

TargetLink

New Features and Migration Guide

For TargetLink 5.1

Release 2020-B – November 2020

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

© 2000 - 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 Guide 21

New Features 25

Current TargetLink Version.....	26
New Features of TargetLink 5.1.....	26
Modeling in Simulink or Stateflow.....	26
Referenced Subsystems.....	27
Array-of-Struct Support.....	27
Support of Simulink Functions.....	28
Probe Block.....	28
MATLAB Code.....	28
MATLAB Code Improvements.....	28
Adaptive AUTOSAR.....	29
Enhanced Support for Adaptive AUTOSAR.....	29
Classic AUTOSAR.....	29
Dynamically Accessing Array-of-Struct Variables Specified at Data Stores.....	30
General Enhancements and Changes for Classic AUTOSAR.....	30
AUTOSAR Import and Export Improvements.....	30
Tool Chain Support.....	31
Improved Container Export.....	31
Code Generation Core Functionality.....	33
Unicode.....	33
C99 Support Enhancements.....	33
Floating-Point Limit Handling.....	34
Improved Data Flow Analysis.....	34
Target Simulation (PIL).....	35
Changes in the Target Simulation Modules.....	35
Usability.....	36
Property Manager.....	36
Data Dictionary.....	36
Code Generator Options.....	37
New Code Generator Options.....	37

API Functions and Hook Scripts.....	38
New API Functions.....	38
Other.....	38
General Enhancements and Changes.....	39
TargetLink Demos.....	39
Prior TargetLink Versions.....	41
New Features of TargetLink 5.0.....	41
Modeling in Simulink or Stateflow.....	42
Improved Support of Array-of-Structs.....	42
Data Stores Usable Across Code Generation Units.....	42
N-Dimensional Look-Up Tables.....	43
Other MATLAB/Simulink/Stateflow Features.....	43
MATLAB Code.....	44
MATLAB Code Improvements.....	44
Adaptive AUTOSAR.....	45
Adaptive AUTOSAR.....	45
Classic AUTOSAR.....	45
Supported AUTOSAR Releases.....	46
Improved Support of Data Store Blocks for AUTOSAR Communication.....	46
Improved Import and Export for AUTOSAR 4.4.....	46
Tool Chain Support.....	47
Using Several TargetLink-Generated SIC Files in the Same Application Process.....	47
Code Generation Core Functionality.....	48
Support of Modern Standards of C.....	48
Void casts in function calls.....	49
Generating Code for 64-Bit Linux Targets.....	49
Improved Code Efficiency.....	49
Target Simulation (PIL).....	50
Changes in the Target Simulation Modules.....	50
Usability.....	51
Data Dictionary and Data Dictionary Manager.....	51
Property Manager.....	53
Code Generator Options.....	53
New Code Generator Options.....	53

API Functions and Hook Scripts.....	55
New API Functions.....	55
Improved API Functions and Hook Scripts.....	55
Other.....	56
General Enhancements and Changes.....	56
TargetLink Demos.....	57
New Features of TargetLink 4.4.....	57
Tool Chain Support.....	58
Executing TargetLink Code on dSPACE Real-Time Hardware.....	58
Modeling in Simulink or Stateflow.....	58
Support of Arrays of Struct.....	58
Bus Support of the Constant Block.....	59
Other MATLAB/Simulink/Stateflow Features.....	60
TargetLink Module for MATLAB Code.....	61
Code Generation Core Functionality.....	62
New Value Copy Access Functions.....	62
AUTOSAR.....	63
Supported AUTOSAR Releases.....	63
Support of Provide-Require Ports for Sender-Receiver Communication.....	64
Support of Data Store Blocks for Sender-Receiver Communication.....	64
Support of Explicit NvData Communication.....	64
Support of Rte_IWrite and Rte_IWriteRef in Sender-Receiver and NvData Communication.....	64
Exporting Splittable Elements.....	65
Target Simulation (PIL).....	65
Support for Virtual Evaluation Boards.....	65
Changes in the Target Simulation Modules.....	66
Data Dictionary and Data Management.....	67
Support of Show Masked Content and Show Library Content in the Property Manager.....	67
New Import and Export of View Sets in the Property Manager.....	67
Support for User Mode per Workspace in the Data Dictionary Manager.....	68
Code Generator Options.....	68
New Code Generator Options.....	68

