePortInfo ▪ swcRef ▪ runnablePortKind ▪ runnablePortName ▪ runnableSubsystemPath ▪ tlBlockType ▪ slBlockPath ▪ connectionPort ▪ portRef ▪ dataElementRef ▪ hDDPort ▪ hDDDataElement ▪ hDDSwc 	Data receive point, data send point.
runnablePortKind = 3	<ul style="list-style-type: none"> ▪ runnablePortInfo ▪ swcRef ▪ runnablePortKind ▪ runnablePortName ▪ runnableSubsystemPath ▪ tlBlockType ▪ slBlockPath ▪ connectionPort ▪ hDDIrVariable ▪ hDDSwc ▪ variableRef 	Interrunnable variable read access, interrunnable variable write access.

Condition	Related Variables	Description
<code>runnablePortKind = 4</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef runnablePortKind runnablePortName runnableSubsystemPath tlBlockType slBlockPath connectionPort portRef operationRef hDDPort hDDOperation hDDArgInList hDDArgOutList hDDSwc 	Synchronous server call point.
<code>runnablePortKind = 5</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef runnablePortKind runnablePortName runnableSubsystemPath tlBlockType slBlockPath connectionPort portRef operationRef operationArgumentRef hDDPort hDDOperation hDDOperationArgument hDDSwc 	Server-runnable operation argument.
<code>runnablePortKind = 5</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef runnablePortKind runnablePortName runnableSubsystemPath tlBlockType slBlockPath connectionPort portRef operationRef operationArgumentRef hDDPort hDDOperation hDDOperationArgument operationSubsystemPath hDDSwc 	Operation call subsystem operation argument.

Condition	Related Variables	Description
<code>runnablePortKind = 6</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef runnablePortKind runnablePortName runnableSubsystemPath tlBlockType slBlockPath connectionPort portRef hDDSwc operationRef hDDOperation hDDPort 	Application error.
<code>runnablePortKind = 7</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef portRef runnablePortKind runnablePortName runnableSubsystemPath hDDSwc hDDPort tlBlockType slBlockPath connectionPort hDDModeElement modeElementRef 	Mode switch point, mode access point.
<code>runnablePortKind = 8</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef runnablePortKind runnablePortName runnableSubsystemPath hDDSwc tlBlockType slBlockPath connectionPort hDDActivationReasonList 	Activation reasons argument.
<code>runnablePortKind = 9</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef portRef hDDSwc hDDPort tlBlockType slBlockPath connectionPort hDDPortDefinedArgument portDefinedArgumentRef 	Port-defined argument.

Condition	Related Variables	Description
<code>runnablePortKind = 10</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>hDDSwc</code> ▪ <code>hDDPort</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>hDDNvDataElement</code> ▪ <code>nvDataElementRef</code> ▪ <code>connectionPort</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> 	NvData read access, NvData receive point, NvData write access, NvData send point.
<code>runnablePortKind = 11</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>operationRef</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> ▪ <code>hDDSwc</code> ▪ <code>hDDPort</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>connectionPort</code> ▪ <code>hDDOperation</code> 	Transformer error.

Remark

The fields of the `runnablePortInfo` struct are set according to the structure's `runnablePortKind` field. Only the following fields are always available:

- `swcRef`
- `runnablePortKind`
- `runnablePortName`
- `runnableSubsystemPath`
- `tlBlockType`
- `slBlockPath`
- `connectionPort`

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)
[tl_pre_add_arportblock_hook..... 95](#)

tl_post_add_comspecblock_hook

Purpose Lets you customize AUTOSAR frame model generation/update.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_generate_swc_model.m** after a ReceiverComSpec or SenderComSpec block was added.

Description You can enter your own commands to be executed after a SenderComSpec/ReceiverComSpec block was added to the model.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
hDDComSpecObject	Yes	Handle of the DD object representing the communication specification: <ul style="list-style-type: none"> ▪ DD DataReceiverComSpec object ▪ DD EventReceiverComSpec object ▪ DD DataSenderComSpec object ▪ DD EventSenderComSpec object
hComSpecBlock	Yes	Simulink handle of the added SenderComSpec/ReceiverComSpec block.
hRunnablePort	Yes	Simulink handle of the [Bus]Output/[Bus]Inport block that the SenderComSpec/ReceiverComSpec block is to be connected to.

Related topics

References

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)
 tl_pre_add_comspecblock_hook..... 102

tl_post_add_operationcallsubsystem_hook

Purpose

Lets you customize AUTOSAR frame model generation.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** (📖 TargetLink API Reference) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) (📖 TargetLink Customization and Optimization Guide).

Hook point

This M script is evaluated by **tl_generate_swc_model.m** after the generation of the operation call subsystem within a runnable subsystem was finished.

Description

You can enter your own commands to be executed after an operation call subsystem was added to the frame model.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
operationSubsystemInfo	Yes	Structure describing an operation call subsystem.
operationSubsystemPath	Yes	Simulink path of the operation call subsystem that was added to the frame model.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDClientPort	Yes	Handle of the DD ClientPort object.
hDDOperation	Yes	Handle of the DD Operation object.
argInConnectionList	Yes	Vector of structures describing all the input arguments of the operation. If the operation has no input arguments, the list is empty.
swcRef	Yes	Reference to the DD SoftwareComponent object.
portRef	Yes	Reference to the ClientPort DD object that represents the port.
operationRef	Yes	Reference to the DD Operation object.
operationArgumentRef	Yes	Reference to the DD OperationArgument object.
runnableSubsystemPath	Yes	Simulink path of the subsystem that represents the runnable.
operationSubsystemPath	Yes	Simulink path of the operation call subsystem.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDPort	Yes	Handle of the DD port object.
hDDOperation	Yes	Handle of the DD Operation object.
hDDOperationArgument	Yes	Handle of the DD OperationArgument object.
tlBlockType	Yes	Type of the TargetLink block that represents the operation argument (TL_Inport, TL_Outport, TL_BusInport, TL_BusOutport).
s1BlockPath	Yes	Simulink path of the port block that was added.
argOutConnectionList	Yes	Vector of structures describing all the output arguments of the operation. If the operation has no output arguments, the list is empty.
swcRef	Yes	Reference to the DD SoftwareComponent object.
portRef	Yes	Reference to the ClientPort DD object that represents the port.
operationRef	Yes	Reference to the DD Operation object.
operationArgumentRef	Yes	Reference to the DD OperationArgument object.
runnableSubsystemPath	Yes	Simulink path of the subsystem that represents the runnable.
operationSubsystemPath	Yes	Simulink path of the operation call subsystem.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDPort	Yes	Handle of the DD port object.
hDDOperation	Yes	Handle of the DD Operation object.
hDDOperationArgument	Yes	Handle of the DD OperationArgument object.
tlBlockType	Yes	Type of the TargetLink block that represents the operation argument (TL_Inport, TL_Outport, TL_BusInport, TL_BusOutport).
s1BlockPath	Yes	Simulink path of the port block that was added.

Related topics

References

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)
 tl_post_add_operationresultprovidersubsystem_hook..... 87

tl_post_add_operationresultprovidersubsystem_hook

Purpose

Lets you customize AUTOSAR frame model generation.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** (📖 TargetLink API Reference) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) (📖 TargetLink Customization and Optimization Guide).

Hook point

This M script is evaluated by **tl_generate_swc_model.m** after the generation of the operation result provider subsystem within a runnable subsystem was finished.

Description

You can enter your own commands to be executed after an operation result provider subsystem was added to the frame model.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
operationSubsystemInfo	Yes	Structure describing an operation result provider subsystem.
operationSubsystemPath	Yes	Simulink path of the operation result provider subsystem that was added to the frame model.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDClientPort	Yes	Handle of the DD ClientPort object.
hDDOperation	Yes	Handle of the DD Operation object.
argInConnectionList	Yes	An empty structure. Operation input arguments are not modeled within an operation result provider subsystem.
argOutConnectionList	Yes	Vector of structures describing all the output arguments of the operation. If the operation has no output arguments, the list is empty.
swcRef	Yes	Reference to the DD SoftwareComponent object.
portRef	Yes	Reference to the ClientPort DD object that represents the port.
operationRef	Yes	Reference to the DD Operation object.
operationArgumentRef	Yes	Reference to the DD OperationArgument object.
runnableSubsystemPath	Yes	Simulink path of the subsystem that represents the runnable.
operationSubsystemPath	Yes	Simulink path of the operation result provider subsystem.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDPort	Yes	Handle of the DD port object.
hDDOperation	Yes	Handle of the DD Operation object.
hDDOperationArgument	Yes	Handle of the DD OperationArgument object.
tlBlockType	Yes	Type of the TargetLink block that represents the operation argument (TL_Inport, TL_Outport, TL_BusInport, TL_BusOutport).
s1BlockPath	Yes	Simulink path of the port block that was added.
hDDAccessPoint	Yes	Handle of a DD AsynchronousServerCallResultPoint object (Autosar 4.x) or AsynchronousServerCallResults RteEvent object (Autosar 4.x).

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)

tl_post_add_operationcallsubsystem_hook..... 85

tl_post_add_runnablesubsystem_hook

Purpose	Lets you customize AUTOSAR frame model generation.
Access	<p>You can access customization files via the Create Customization Files dialog that opens if you enter tlCustomizationFiles (📖 TargetLink API Reference) in the MATLAB Command Window.</p> <p>TargetLink performs the following steps:</p> <ul style="list-style-type: none">▪ Copies the customization file template to your current working directory.▪ Derives the functional customization file from its template.▪ Opens the customization file in the editor. <div>Note<p>Customization files must reside on the MATLAB <i>and</i> TargetLink search paths.</p><p>For instructions, refer to Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators (📖 TargetLink Customization and Optimization Guide).</p></div>
Hook point	This M script is evaluated by tl_generate_sw_model.m after the generation of the runnable subsystem within a software component subsystem was finished.
Description	You can enter your own commands to be executed after a runnable subsystem was added to the frame model.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
runnableSubsystemInfo	Yes	<code>runnableSubsystemInfo</code> structure describing a runnable subsystem.
runnableSubsystemPath	Yes	Simulink path of the subsystem that represents the runnable.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDRRunnable	Yes	Handle of the DD Runnable object.
runnableInPortInfoList	Yes	Vector of <code>runnablePortInfo</code> structures describing the inport blocks of the runnable subsystem.
runnableOutPortInfoList	Yes	Vector of <code>runnablePortInfo</code> structures describing the outport blocks of the runnable subsystem.
operationSubsystemInfoList	Yes	Vector of <code>operationSubsystemInfo</code> structures describing the operation subsystems of the runnable subsystem.
swcRef	Yes	Reference to the DD SoftwareComponent object.
runnableRef	Yes	Reference to the DD Runnable object.

Remark

For more information on `runnablePortInfo` structure, refer to `tl_pre_add_arportblock_hook` or `tl_post_add_arportblock_hook`. For more information on `operationSubsystemInfo` structure, refer to `tl_post_add_operationcallsubsystem_hook`.

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)	
<code>tl_post_add_arportblock_hook</code>	77
<code>tl_post_add_operationcallsubsystem_hook</code>	85
<code>tl_post_add_operationresultprovidersubsystem_hook</code>	87
<code>tl_pre_add_arportblock_hook</code>	95

tl_post_add_swcsubsystem_hook

Purpose

Lets you customize AUTOSAR frame model generation.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** (📖 TargetLink API Reference) in the MATLAB Command Window.


TargetLink performs the following steps:

- Copies the customization file template to your current working directory.

- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_generate_swc_model.m` after the generation of the single software component subsystem was finished.

Description

You can enter your own commands to be executed after an SWC subsystem was added to the frame model.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
swcSubsystemInfo	Yes	Structure describing a SWC subsystem.
swcSubsystemPath	Yes	Simulink path of the subsystem that was added to the frame model.
hDDSwc	Yes	Handle of the related DD SoftwareComponent object
bIsTLSubsystem	Yes	Depends on the value of the <code>UseOneTLSubsystemForAllSwcs</code> property of the <code>tl_generate_swc_model</code> API command: <ul style="list-style-type: none"> ▪ 0 - The SWC subsystem is not a TargetLink subsystem. ▪ 1 - The SWC subsystem is a TargetLink subsystem.
swcReceiverPortInfoList	Yes	Vector of runnablePortInfo structures describing all the SWC receiver ports of the software component.
swcSenderPortInfoList	Yes	Vector of runnablePortInfo structures describing all the SWC sender ports of the software component.
swcInportInfoList	Yes	Vector of runnablePortInfo structures describing the following: <ul style="list-style-type: none"> ▪ SWC ReceiverPort blocks for ports with a sender-receiver interface that has one data element. ▪ Simulink ports that are directly connected with runnable subsystem's inports, for example with ports that models a Get operation. ▪ BusSelector block output corresponding to one data element carried on the bus. Placed on the right side of an SWC SenderPort block with a sender-receiver interface that has more than one data element.
swcOutportInfoList	Yes	Vector of runnablePortInfo structures describing the following: <ul style="list-style-type: none"> ▪ SWC SenderPort blocks for ports with a sender-receiver interface that has one data element. ▪ Simulink ports that are directly connected with a runnable outputs, for example with ports that models a Set operation. ▪ BusCreator block inport corresponding to one data element carried on the bus. Placed on the left side of an SWC SenderPort block with a sender-receiver interface that has more than one data element.
runnableSubsystemInfoList	Yes	Vector of runnableSubsystemInfo structures describing all the runnables of the software component.

Remark

For more information on **runnablePortInfo** structure, refer to `tl_pre_add_arportblock_hook`. For more information on **runnableSubsystemInfo** structure, refer to `tl_post_add_runnablesubsystem_hook`.

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)	
<code>tl_post_add_runnablesubsystem_hook</code>	89
<code>tl_pre_add_arportblock_hook</code>	95

tl_post_swcmodeigen_hook

Purpose

Lets you customize AUTOSAR frame model generation.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_generate_swc_model.m** after the software component model generation was finished successfully, i.e., after the new TargetLink subsystem was created for the specified software components and copied, to the specified destination subsystem, if applicable.

Description

You can enter your own commands to be executed after a frame model was generated.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
model	Yes	Name of the Simulink model.
SWCNameList	Yes	List of the model's software components.
bOneTLSubsystemForAllSWC	Yes	The value of the UseOneTLSubsystemForAllSwcs property of the executed tl_generate_swc_model command: <ul style="list-style-type: none"> ▪ 1 - If software components were modeled in subsystems of one TargetLink subsystem. ▪ 0 - If software components were modeled in separate TargetLink subsystems.
hTLSubsystem	Yes	Depends on bOneTLSubsystemForAllSWC.
hTLSubsystemList	Yes	Depends on bOneTLSubsystemForAllSWC.

Predefined Variable	Read Only	Description
swcSubsystemInfoList	Yes	Vector of swcSubsystemInfo structures describing all the model's software components.

Conditions

Condition	Related Variables	Description
bOneTLSubsystemForAllSWC equals '1'	<ul style="list-style-type: none"> hTlSubsystem hTlSubsystemList 	Software components were modeled in virtual subsystems of one TargetLink subsystem.
bOneTLSubsystemForAllSWC equals '0'	<ul style="list-style-type: none"> hTlSubsystem hTlSubsystemList 	Software components were modeled in separate TargetLink subsystems.

Remark

For more information on swcSubsystemInfo structure, refer to `tl_post_add_swcsystem_hook`.

Example

```
fprintf('Name of the generated model: %s\n', model);
fprintf('Software component(s):\n');
for m = 1:numel(SWCNameList)
    fprintf('    %s\n', SWCNameList{m});
end
if bOneTLSubsystemForAllSWC
    fprintf('TL subsystem: %s\n', getfullname(hTlSubsystem));
else
    fprintf('TL subsystem(s):\n');
    for m = 1:numel(hTlSubsystemList)
        fprintf('    %s\n', getfullname(hTlSubsystemList(m)));
    end
end
```

Related topics

References

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)

`tl_post_add_swcsystem_hook`..... 90

tl_pre_add_arportblock_hook

Purpose

Lets you customize AUTOSAR frame model generation.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([📖 TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([📖 TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_generate_swc_model.m** before a TargetLink port block (Inport, Output, BusInport or BusOutport) block that represents one of the following is added:

- Data read/write access (runnablePortKind == 1)
- Data receive/send point (runnablePortKind == 2)
- Interrunnable variable (runnablePortKind == 3)
- Operation call (runnablePortKind == 4)
- Operation argument within a server runnable (runnablePortKind == 5, operationCallSubsystemPath empty)
- Operation argument within an operation call subsystem (runnablePortKind == 5, operationCallSubsystemPath set)
- Application error (runnablePortKind == 6)
- Mode access/mode switch point (runnablePortKind == 7)
- Activation reasons argument (runnablePortKind == 8)
- Port-defined argument (runnablePortKind == 9)
- NvData read access, NvData write access (runnablePortKind == 10)
- Transformer error (runnablePortKind == 11)

Description

You can enter your own commands to be executed before one of the following port blocks is added to the frame model:

- Inport
- Outport
- Bus Inport
- Bus Outport

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
runnablePortInfo	Yes	Structure describing the current port.
swcRef	Yes	Reference to the DD SoftwareComponent object.
portRef	Yes	Reference to the DD object that represents the port, e.g., a DD SenderPort or ClientPort object.
dataElementRef	Yes	Reference to the DD DataElement object.
modeElementRef	Yes	Reference to the DD ModeElement object.
variableRef	Yes	Reference to the DD Variable object.
operationRef	Yes	Reference to the DD Operation object.
operationArgumentRef	Yes	Reference to the DD OperationArgument object.
runnablePortKind	Yes	Specifies one of the following runnable port kinds: <ul style="list-style-type: none"> ▪ 1 - Data read access, data write access ▪ 2 - Data receive point, data send point ▪ 3 - Interrunnable variable read access, interrunnable variable write access ▪ 4 - Synchronous server call point ▪ 5 - Server runnable operation argument, operation call subsystem operation argument ▪ 6 - Application error ▪ 7 - Mode switch point, mode access point ▪ 8 - Activation reasons argument ▪ 9 - Port-defined argument ▪ 10 - NvData write access, NvData read access ▪ 11 - Transformer error
runnablePortName	Yes	Name of the TargetLink port that represents the runnable port.
runnableSubsystemPath	Yes	Simulink path of the subsystem that represents the runnable.
operationSubsystemPath	Yes	Simulink path of the operation subsystem.
hDDSwc	Yes	Handle of the DD SoftwareComponent object.
hDDPort	Yes	Handle of the DD object that represents the port, e.g., a DD SenderPort or ClientPort object.
hDDDataElement	Yes	Handle of the DD DataElement object.
hDDModeElement	Yes	Handle of the DD ModeElement object.
hDDIrvVariable	Yes	Handle of the DD Variable object.
hDDOperation	Yes	Handle of the DD Operation object.
hDDOperationArgument	Yes	Handle of the DD OperationArgument object.
hDDArgInList	Yes	Vector of handles of the DD OperationArgument objects whose Kind property is set to ARGIN .
hDDArgOutList	Yes	Vector of handles of the DD OperationArgument objects whose Kind property is set to ARGOUT .
t1BlockType	Yes	Type of the TargetLink block that represents the port (TL_Inport, TL_Outport, TL_BusInport, TL_BusOutport).
s1BlockPath	Yes	Simulink path of the inserted TargetLink port.
connectionPort	Yes	Runnable subsystem port that matches the port block inserted by TargetLink.
hDDActivationReasonList	Yes	Vector of handles of the DD ActivationReason objects.

Predefined Variable	Read Only	Description
hDDPortDefinedArgument	Yes	Handle of the DD PortDefinedArgument object.
hDDNvDataElement	Yes	Handle of the DD NvDataElement object.
nvDataElementRef	Yes	Reference to the DD NvDataElement object.
portDefinedArgumentRef	Yes	Reference to the DD PortDefinedArgument object.