API Functions and Hook Scripts.....	69
New API Functions.....	69
Improved Hook Scripts.....	69
Other.....	70
General Enhancements and Changes.....	70
TargetLink Demos.....	70
Synchronizing Simulink and TargetLink.....	71
New Features of TargetLink 4.3.....	72
Modeling in Simulink or Stateflow.....	72
Newly Supported Simulink Block.....	73
Bus-Capable Custom Code Block (Type II).....	73
Other Simulink/Stateflow Features.....	73
Code Generation Core Functionality.....	74
Code Decorations (Declaration Statements and Section Names).....	74
MISRA C Compliance.....	75
Improved Support of Variable Vector Widths.....	76
Modular Development.....	77
Improved A2L File Generation.....	77
Improved Workflow for Distributed Development.....	77
AUTOSAR.....	78
Supported AUTOSAR Releases.....	78
Memory Mapping.....	79
Static Memories and Constant Memories for Measurement and Calibration.....	79
Support for Rte_IsUpdated.....	80
AUTOSAR Import and Export Improvements.....	80
Target Simulation (PIL).....	81
Changes in the Target Simulation Modules.....	81
Data Dictionary and Data Management.....	82
Further Improvements to the Data Dictionary.....	82
Code Generator Options.....	82
New Code Generator Options.....	82
API Functions and Hook Scripts.....	83
New API Functions.....	83
Other.....	83
New Property Manager.....	83

General Enhancements and Changes.....	85
TargetLink Demos.....	86
New Features of TargetLink 4.2.....	87
Modeling in Simulink or Stateflow.....	87
Newly Supported Simulink Blocks.....	88
Improved Data Store Blocks and Bus Support.....	88
Improvements to Stateflow Code Generation.....	89
Improvements to Model Referencing.....	90
Code Generation Core Functionality.....	90
MISRA C Compliance.....	90
Improved Code Stability.....	91
Improved Code Efficiency.....	91
Improved Enumeration Support.....	95
More Flexibility When Using TargetLink Custom Code Blocks.....	95
AUTOSAR.....	96
Supported AUTOSAR Releases.....	96
Asynchronous Client-Server Communication.....	97
Data Transformer in Client-Server Communication.....	97
Non-Scalar Interrunnable Variables.....	98
Improved Roundtrip with SystemDesk and other AUTOSAR Tools.....	98
Miscellaneous AUTOSAR Features.....	98
Target Simulation (PIL).....	99
High-Speed Baud Rates for PIL Simulations.....	99
Changes in the Target Simulation Modules.....	99
Data Dictionary and Data Management.....	100
Improvements to the Data Dictionary.....	100
Predefined Code Generator Option Sets in Data Dictionaries.....	101
Code Generator Options	102
New Code Generator Options.....	102
Tool Chain Integration.....	102
Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing.....	103
Build Binary Files for Functional Mock-up Units.....	103
Other.....	104
General Enhancements and Changes.....	104
TargetLink Demos.....	106