Conditions

Condition	Related Variables	Description
runnablePortKind = 1	<ul style="list-style-type: none"> ▪ runnablePortInfo ▪ swcRef ▪ portRef ▪ dataElementRef ▪ runnablePortKind ▪ runnablePortName ▪ runnableSubsystemPath ▪ hDDSwc ▪ hDDPort ▪ hDDDataElement ▪ tlBlockType ▪ slBlockPath ▪ connectionPort 	Data read access, data write access.
runnablePortKind = 2	<ul style="list-style-type: none"> ▪ runnablePortInfo ▪ swcRef ▪ portRef ▪ dataElementRef ▪ runnablePortKind ▪ runnablePortName ▪ runnableSubsystemPath ▪ hDDSwc ▪ hDDPort ▪ hDDDataElement ▪ tlBlockType ▪ slBlockPath ▪ connectionPort 	Data receive point, data send point.
runnablePortKind = 3	<ul style="list-style-type: none"> ▪ runnablePortInfo ▪ swcRef ▪ variableRef ▪ runnablePortKind ▪ runnablePortName ▪ runnableSubsystemPath ▪ hDDSwc ▪ hDDIrvVariable ▪ tlBlockType ▪ slBlockPath ▪ connectionPort 	Interrunnable variable read access, interrunnable variable write access.

Condition	Related Variables	Description
<code>runnablePortKind = 4</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>operationRef</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> ▪ <code>hDDPort</code> ▪ <code>hDDSwc</code> ▪ <code>hDDOperation</code> ▪ <code>hDDArgInList</code> ▪ <code>hDDArgOutList</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>connectionPort</code> 	Synchronous server call point.
<code>runnablePortKind = 5</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>operationRef</code> ▪ <code>operationArgumentRef</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> ▪ <code>hDDSwc</code> ▪ <code>hDDPort</code> ▪ <code>hDDOperation</code> ▪ <code>hDDOperationArgument</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>connectionPort</code> 	Server runnable operation argument.
<code>runnablePortKind = 5</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>operationRef</code> ▪ <code>operationArgumentRef</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> ▪ <code>operationSubsystemPath</code> ▪ <code>hDDSwc</code> ▪ <code>hDDPort</code> ▪ <code>hDDOperation</code> ▪ <code>hDDOperationArgument</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>connectionPort</code> 	Operation call subsystem operation argument.

Condition	Related Variables	Description
<code>runnablePortKind = 6</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef portRef operationRef runnablePortKind runnablePortName runnableSubsystemPath hDDSwc hDDPort hDDOperation tlBlockType slBlockPath connectionPort 	Application error.
<code>runnablePortKind = 7</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef portRef modeElementRef runnablePortKind runnablePortName runnableSubsystemPath hDDModeElement tlBlockType slBlockPath connectionPort hDDSwc hDDPort 	Mode switch point, mode access point.
<code>runnablePortKind = 8</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef runnablePortKind runnablePortName runnableSubsystemPath tlBlockType slBlockPath connectionPort hDDActivationReasonList hDDSwc 	Activation reasons argument.
<code>runnablePortKind = 9</code>	<ul style="list-style-type: none"> runnablePortInfo swcRef portRef hDDSwc hDDPort tlBlockType slBlockPath connectionPort hDDPortDefinedArgument portDefinedArgumentRef 	Port-defined argument.

Condition	Related Variables	Description
<code>runnablePortKind = 10</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>connectionPort</code> ▪ <code>hDDNvDataElement</code> ▪ <code>nvDataElementRef</code> ▪ <code>hDDSwc</code> ▪ <code>hDDPort</code> 	NvData read access, NvData receive point, NvData write access, NvData send point.
<code>runnablePortKind = 11</code>	<ul style="list-style-type: none"> ▪ <code>runnablePortInfo</code> ▪ <code>swcRef</code> ▪ <code>portRef</code> ▪ <code>operationRef</code> ▪ <code>runnablePortKind</code> ▪ <code>runnablePortName</code> ▪ <code>runnableSubsystemPath</code> ▪ <code>hDDSwc</code> ▪ <code>hDDPort</code> ▪ <code>hDDOperation</code> ▪ <code>tlBlockType</code> ▪ <code>slBlockPath</code> ▪ <code>connectionPort</code> 	Transformer error.

Remark

The fields of the `runnablePortInfo` struct are set according to the structure's `runnablePortKind` field. Only the following fields are always available:

- `swcRef`
- `runnablePortKind`
- `runnablePortName`
- `runnableSubsystemPath`
- `tlBlockType`
- `slBlockPath`
- `connectionPort`

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)

`tl_post_add_arportblock_hook`..... 77

tl_pre_add_comspecblock_hook

Purpose Lets you customize AUTOSAR frame model generation/update.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_generate_swc_model.m** before a ReceiverComSpec or SenderComSpec block is added.

Description You can enter your own commands to be executed before a SenderComSpec/ReceiverComSpec block is added to the model.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
hDDComSpecObject	Yes	Handle of the DD object representing the communication specification: <ul style="list-style-type: none"> ▪ DD DataReceiverComSpec object ▪ DD EventReceiverComSpec object ▪ DD DataSenderComSpec object ▪ DD EventSenderComSpec object
hRunnablePort	Yes	Simulink handle of the [Bus]Output/[Bus]Inport block that the SenderComSpec/ReceiverComSpec block is to be connected to.
bAddComSpecBlock	No	Possible values: <ul style="list-style-type: none"> ▪ 1 - TargetLink adds a SenderComSpec/ReceiverComSpec block for the DD object specified in hDDComSpecObject . (default) ▪ 0 - TargetLink does not add a SenderComSpec/ReceiverComSpec block for the DD object specified in hDDComSpecObject.

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)	
tl_post_add_comspecblock_hook.....	84

Function Subsystem Generation Hook Scripts

Where to go from here**Information in this section**

Data Dictionary to Model.....	103
Model to Data Dictionary.....	116
Stimulus Creation.....	121

Data Dictionary to Model

Where to go from here**Information in this section**

tl_post_add_systemsignatureport_hook.....	104
tl_post_sync_systemsignatureport_hook.....	106
tl_post_syncsystemsignature_hook.....	108
tl_pre_add_systemsignatureport_hook.....	111
tl_pre_sync_systemsignatureport_hook.....	113
tl_pre_syncsystemsignature_hook.....	115

tl_post_add_systemsignatureport_hook

Purpose Lets you customize the synchronization of DD Signature and Function Block objects.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tlSyncSystemSignature** after it added a new port block to the generated system.

Description You can enter your own commands to be executed after a port block was added to the function subsystem.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
systemPath	Yes	Simulink path of the system the port was added to.

Predefined Variable	Read Only	Description
portInfoStruct	Yes	Structure describing the added port block.
hDDSignaturePort	Yes	DD handle of the DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLInport ▪ TLOutport
DDSignaturePortName	Yes	Name of the added port block.
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object.
hDDVariable	Yes	DD handle of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block to be added: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
blockSLPath	Yes	Simulink path of the port block.
bFirstPort	Yes	True if this function is called after the first of all inports outports was added. Otherwise, false.
bLastPort	Yes	True if this function is called after the last of all inports or all outports was added. Otherwise, false.

Related topics

Basics

[Centrally Specifying and Synchronizing Function System Signatures](#) ( TargetLink Customization and Optimization Guide)

References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[Synchronize System Signature](#) ( TargetLink Data Dictionary Manager Reference)

[tl_pre_add_systemsignatureport_hook](#)..... 111

[tlSyncSystemSignature](#) ( TargetLink API Reference)

tl_post_sync_systemsignatureport_hook

Purpose Lets you customize the synchronization of DD Signature and Function Block objects.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tlSyncSystemSignature** after it synchronized an existing port block with the corresponding DD SignaturePort object.

Description You can enter your own commands to be executed after existing port blocks were synchronized with their corresponding DD SignaturePort objects.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
systemPath	Yes	Simulink path of the system the port block resides in.

Predefined Variable	Read Only	Description
portInfoStruct	Yes	Structure describing the synchronized port.
hDDSignaturePort	Yes	DD handle of the DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLInport ▪ TLOutport
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object.
hDDVariable	Yes	DD handle of the Variable object referenced by the PortDataElement object, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the PortDataElement object, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the PortDataElement object, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced by the PortDataElement object, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the synchronized port block: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
syncInfoStruct	Yes	List of structures with the following fields: <ul style="list-style-type: none"> ▪ parameter - Modified parameter. ▪ previousValue - Parameter value before modification. ▪ currentValue - Parameter value after modification.
action	Yes	Action performed for the port during DD-to-model synchronization. The following values are possible: <ul style="list-style-type: none"> ▪ 1 - A port was generated. ▪ 2 - A port was synchronized or must be synchronized. ▪ 3 - A port was skipped. ▪ 4 - A block must be deleted (by the user). ▪ 5 - No action was required.
annotations	Yes	Annotations to be shown in the generated synchronization report, specified as a cell array of strings.
DDSignaturePortName	Yes	Name of the synchronized port block.
bFirstPort	Yes	True if this function is called after the first of all inports or all outports was synchronized. Otherwise, false.
bLastPort	Yes	True if this function is called after the last of all inports or all outports was synchronized. Otherwise, false.

Related topics

Basics

Centrally Specifying and Synchronizing Function System Signatures ([TargetLink Customization and Optimization Guide](#))

References

Create Customization Files Dialog Description ([TargetLink Tool and Utility Reference](#))
 Synchronize System Signature ([TargetLink Data Dictionary Manager Reference](#))
[tl_pre_sync_systemsignaturereport_hook](#)..... 113
[tlSyncSystemSignature](#) ([TargetLink API Reference](#))

tl_post_syncsystemsignature_hook

Purpose

Lets you customize the synchronization of DD Signature and Function Block objects.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tlSyncSystemSignature** after a system was generated from the specified DD objects.

Description

You can enter your own commands to be executed after the function subsystem was created.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
systemInfoStructure	Yes	Structure describing the generated system.
hDDSignature	Yes	DD handle of the Signature object, if applicable. Otherwise, an empty matrix.
hDDFcnBlock	Yes	DD handle of the Block object with BlockType == TL_Function, if applicable. Otherwise, an empty matrix.
systemSLPath	Yes	Simulink path of the generated system.
fcnBlockSLPath	Yes	Simulink path of the Function block inserted into the generated system.
systemInportList	Yes	List of structures containing information about inserted inport blocks.
hDDSignaturePort	Yes	DD handle of the DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLInport ▪ TLOutport
DDSignaturePortName	Yes	Name of the port block to be added.
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object.
hDDVariable	Yes	DD handle of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced from the PortDataElement object, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block to be added: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
blockSLPath	Yes	Simulink path of the port block.


Predefined Variable	Read Only	Description
systemOutputportList	Yes	List of structures containing information about inserted outputport blocks.
hDDSignaturePort	Yes	DD handle of the DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLOutport
DDSignaturePortName	Yes	Name of the port block to be added.
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object.
hDDVariable	Yes	DD handle of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced from the PortDataElement object, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block to be added: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
blockSLPath	Yes	Simulink path of the port block.

Related topics

Basics

[Centrally Specifying and Synchronizing Function System Signatures](#) ( TargetLink Customization and Optimization Guide)

References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[Synchronize System Signature](#) ( TargetLink Data Dictionary Manager Reference)

[tl_pre_syncsystemsyntax_hook](#)..... 115

[tlSyncSystemSignature](#) ( TargetLink API Reference)

tl_pre_add_systemsignatureport_hook

Purpose Lets you customize the synchronization of DD Signature and Function Block objects.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tlSyncSystemSignature** before it adds a new port block to the generated system.

Description You can enter your own commands to be executed before a port block is added to the function subsystem.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
systemPath	Yes	Simulink path of the system the port is to be added to.


Predefined Variable	Read Only	Description
portInfoStruct	Yes	Structure describing the current port.
hDDSignaturePort	Yes	DD handle of the DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLInport ▪ TLOutport
DDSignaturePortName	Yes	Name of the port block to be added.
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object.
hDDVariable	Yes	DD handle of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced from the PortDataElement, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of ReplaceableDataItem object referenced by the PortDataElement object, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block to be added: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
bFirstPort	Yes	True if this function is called before the first of all inports or outports is added. Otherwise, false.
bLastPort	Yes	True if this function is called before the last of all inports or outports is added. Otherwise, false.

Related topics

Basics


[Centrally Specifying and Synchronizing Function System Signatures](#) ( TargetLink Customization and Optimization Guide)

References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[Synchronize System Signature](#) ( TargetLink Data Dictionary Manager Reference)

[tl_post_add_systemsignatureport_hook](#)..... 104

[tlSyncSystemSignature](#) ( TargetLink API Reference)

tl_pre_sync_systemsignatureport_hook

Purpose Lets you customize the synchronization of DD Signature and Function Block objects.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tlSyncSystemSignature** before it synchronizes an existing port block with the corresponding DD SignaturePort object.

Description You can enter your own commands to be executed before existing port blocks are synchronized with the corresponding DD SignaturePort objects.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
systemPath	Yes	Simulink path of the system the port block resides in.

Predefined Variable	Read Only	Description
portInfoStruct	Yes	Structure describing the current port.
hDDSignaturePort	Yes	DD handle of the DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLInport ▪ TLOutport
DDSignaturePortName	Yes	Name of the port block to be synchronized.
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object.
hDDVariable	Yes	DD handle of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced by the PortDataElement, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block to be synchronized: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
blockSLPath	Yes	Simulink path of the port block to be synchronized.
action	Yes	Action performed for the port during synchronization. The following values are possible: <ul style="list-style-type: none"> ▪ 1 - Generates a new port. ▪ 2 - Synchronizes an existing port. ▪ 3 - Skips a port. ▪ 4 - A block must be deleted (by the user). ▪ 5 - No action is required.
annotations	Yes	Annotations to be shown in the generated synchronization report, displayed as a cell array of strings.
bFirstPort	Yes	True if this script is called before the first of all inports or outports is synchronized. Otherwise, false.
bLastPort	Yes	True if this script is called before the last of all inports or outports is synchronized. Otherwise, false.
bSkipSync	No	<ul style="list-style-type: none"> ▪ True - If the current port block is not to be synchronized. ▪ False- If the current port block is to be synchronized.

Related topics

Basics

[Centrally Specifying and Synchronizing Function System Signatures](#) ([TargetLink Customization and Optimization Guide](#))

References

[Create Customization Files Dialog Description](#) ([TargetLink Tool and Utility Reference](#))

[Synchronize System Signature](#) ([TargetLink Data Dictionary Manager Reference](#))

[tl_post_sync_systemsignaturereport_hook](#)..... 106

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

tl_pre_syncsystemsignature_hook

Purpose

Lets you customize the synchronization of DD Signature and Function Block objects.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tlSyncSystemSignature** before it starts generating a system for the specified DD object:

- Signature object
- Block object whose BlockType property is set to TL_Function

Description

You can enter your own commands to be executed before function subsystem generation starts.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
hDDSignature	Yes	DD handle of the Signature object, if applicable. Otherwise, an empty matrix.
hDDFunctionBlock	Yes	DD handle of the Block object with BlockType == TL_Function, if applicable. Otherwise, an empty matrix.
tlSyncSystemSignatureOptions	No	Configuration options of the tlSyncSystemSignature tool, displayed as a list of propertyName/propertyValue pairs.


Example


```
% Modify the type of the system to be created.
tlSyncSystemSignatureOptions = [tlSyncSystemSignatureOptions{:}
{'SystemType','TLSubsystem'}];
```

Related topics**Basics**


[Centrally Specifying and Synchronizing Function System Signatures](#) ( TargetLink Customization and Optimization Guide)

References

[Create Customization Files Dialog Description](#) ( TargetLink Tool and Utility Reference)

[Synchronize System Signature](#) ( TargetLink Data Dictionary Manager Reference)

[tl_post_syncsystemsignature_hook](#)..... 108

[tlSyncSystemSignature](#) ( TargetLink API Reference)

Model to Data Dictionary

Where to go from here**Information in this section**

[tl_post_add_ddsignatureport_hook](#)..... 117

[tl_pre_add_ddsignatureport_hook](#)..... 119

tl_post_add_ddsignatureport_hook

Purpose Lets you customize the generation of a DD Signature object from a function subsystem's signature.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tlSyncSystemSignature** after it added a new DD SignaturePort object to the specified DD Signature object.

Description You can enter your own commands to be executed after a new DD SignaturePort object was added to the specified DD Signature object.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
hDDSignature	Yes	DD handle of the Signature object to which the SignaturePort object was added.
systemPath	Yes	Simulink path of the system in which the port block resides that a corresponding DD SignaturePort object was added to.

Predefined Variable	Read Only	Description
portInfoStruct	Yes	Structure describing the port block and the corresponding SignaturPort object.
hDDSignaturePort	Yes	DD handle of the created DD SignaturePort object.
DDSignaturePortKind	Yes	Signature port kind: <ul style="list-style-type: none"> ▪ SLInport ▪ SLOutport ▪ TLInport ▪ TLOutport
DDSignaturePortName	Yes	Name of the created DD SignaturePort object.
hDDPortDataElement	Yes	DD handle of the associated PortDataElement object, if applicable. Otherwise, an empty matrix.
hDDVariable	Yes	DD handle of the Variable object referenced by the port block and PortDataElement object, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the port block and PortDataElement object, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the port block and PortDataElement object, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced by the port block and PortDataElement object, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block to be added: <ul style="list-style-type: none"> ▪ For Simulink ports: <ul style="list-style-type: none"> ▪ Inport ▪ Outport ▪ For TargetLink ports: <ul style="list-style-type: none"> ▪ TL_Inport ▪ TL_BusInport ▪ TL_Outport ▪ TL_BusOutport
blockSLPath	Yes	Simulink path of the port block.
blockName	Yes	Name of the port block.
blockSLPath	Yes	Simulink path of the port block.
bIsTLBusPort	Yes	True if the port block is a TargetLink Bus Inport or Bus Outport. Otherwise, false.
annotations	Yes	Cell array with annotations regarding the creation of the new DD SignaturePort object.
bFirstPort	Yes	True if this function is called after a DD SignaturePort object was added for the first of all inport blocks or for the first of all outport blocks. Otherwise, false.
bLastPort	Yes	True if this function is called after a DD SignaturePort object was added for the last of all inport blocks or for the last of all outport blocks. Otherwise, false.

Related topics

Basics

Centrally Specifying and Synchronizing Function System Signatures ([TargetLink Customization and Optimization Guide](#))

References

Create Customization Files Dialog Description ([TargetLink Tool and Utility Reference](#))

Synchronize System Signature ([TargetLink Data Dictionary Manager Reference](#))

`tl_pre_add_ddsignatureport_hook`..... 119

`tlSyncSystemSignature` ([TargetLink API Reference](#))

tl_pre_add_ddsignatureport_hook

Purpose

Lets you customize the generation of a DD Signature object from a function subsystem's signature.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tlSyncSystemSignature` before it adds a new DD SignaturePort object to the specified DD Signature object.

Description

You can enter your own commands to be executed before a new DD SignaturePort object was added to the specified DD Signature object.

Predefined variables


The following predefined variables are available:

Predefined Variable	Read Only	Description
hDDSignature	Yes	DD handle of the Signature object to which the SignaturePort object was added.
systemPath	Yes	Simulink path of the system in which the port block resides that a corresponding SignaturePort object was added for.
portInfoStruct	Yes	Structure describing the port block and the corresponding SignaturPort object.
hDDVariable	Yes	DD handle of the Variable object referenced by the port block, if applicable. Otherwise, an empty matrix.
variableRef	Yes	DD reference of the Variable object referenced by the port block, if applicable. Otherwise, an empty matrix.
hDDRDI	Yes	DD handle of the ReplaceableDataItem object referenced by the port block, if applicable. Otherwise, an empty matrix.
rDIRef	Yes	DD reference of the ReplaceableDataItem object referenced by the port block, if applicable. Otherwise, an empty matrix.
blockType	Yes	Type of the port block: <ul style="list-style-type: none"> For Simulink ports: <ul style="list-style-type: none"> Inport Outport For TargetLink ports: <ul style="list-style-type: none"> TL_Inport TL_BusInport TL_Outport TL_BusOutport
blockName	Yes	Name of the port block.
blockSLPath	Yes	Simulink path of the port block.
bIsTLBusPort	Yes	<ul style="list-style-type: none"> True - The port block is a TargetLink BusInport or BusOutport block. False - The port block is not a TargetLink BusInport or a BusOutport block.
bFirstPort	Yes	<ul style="list-style-type: none"> True - If this function is called after a DD SignaturePort object was added for the first of all inport blocks or for the first of all outport blocks. False - Otherwise.
bLastPort	Yes	<ul style="list-style-type: none"> True - If this function is called after a DD SignaturePort object was added for the last of all inport blocks or for the last of all outport blocks. False - Otherwise.

Related topics**Basics**

Centrally Specifying and Synchronizing Function System Signatures ( TargetLink Customization and Optimization Guide)

References

Create Customization Files Dialog Description ( TargetLink Tool and Utility Reference)

Synchronize System Signature ( TargetLink Data Dictionary Manager Reference)

tl_post_add_ddsignaturereport_hook..... 117

tlSyncSystemSignature ( TargetLink API Reference)

Stimulus Creation

Where to go from here**Information in this section**

tl_post_create_stimuli_hook..... 121


tl_pre_create_stimuli_hook..... 123

tl_post_create_stimuli_hook

Purpose

Lets you customize AUTOSAR frame model generation and the synchronization of DD Signature and Function Block objects.

Access


You can access customization files via the **Create Customization Files** dialog that opens if you enter **tlCustomizationFiles** ( TargetLink API Reference) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_create_stimuli_signal.m` after a stimulus signal was generated in the specified Simulink subsystem. The stimulus signal is modeled by means of Ground/Constant/TL_Const and BusCreator blocks.

Description

You can enter your own commands to be executed after stimulus signals of one block were added to the function subsystem.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>hSubsystem</code>	Yes	Simulink handle of the stimulus subsystem.
<code>hDDObject</code>	Yes	DD handle of the object according to which a stimulus signal was generated.
<code>busHierarchy</code>	Yes	Vector of structures describing the hierarchy of the generated bus signal. If the <code>busHierarchy</code> contains only one element, stimuli are generated as a single signal. Otherwise, they are generated as a bus signal.
<code>signalName</code>	Yes	Name of the signal in the bus.
<code>bIsLeaf</code>	Yes	Possible values: <ul style="list-style-type: none"> ▪ 1 - The signal is a leaf signal. ▪ 0 - The signal is not a leaf signal.
<code>parent</code>	Yes	Index within the <code>busHierarchy</code> vector of the element representing the parent signal.
<code>level</code>	Yes	Hierarchy level with base '1'.
<code>width</code>	Yes	Signal width.

Remark

If the `busHierarchy` contains only one element, stimuli are generated as a single signal. Otherwise, they are generated as a bus signal.

Example

```
fprintf('Generation of the stimulus signal corresponding with the DD
object = %s finished.\n', ...
    dsdd('GetAttribute', hDDObject, 'path'));
```

Related topics

Basics

[Centrally Specifying and Synchronizing Function System Signatures](#) ([TargetLink Customization and Optimization Guide](#))

References

[Create Customization Files Dialog Description](#) ([TargetLink Tool and Utility Reference](#))

[Synchronize System Signature](#) ([TargetLink Data Dictionary Manager Reference](#))

[tl_pre_create_stimuli_hook](#) 123

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

tl_pre_create_stimuli_hook

Purpose

Lets you customize AUTOSAR frame model generation and the synchronization of DD Signature and Function Block objects.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_create_stimuli_signal.m** before a stimulus signal is generated in the specified Simulink subsystem from the Constant/TL_Const/Ground and BusCreator blocks.

Description

You can enter your own commands to be executed before stimulus signals of one block are added to the function subsystem.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
hSubsystem	Yes	Simulink handle of the stimulus subsystem.
hDDObject	Yes	DD handle of the object according to which a stimulus signal is generated.
busHierarchy	Yes	Vector of structures describing the hierarchy of the generated bus signal. If the busHierarchy contains only one element, stimuli are generated as a single signal. Otherwise, they are generated as a bus signal.
signalName	Yes	Name of the signal in the bus.
bIsLeaf	Yes	Possible values: <ul style="list-style-type: none"> ▪ 1 - The signal is a leaf signal. ▪ 0 - The signal is not a leaf signal.
parent	Yes	Index within the busHierarchy vector of the element representing the parent signal.
level	Yes	Hierarchy level with base '1'.
width	Yes	Signal width.

Remark

If the **busHierarchy** contains only one element, stimuli are generated as a single signal. Otherwise, they are generated as a bus signal.

Example

```
fprintf('Generating stimuli signal corresponding with the DD object
= %s\n', ...
dsdd('GetAttribute', hDDObject, 'path'));
```

Related topics**Basics**

[Centrally Specifying and Synchronizing Function System Signatures \(TargetLink Customization and Optimization Guide\)](#)

References

[Create Customization Files Dialog Description \(TargetLink Tool and Utility Reference\)](#)
[Synchronize System Signature \(TargetLink Data Dictionary Manager Reference\)](#)
[tl_post_create_stimuli_hook](#) 121
[tlSyncSystemSignature \(TargetLink API Reference\)](#)

Import/Export Hook Scripts (AUTOSAR)

Where to go from here

Information in this section

tl_post_arexport_hook.....	125
tl_post_arimport_hook.....	127
tl_pre_arexport_hook.....	128
tl_pre_arimport_hook.....	131
tl_pre_containerexport_hook.....	132

tl_post_arexport_hook

Purpose

Lets you customize the AUTOSAR file export from the TargetLink Data Dictionary.

Access

You can access customization files via the **Create Customization Files** dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated after the ARXML files were created upon AUTOSAR export.

Description

You can enter your own commands to be executed after AUTOSAR files were exported from the TargetLink Data Dictionary.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
ExportArguments	Yes	Structure with the following fields:
Files	Yes	Cell array of strings with the full path name of the export file. The cell array contains only the file that is entered as an export option. If package support is enabled, the actual file name(s) depend on the package information.
AUTOSARVersion	Yes	String containing the exact AUTOSAR version selected for export, e.g., '3.0.2'.
Validate	Yes	The following values are possible: <ul style="list-style-type: none"> ▪ on - Validation is performed. ▪ off - Validation is not performed.
ForceFileOverwrite	Yes	The following values are possible: <ul style="list-style-type: none"> ▪ on - Existing files of identical names are overwritten automatically. ▪ off - Existing files of identical names are not overwritten automatically
DdObjectToExport	Yes	Path to the DD object selected for export. This value is set only if an export from the /Pool area of the Data Dictionary is performed. [LF] If a node from the /Subsystems area of the Data Dictionary is selected, the property is set to ''.
Parent	Yes	This option is used only during import. During export, it is set to '' and has no effect.
Subsystems	Yes	Cell array of strings containing the names of the /Subsystem nodes of the Data Dictionary that were selected for export. Example: {'controller'}. This property is irrelevant if data is exported from the /Pool area of the Data Dictionary.
Prefix	Yes	The following values are possible: <ul style="list-style-type: none"> ▪ on - The config file for the names for the internal behavior, implementation, etc. is applied. ▪ off - The config file for the names for the internal behavior, implementation, etc. is not applied.

Remark

The settings above were already used during the export. Changing the options therefore has no effect on the export, because the files were already created.

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)
 tl_pre_arexport_hook..... 128

tl_post_arimport_hook

Purpose Lets you customize the AUTOSAR file import to the TargetLink Data Dictionary.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated after the content of ARXML files was imported to the Data Dictionary.

Description You can enter your own commands to be executed after AUTOSAR files were imported to the TargetLink Data Dictionary.

Predefined variables The following predefined variables are available:


Predefined Variable	Read Only	Description
SWCNameList	Yes	Contains the imported DD SoftwareComponent object names.

Predefined Variable	Read Only	Description
ImportArguments	Yes	Structure with the following fields:
Files	Yes	Cell array of strings with the full path names of the files selected for import.
AUTOSARVersion	Yes	String containing the exact AUTOSAR version selected for import, e.g., '3.0.2'.
Validate	Yes	The following values are possible: <ul style="list-style-type: none"> on <ul style="list-style-type: none"> - Validation is performed. off <ul style="list-style-type: none"> - Validation is not performed.
Prefix	Yes	Possible values: 'on' or 'off'. This option is irrelevant for the import.
ForceFileOverwrite	Yes	Possible values: 'on' or 'off'. This option is irrelevant for the import.
DdObjectToExport	Yes	Not required for importing AUTOSAR files.
Parent	Yes	The import location. Typically set to <code>//DD0</code> because <code>DD0</code> is the location the content of ARXML files is normally imported to. Can also be set to, e.g., <code>//DD1</code> to import in a non-default Data Dictionary.
Subsystems	Yes	Not required for importing AUTOSAR files.

Remark The settings above were already used during AUTOSAR import. Therefore, changes to the options above have no effect.

Related topics


References

Create Customization Files Dialog Description ( TargetLink Tool and Utility Reference)

tl_pre_arimport_hook..... 131

tl_pre_arexport_hook

Purpose Lets you customize the AUTOSAR file export from the TargetLink Data Dictionary.


Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( TargetLink API Reference) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated before the ARXML files are created upon AUTOSAR export but after the options for the AUTOSAR export were set.

Description

You can enter your own commands to be executed before AUTOSAR files are exported from the TargetLink Data Dictionary.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
ExportArguments	Yes	Structure with the following fields:
Files	Yes	Cell array of strings with the full path name of the export file. The cell array contains only the file that is entered as an export option. If package support is enabled, the actual file name(s) depend on the package information.
AUTOSARVersion	Yes	String containing the exact AUTOSAR version selected for export, e.g., '3.0.2'.
Validate	Yes	The following values are possible: <ul style="list-style-type: none"> ▪ on - Validation is performed. ▪ off - Validation is not performed.
ForceFileOverwrite	Yes	The following values are possible: <ul style="list-style-type: none"> ▪ on - Existing files of identical names are overwritten automatically. ▪ off - Existing files of identical names are not overwritten automatically.
DdObjectToExport	Yes	Path to the DD object selected for export. This value is set only if an export from the /Pool area of the DD is performed. If a node from the /Subsystems area of the DD is selected, the property is set to ''.
Parent	Yes	This option is used only during import. During export, it is set to '' and has no effect.
Subsystems	Yes	Cell array of strings containing the names of the /Subsystem nodes of the Data Dictionary that were selected for export. Example: {'controller'}. This property is irrelevant if data is exported from the /Pool area of the Data Dictionary.
Prefix	Yes	The following values are possible: <ul style="list-style-type: none"> ▪ on - The config file for the names for the internal behavior, implementation, etc. is applied. ▪ off - The config file for the names for the internal behavior, implementation, etc. is not applied.

Related topics**References**

Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference)

tl_post_arexport_hook..... 125

tl_pre_arimport_hook

Purpose	Lets you customize the AUTOSAR file import to the Data Dictionary.
Access	<p>You can access customization files via the Create Customization Files dialog that opens if you enter tlCustomizationFiles (📖 TargetLink API Reference) in the MATLAB Command Window.</p> <p>TargetLink performs the following steps:</p> <ul style="list-style-type: none">▪ Copies the customization file template to your current working directory.▪ Derives the functional customization file from its template.▪ Opens the customization file in the editor. <div>Note<p>Customization files must reside on the MATLAB <i>and</i> TargetLink search paths.</p><p>For instructions, refer to Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators (📖 TargetLink Customization and Optimization Guide).</p></div>
Hook point	This M script is evaluated before the content of ARXML files is imported to the Data Dictionary.
Description	You can enter your own commands to be executed before AUTOSAR files are imported to the Data Dictionary.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
ImportArguments	Yes	Structure with the following fields:
Files	Yes	Cell array of strings with the full path names of the files selected for import.
AUTOSARVersion	Yes	String containing the exact AUTOSAR version selected for export, e.g., '3.0.2'.
Validate	Yes	The following values are possible: <ul style="list-style-type: none"> on <ul style="list-style-type: none"> - Validation is performed. off <ul style="list-style-type: none"> - Validation is not performed.
Prefix	Yes	Possible values: 'on' or 'off'. This option is irrelevant for the import.
ForceFileOverwrite	Yes	Possible values: 'on' or 'off'. This option is irrelevant for the import.
DdObjectToExport	Yes	Irrelevant for the import.
Parent	Yes	The import location. Typically set to //DD0 because DD0 is the location the content of ARXML files is normally imported to. Can also be set to, e.g., //DD1 to import in a non-default Data Dictionary.
Subsystems	Yes	Irrelevant for the import.

Remark

The settings above were entered using the DD AUTOSAR Import Dialog or the DD API.

Related topics**References**

Create Customization Files Dialog Description ( TargetLink Tool and Utility Reference)


tl_post_arimport_hook..... 127

tl_pre_containerexport_hook

Purpose

Lets you customize the container export.


Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( TargetLink API Reference) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.
For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by `tl_export_container.m` before the container is exported.


Description You can enter your own commands to be executed before the container is exported.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
hDDSubsystemList	Yes	List of DD handles of Subsystem objects.

Related topics

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

Model Processing Hook Scripts

Where to go from here Information in this section

tl_post_load_hook	134
tl_pre_save_hook	135

tl_post_load_hook

Purpose

Lets you execute commands after loading a model from file.

Access

You can access customization files via the **Create Customization Files** dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated during the PostLoadFcn callback of a Simulink model.

Description

You can enter your own commands to be executed each time the model is loaded.

Predefined variables

The following predefined variables are available:

Predefined Variable	Read Only	Description
system	Yes	The Simulink model or library that caused the callback.

Related topics

References

[Create Customization Files Dialog Description](#) ([TargetLink Tool and Utility Reference](#))

tl_pre_save_hook..... 135

tl_pre_save_hook

Purpose Lets you execute commands before saving a model to file.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated during the PreSaveFcn callback of a model, immediately before it is saved to file.

Description You can enter your own commands to be executed each time the model is saved.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
system	Yes	The Simulink model or library that caused the callback.

Related topics

References

Create Customization Files Dialog Description ([TargetLink Tool and Utility Reference](#))

tl_post_load_hook..... 134

Production Code Simulation Hook Scripts

tl_pre_prodcode_sim_hook

Purpose

Lets you modify the values of the production code variables, e.g., parameters and data variant switch variables.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by the production code simulation S-function at the end of the mdlStart function, after the initialization was finished but before any simulation step is calculated.

If to the simulation application production code of the DD-based code generation units was linked and if it contains variables that are initialized in restart functions, the S-function also calls these restart functions in its mdlStart function, regardless of the TargetLink subsystem for which is was generated. This means that the restart functions from the DD-based code generation units can be called and the corresponding Variables initialized multiple times if multiple TargetLink subsystems are simulated in the SIL/PIL simulation mode concurrently. In contrast, the restart functions of the model-based code generation units, such as TargetLink subsystems, are called only if the S-function of the corresponding TargetLink subsystem is evaluated. Therefore, if you intend to change the values of some DD-based variables in this hook script, make sure that these changes are made independently of the current TargetLink subsystem for which the S-function is called.

Description You can enter your own commands, e.g., `tlSimInterface`, to be executed before the first simulation step is calculated.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
<code>model</code>	Yes	Name of the Simulink model.
<code>tlSubsystem</code>	Yes	Name of the current TargetLink subsystem.
<code>board</code>	Yes	Name of the board the simulation is currently performed on. For SIL simulation, <code>HostPC</code> is set as the board name.
<code>hPlatformHandle</code>	Yes	Platform handle needed to access the production code application.


Remark You can use this hook script to modify the values of the production code variables, e.g., parameters and data variant switch variable.

Example


```
% Get the name of the application downloaded to the active
simulation platform.
[bIsLoaded, ApplName] = tlSimInterface('IsApplDownloaded',
'BoardName', board);
fprintf('At the current simulationplatform '%s' the following
application is loaded: %s\n',board, ApplName);
```

Related topics

Examples

[Example of Writing Parameter Values via Hook Scripts](#) ( [TargetLink Preparation and Simulation Guide](#))

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

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

System Preparation Hook Scripts

Where to go from here

Information in this section

tl_post_clear_hook.....	138
tl_post_preparation_hook.....	139
tl_pre_clear_hook.....	141
tl_pre_preparation_hook.....	142

tl_post_clear_hook

Purpose

Lets you customize the phase when a TargetLink system is being cleared from TargetLink data.

Access

You can access customization files via the **Create Customization Files** dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by **tl_clear_system.m** immediately after the system was cleared from TargetLink.

Description

You can enter commands to be executed after the TargetLink system was cleared.

Predefined variables

The following predefined variables are available:


Predefined Variable	Read Only	Description
hSubSystem	Yes	Handle of the system (model or subsystem) that is being processed.

Remark

In `tl_post_clear_hook.m`, you can perform system checks to ensure that defined conditions are met, and use TargetLink error handling functions to issue error messages if the conditions are not met.

Related topics**Basics**

[Making a Simulink Model TargetLink-Compliant](#) ( [TargetLink Preparation and Simulation Guide](#))

[Using the TargetLink M-Script Interface \(API\)](#) ( [TargetLink Interoperation and Exchange Guide](#))

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))


[tl_pre_clear_hook](#)..... 141

tl_post_preparation_hook

Purpose

Lets you customize the phase when a Simulink system is being prepared for TargetLink.

Access


You can access customization files via the **Create Customization Files** dialog that opens if you enter `tlCustomizationFiles` ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by `tl_prepare_system.m` immediately before the system preparation finishes.

Description

You can enter commands to be executed after the Simulink system was prepared.

Predefined variables


The following predefined variables are available:


Predefined Variable	Read Only	Description
<code>hSubSystem</code>	Yes	Handle of the system (model or subsystem) prepared for TargetLink.

Remark

You can call TargetLink API commands in `tl_post_preparation_hook.m` to set TargetLink block properties to values different from the defaults specified in the TargetLink library.

Related topics**Basics**

[Making a Simulink Model TargetLink-Compliant](#) ( [TargetLink Preparation and Simulation Guide](#))

[Using the TargetLink M-Script Interface \(API\)](#) ( [TargetLink Interoperation and Exchange Guide](#))

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))

`tl_pre_preparation_hook`..... 142

tl_pre_clear_hook

Purpose Lets you customize the phase when a TargetLink system is being cleared from TargetLink data.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by **tl_clear_system.m** immediately before the system is cleared from TargetLink.

Description You can enter your own commands to be executed before the TargetLink system is cleared.

Predefined variables The following predefined variables are available:


Predefined Variable	Read Only	Description
hSubSystem	Yes	Handle of the system (model or subsystem) to be processed.


Remark You can call TargetLink API commands in **tl_pre_clear_hook.m** to set TargetLink block properties to values different from the defaults specified in the **tl1lib** TargetLink library.

Example

```
fprintf('%s\n', which(mfilename));
```

Related topics**Basics**

Making a Simulink Model TargetLink-Compliant ( [TargetLink Preparation and Simulation Guide](#))

Using the TargetLink M-Script Interface (API) ( [TargetLink Interoperation and Exchange Guide](#))

References

Create Customization Files Dialog Description ( [TargetLink Tool and Utility Reference](#))


tl_post_clear_hook..... 138

tl_pre_preparation_hook

Purpose

Lets you customize the phase when a Simulink system is being prepared for TargetLink.

Access


You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ( [TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ( [TargetLink Customization and Optimization Guide](#)).

Hook point

This M script is evaluated by tl_prepare_system.m immediately before system preparation starts.

Description

You can enter your own commands to be executed immediately before system preparation starts.

Predefined variables		The following predefined variables are available:
Predefined Variable	Read Only	Description
hSubSystem	Yes	Handle of the system (model or subsystem) to be prepared for TargetLink.

Remark	In <code>tl_pre_preparation_hook.m</code> , you can perform system checks to ensure that defined conditions are met, and use TargetLink error handling functions to issue error messages if the conditions are not met.
--------	---

Related topics	Basics
	<div>Making a Simulink Model TargetLink-Compliant (📖 TargetLink Preparation and Simulation Guide) Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)</div>
	References
	<div>Create Customization Files Dialog Description (📖 TargetLink Tool and Utility Reference) tl_post_preparation_hook..... 139</div>

Upgrade Hook Scripts

Where to go from here	Information in this section
	<div>tl_post_upgrade_hook..... 144 tl_pre_upgrade_hook..... 145</div>

tl_post_upgrade_hook

Purpose Lets you customize the system upgrade.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by tl_upgrade.m immediately before the upgrade for each system finishes.

Description You can enter your own commands to be executed after system upgrade.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
hSystem	Yes	Handle of the system (model or library) to be upgraded.
userData	Yes	Intended for data exchange between upgrade hook scripts.

Related topics

References

Create Customization Files Dialog Description ([TargetLink Tool and Utility Reference](#))

tl_pre_upgrade_hook..... 145

tl_pre_upgrade_hook

Purpose Lets you customize the system upgrade.

Access You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.

For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#)).

Hook point This M script is evaluated by tl_upgrade.m immediately before the system upgrade is started.

Description You can enter your own commands to be executed before system upgrade.

Predefined variables The following predefined variables are available:

Predefined Variable	Read Only	Description
hSystem	Yes	Handle of the system (model or library) to be upgraded.
userData	No	Intended for data exchange between upgrade hook scripts.

Related topics

References

Create Customization Files Dialog Description ([TargetLink Tool and Utility Reference](#))
 tl_post_upgrade_hook..... 144

Look-Up Table Functions

Look-Up Table Functions Customization Files


Purpose

To customize look-up table functions in the code.


Access

You can use the files in
<DocumentsFolder>\dSPACE\TargetLink\<Version>\Demos\table\ as
template.

Description

Filename	Description
lookup1d_script.m lookup2d_script.m indexsearch_script.m interpolation_script.m	User-defined M-script files to customize the code generated for the following blocks: <ul style="list-style-type: none">▪ Look-Up Table▪ Look-Up Table (2-D)▪ PreLook-Up Index Search▪ Interpolation (n-D) using PreLook-Up See also Using Custom Look-Up Functions ( TargetLink Preparation and Simulation Guide).

Related topics**Basics**

[Using Custom Look-Up Functions](#) ( [TargetLink Preparation and Simulation
Guide](#))

Code Formatting

Code Formatting Customization Files

Purpose

Customization files related to code formatting.

Access



There is no common folder for these files. For information on file access, refer to the table below.

Description



Filename	Description
cconfig.xml	Specifies the code output format, particularly the formatting of comments. The file can be found in <TL_InstRoot>\Matlab\T1\Config\CodeGen
XSL Files Related to Code Formatting	
TL_CSourceCodeSS.xsl	TargetLink's root style sheet. The root style sheet and further XSL files are placed in <TL_InstRoot>\Matlab\T1\XML\CodeGen\StyleSheets.

Related topics

Basics

Basics on Code Formatting ( TargetLink Customization and Optimization Guide)
Defining Search Paths for Customization Files, Demo Models, TSM Extension
Packages, and Instruction Set Simulators ( TargetLink Customization and Optimization Guide)

HowTos

How to Edit the Style Definition File ( TargetLink Customization and Optimization Guide)
How to Select the Predefined Root Style Sheet ( TargetLink Customization and Optimization Guide)

References

tlCustomizationFiles ( TargetLink API Reference)

Bus Initialization

Where to go from here

Information in this section

[tlGetBusStructMapping](#)..... 148

Information in other sections

[Basics on Initializing Buses via Initial Condition Structures](#)
([📖 TargetLink Preparation and Simulation Guide](#))

[How to Manually Create a Mapping Between a DD Variable Object and a Simulink Bus](#) ([📖 TargetLink Preparation and Simulation Guide](#))

[ddv](#) ([📖 TargetLink API Reference](#))

tlGetBusStructMapping

Purpose

To provide custom mappings between struct-based DD Variable objects and Simulink.Bus objects.

Access

You can access customization files via the Create Customization Files dialog that opens if you enter **tlCustomizationFiles** ([📖 TargetLink API Reference](#)) in the MATLAB Command Window.

TargetLink performs the following steps:

- Copies the customization file template to your current working directory.
- Derives the functional customization file from its template.
- Opens the customization file in the editor.

Note

Customization files must reside on the MATLAB *and* TargetLink search paths.


For instructions, refer to [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([📖 TargetLink Customization and Optimization Guide](#)).

Result

This customization file is called whenever **ddv** is called for struct variables.

Description

You can enter your own commands in a copy of this template. These commands are executed whenever **ddv** is called for struct variables.


For details on defining custom mappings, refer to [How to Manually Create a Mapping Between a DD Variable Object and a Simulink Bus](#) ( [TargetLink Preparation and Simulation Guide](#)).

Predefined variables


The following predefined variables are available:

Predefined Variable	Description
myBusObj	The Simulink.Bus object that describes the bus whose elements you want to map to the DD Variable object's components.
mapping	The mapping table returned by this hook script.
ddStructVar	The DD Variable object whose components you want to map to the leaf bus signals of the bus.


Related topics**Basics**

[Basics on Initializing Buses via Initial Condition Structures](#) ( [TargetLink Preparation and Simulation Guide](#))

HowTos

[How to Manually Create a Mapping Between a DD Variable Object and a Simulink Bus](#) ( [TargetLink Preparation and Simulation Guide](#))

References

[Create Customization Files Dialog Description](#) ( [TargetLink Tool and Utility Reference](#))
[ddv](#) ( [TargetLink API Reference](#))

MATLAB Import Export API (MLIE)

Customization Files For The MATLAB Import Export API (MLIE)

Provided files

TargetLink provides several customization files for the MATLAB Import Export API (MLIE) in the following directory:

<TL_InstRoot>\Demos\TL\ML_ImportExport\Private

The following list shows all the files located in this folder and their functions:

File Name	Function
mlie_add_custom_property.m	Adds a custom property to a DD Variable object.
mlie_calc_width.m	Returns a DD Width property for a MATLAB size.
mlie_determine_type_for_var.m	Determines the type of the variable that is to be exported
mlie_default_export_cb.m	Serves as the default export callback.
mlie_default_import_cb.m	Serves as the default import callback.
mlie_export_statement.m	<ul style="list-style-type: none"> Writes a MATLAB language statement to an M script file or Evaluates it in the MATLAB workspace.
mlie_filename_ends_with_extension.m	Helps to identify a file name extension.
mlie_get_model_workspace.m	Retrieves the handle of the model workspace for the specified model name (supported from MATLAB R 14 and upwards).
mlie_map_object_property.m	Handles the mapping between Simulink and DD Variable object properties.
mlie_map_property_for_export.m	Handles the mapping of a DD property to a MATLAB/Simulink value/property during export.
mlie_map_simple_type.m	Maps data types of simple variables between the DD and MATLAB/Simulink.
mlie_number_is_even.m	Checks whether a number is even.

These files are all either directly or indirectly called by the **dsdd_mlie_import** ('<propertyName>', '<propertyValue>',...) and **dsdd_mlie_export** ('<propertyName>', '<propertyValue>',...), API functions which are located in <TL_InstRoot>\Demos\TL\ML_ImportExport. These two API functions call the callback functions for each object to be processed and rejected by a filter. One object is processed at a time.

Related topics**Basics**

[Basics on Customizing the MATLAB Import Export API \(📖 TargetLink Interoperation and Exchange Guide\)](#)
[Basics on Importing and Exporting Data Between MATLAB and DD Variable Objects \(📖 TargetLink Interoperation and Exchange Guide\)](#)

HowTos

[How to Customize the MATLAB Import Export API \(📖 TargetLink Interoperation and Exchange Guide\)](#)

M Files for Customizing MATLAB

Where to go from here

Information in this section

dsfinish.m	152
To carry out user-specific commands before MATLAB is shut down.	
dspoststartup.m	153
To carry out user-specific commands after the initialization phase of the dSPACE software.	
dsstartup.m	154
To carry out user-specific commands after the dSPACE software is initialized.	

dsfinish.m

Purpose

To carry out user-specific commands before MATLAB is shut down.

Result

Each time you exit MATLAB, this M file is executed.


Description

Create a new M file and place it in the MATLAB search path, for example, in MATLAB's working folder. All valid MATLAB commands are allowed in it, including the dSPACE-specific commands. Use the standard M file syntax.

Note

- You can specify several **dsfinish.m** files. All **dsfinish.m** files located in the MATLAB search path are executed before MATLAB is shutdown.
- For compatibility reasons available user-specific **finish.m** files are executed. This behavior can, however, change with future MATLAB versions. It is therefore recommended to use **dsfinish.m** files.

Related topics**HowTos**

[How to Customize the MATLAB Start-Up](#) ( [TargetLink Customization and Optimization Guide](#))

References

[dsstartup.m](#)..... 154

dspoststartup.m

Purpose

To carry out user-specific commands after the initialization phase of the dSPACE software.

Result

Each time you start MATLAB, this M-file is executed after the initialization phase of the dSPACE software.

Description

If you create the M file representing this optional script, you can add generally all valid MATLAB commands and dSPACE-specific commands in it. The file must be placed in the MATLAB search path. It is executed directly in the MATLAB base workspace, which is necessary for executing some specific MATLAB functions.

The script is also executed if the initialization of the dSPACE software failed. It is executed after **dsstartup.m**.

Related topics**HowTos**

[How to Customize the MATLAB Start-Up](#) ( [TargetLink Customization and Optimization Guide](#))

References

[dsstartup.m](#)..... 154

dsstartup.m

Purpose	To carry out user-specific commands after the dSPACE software is initialized.
Result	Each time you start MATLAB, this M-file is executed after the dSPACE software is initialized.
Description	Create a new M file and place it in the MATLAB search path, for example, in MATLAB's working folder. In general, all valid MATLAB commands are allowed in it, including the dSPACE-specific commands. Use the standard M file syntax.
Example	<p>Suppose you want MATLAB to automatically change to your working folder <code>d:\work</code> and open the Simulink model <code>my_model.slx</code>. Create the <code>dsstartup.m</code> file and write the following commands to it:</p> <pre>cd d:\work my_model</pre>

Related topics

HowTos

[How to Customize the MATLAB Start-Up \(📖 TargetLink Customization and Optimization Guide\)](#)

References

dsfinish.m	152
dspoststartup.m	153

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.....	156
A.....	157
B.....	160
C.....	161
D.....	164
E.....	166
F.....	167
G.....	168
I.....	168
L.....	170
M.....	171
N.....	173
O.....	174
P.....	175
R.....	176
S.....	178
T.....	180
U.....	182
V.....	182
W.....	183

Numerics

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

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

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
- Multiport Switch (Data Input port)
- Probe
- Sink
- Signal Conversion
- Switch (Data Input port)
- Unit Delay
- Stateflow Data
- MATLAB Function Data

C

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

Calibration Changing the [calibration parameter](#) values of [ECUs](#).

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

Calprm Defined in a [calprm interface](#). Calprms represent [calibration parameters](#) that are accessible via a [measurement and calibration system](#).

Calprm interface An [interface](#) that is provided or required by a [software component \(SWC\)](#) via a [port \(AUTOSAR\)](#).

Calprm software component A special [software component \(SWC\)](#) that provides [calprms](#). Calprm software components have no [internal behavior](#).

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

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

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

Classic AUTOSAR Short name for the AUTOSAR *Classic Platform* standard that complements [Adaptive AUTOSAR](#).

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

See also [simplified initialization mode](#).

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

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

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

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

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

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

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

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

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

Code variant Code variants lead to source code that is generated differently depending on which variant is selected (i.e., 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](#).

Symbols

<model>.a2l 24

C

Common Program Data folder 8
CommonProgramDataFolder 8

D

Documents folder 8
DocumentsFolder 8
DSFINISH.M file 152
DSPOSTSTARTUP.M file 153
DSSTARTUP.M file 154

F

files

code generation

indexsearch_script.m 146
interpolation_script.m 146
lookup1d_script.m 146
lookup2d_script.m 146
TL_CConfig.xml 147
TL_CSourceCodeSS.xml 147

containing input data for the TargetLink Data Server

<plot>.mat 24

DSFINISH.M 152

DSPOSTSTARTUP.M 153

DSSTARTUP.M 154

Fixed-Point Library

functions 10

macros 10

generated by TargetLink 17

generated by the code generator

<cgu>.c 18

<cgu>.h 18

<cgu>_fri.c 18

<cgu>_fri.h 18

<customcode>.c 18

<customcode>_flp.c 18

<function>.c 18

<function>.h 18

<user>.c 18

<user>.h 18

tl_aux_defines_<SubsystemID>.h 18

tl_basetypes.h 18

tl_defines_<SubsystemID>.h 18

tl_types.h 18

tlsim_<model>_globals.c 18

udt_<SubsystemID>.h 18

generated by the host MEX compiler

<customcode>_flp.mexw64 21

<subsystem>_sf.mexw64 21

generated by the make procedure of the

target compiler

*.asm 22

*.elc 22

*.elk 22

*.err 22

*.ers 22

*.L 22

*.lis 22

*.ls 22

*.lst 22

*.o 22

*.obj 22

*.s 22

*.src 22

*.xao 22

<application>.abs 22

<application>.map 22

<model>.mk 22

generated by the Message Browser

tl_messages.log 23

generated during ASAM MCD-2 MC file

generation 24

generated during code size information

generation

<application>.csi 22

dummy.map 22

generated during documentation generation

<model>.png 23

<model>_<simulation>.png 23

<model>_<tlsubsystem>.png 23

<model>_<tlsubsystem>_<function>.png 23

<model>_frame.html 23

<model>_link.html 23

<model>_main.html 23

<model>_main.pdf 23

<model>_main.rtf 23

generated during frame generation

<system>_frm.h 20

<system>_pcf.c 20

<system>_sf.c 20

tl_clean.bat 20

static files 9

system clearance

tl_clear_system.log 28

system preparation

tl_map_block_rtw_parameters_config 28

tl_map_constraints_config 28

tl_map_inheritance_config 28

tl_map_lock_flag_config 28

tl_map_output_scaling_config 28

tl_map_param_scaling_config 28

tl_map_saturation_flag_config 28

tl_map_sf_compiled_scaling_config 28

tl_map_sf_scaling_config 28

tl_map_signal_scaling_config 28

tl_map_subsystem_rtw_parameters_config 28

tl_post_clear_hook 28

tl_post_preparation_hook 28

tl_pre_clear_hook.m 28

tl_pre_preparation_hook 28

tl_prepare_system.log 28

tl_remap_constraints_config 28

tl_remap_lock_flag_config 28

tl_remap_output_scaling_config 28

tl_remap_saturation_flag_config 28

tl_remap_sf_scaling_config 28

system synchronization

tl_sync_system.log 28

Fixed-Point Library

functions

controlling the use of 15

naming convention 13

macros

controlling the use of 15

naming convention 13

funcSL2TL field 37

funcTL2SL field 37

H

Hook Scripts

tl_autodoc_hook 67

tl_post_add_arportblock_hook 77

tl_post_add_comspecblock_hook 84

tl_post_add_ddsignaturereport_hook 117

tl_post_add_operationcallsubsystem_hook 85

tl_post_add_operationresultprovidersubsystem_hook 87

tl_post_add_runnablesubsystem_hook 89

tl_post_add_swcsubsystem_hook 90

tl_post_add_systemsignaturereport_hook 104

tl_post_arexport_hook 125

tl_post_arimport_hook 127

tl_post_autodoc_filter_hook 71

tl_post_clear_hook 138

tl_post_codegen_error_hook 57

tl_post_codegen_finished_hook 58

tl_post_codegen_hook 60

tl_post_compile_host_hook 45

tl_post_compile_target_hook 46

tl_post_create_stimuli_hook 121

tl_post_download_hook 48

tl_post_export_files_hook 74

tl_post_framegen_hook 62

tl_post_load_hook 134

tl_post_preparation_hook 139

tl_post_swcmodeigen_hook 93

tl_post_sync_systemsignaturereport_hook 106

tl_post_syncsystemsignature_hook 108

tl_post_upgrade_hook 144

tl_pre_add_arportblock_hook 95

tl_pre_add_comspecblock_hook 102

tl_pre_add_ddsignaturereport_hook 119

tl_pre_add_systemsignaturereport_hook 111

tl_pre_arexport_hook 128

tl_pre_arimport_hook 131

tl_pre_clear_hook 141

tl_pre_codegen_hook 63

tl_pre_codegen_init_hook 65

tl_pre_compare_creator_options_hook 49

tl_pre_compile_host_hook 50

tl_pre_compile_target_hook 53

tl_pre_containerexport_hook 132

tl_pre_create_stimuli_hook 123

tl_pre_download_hook 55
 tl_pre_export_files_hook 75
 tl_pre_preparation_hook 142
 tl_pre_procode_sim_hook 136
 tl_pre_save_hook 135
 tl_pre_sync_systemsignatureport_hook 113
 tl_pre_syncsystemsignature_hook 115
 tl_pre_upgrade_hook 145

L

Local Program Data folder 8
 LocalProgramDataFolder 8

M

mapSL2TL field 36
 mapTL2SL field 36

S

slblock field 34
 static files 9

T

tl_autodoc_hook 67
 tl_post_add_arportblock_hook 77
 tl_post_add_comspecblock_hook 84
 tl_post_add_ddsignatureport_hook 117
 tl_post_add_operationcallsubsystem_hook 85
 tl_post_add_operationresultprovidersubsystem_h
 ook 87
 tl_post_add_runnableSubsystem_hook 89
 tl_post_add_swcsystem_hook 90
 tl_post_add_systemsignatureport_hook 104
 tl_post_arexport_hook 125
 tl_post_arimport_hook 127
 tl_post_autodoc_filter_hook 71
 tl_post_clear_hook 138
 tl_post_codegen_error_hook 57
 tl_post_codegen_finished_hook 58
 tl_post_codegen_hook 60
 tl_post_compile_host_hook 45
 tl_post_compile_target_hook 46
 tl_post_create_stimuli_hook 121
 tl_post_download_hook 48
 tl_post_export_files_hook 74
 tl_post_framegen_hook 62
 tl_post_load_hook 134
 tl_post_preparation_hook 139
 tl_post_swcmodeigen_hook 93
 tl_post_sync_systemsignatureport_hook 106
 tl_post_syncsystemsignature_hook 108
 tl_post_upgrade_hook 144
 tl_pre_add_arportblock_hook 95
 tl_pre_add_comspecblock_hook 102
 tl_pre_add_ddsignatureport_hook 119
 tl_pre_add_systemsignatureport_hook 111
 tl_pre_arexport_hook 128
 tl_pre_arimport_hook 131
 tl_pre_clear_hook 141
 tl_pre_codegen_hook 63

tl_pre_codegen_init_hook 65
 tl_pre_compare_creator_options_hook 49
 tl_pre_compile_host_hook 50
 tl_pre_compile_target_hook 53
 tl_pre_containerexport_hook 132
 tl_pre_create_stimuli_hook 123
 tl_pre_download_hook 55
 tl_pre_export_files_hook 75
 tl_pre_preparation_hook 142
 tl_pre_procode_sim_hook 136
 tl_pre_save_hook 135
 tl_pre_sync_systemsignatureport_hook 113
 tl_pre_syncsystemsignature_hook 115
 tl_pre_upgrade_hook 145
 tlblock field 35