API Functions and Hook Scripts.....	107
New API Functions.....	108
New Hook Scripts.....	108
New Features of TargetLink 4.1.....	109
Modeling in Simulink or Stateflow.....	109
Newly Supported Simulink Blocks.....	109
Improvements to Custom Look-up Tables.....	110
Support of Simulink's Simplified Mode and IC Structures.....	110
Support of Structures in Stateflow Action Language.....	111
Code Generation Core Functionality.....	111
MISRA-C Compliance.....	111
Improved Code Efficiency.....	112
Component-Based Development.....	116
Improvements to Function Reuse.....	116
AUTOSAR.....	117
Supported AUTOSAR Releases.....	117
Support of NvData Communication.....	118
Data Transformation.....	118
Activation Reasons of Runnables.....	118
Port-Defined Argument Values.....	119
Miscellaneous AUTOSAR Features.....	119
Target Simulation (PIL).....	120
Changes in the Target Simulation Modules.....	120
Folder for TSM Extensions.....	122
Data Dictionary and Data Management.....	122
Improvements to the Data Dictionary.....	123
New DD MATLAB API Functions.....	124
Referencing DD CodegenOptions Objects at TargetLink Main Dialog Block.....	124
Code Generator Options.....	125
New Code Generator Options.....	125
Tool Chain Integration.....	126
Exporting Functional Mock-up Units.....	126
Requirement Information in the Data Dictionary.....	126
Other.....	127
General Enhancements and Changes.....	127
TargetLink Demos.....	129

API Functions and Hook Functions.....	131
New API Functions.....	131
New Hook Functions.....	132
New Features of TargetLink 4.0.....	133
Modeling in Simulink or Stateflow.....	133
Support of Matrix Signals.....	134
Newly Supported Simulink Blocks.....	134
Improved Bus Support.....	134
Dynamic Look-Up Tables.....	135
Improvements to TargetLink's Simulation Frame.....	135
Improvements to Scaling-Invariant Systems.....	136
Additionally Supported Block Properties.....	136
Central Specification of Function Subsystem Signatures.....	137
Code Generation Core Functionality.....	139
MISRA-C Compliance.....	139
Improved Code Efficiency.....	140
Data Dictionary and Data Management.....	141
Improvements to the Data Dictionary.....	141
New DD MATLAB API Commands.....	143
AUTOSAR.....	146
Supported AUTOSAR Releases.....	146
New AUTOSAR Features.....	147
Test Support.....	149
Improved Online Parameter Modification.....	149
Changes in the Target Simulation Modules.....	149
Code Generator Options.....	151
New Code Generator Options.....	151
Tool Chain Integration.....	152
Improved Windows Compliance.....	152
Other.....	153
General Enhancements and Changes.....	153
API Commands.....	155
New API Functions.....	155
New Features of TargetLink 3.5.....	156
Modeling in Simulink or Stateflow.....	157
Newly Supported Simulink Blocks.....	157
Enumeration Support.....	157

Improved System Preparation.....	158
Improved Stateflow Support.....	159
Code Generation Core Functionality.....	159
MISRA-C:2004 compliance.....	160
Improved Code Efficiency.....	160
Data Dictionary and Datamanagement.....	160
Enhanced Usability of the Data Dictionary Manager.....	161
Further Improvements to the Data Dictionary.....	163
New DD MATLAB API Commands.....	164
AUTOSAR.....	164
Supported AUTOSAR Releases.....	165
Multiple Instantiation of Software Components.....	165
Per Instance Calibratable Parameters.....	166
Support of Code Generation Units Containing Only Operation Calls.....	166
Easier Specification of AUTOSAR Objects in the Data Dictionary Manager.....	167
Improved Restart Function Behavior.....	168
Improved Container Management.....	168
Testing Support.....	169
Signal Injection/Tunneling.....	169
Third-Party Tool Support for Code Coverage Measurements.....	170
Enhanced Overflow Detection.....	170
Changes in the Target Simulation Modules.....	170
Code Generator Options.....	172
Code Generator Options.....	172
Toolchain Integration.....	175
Handling Customization Files.....	175
Enhanced Handling and Restoring of Demo Models.....	175
TargetLink Simulation Module Extension Packages.....	176
Documentation.....	177
Improved Documentation.....	177
Other.....	178
New Code Generation Report.....	178
General Enhancements and Changes.....	179
API Commands.....	181
New API Commands.....	181
Hook Functions.....	182
New Hook Functions.....	182

New Features of TargetLink 3.4.....	182
New Production Code Generation Features.....	183
Component-Based Development Using Abstract Interfaces.....	183
Improvement for Custom Code.....	184
Optional Deactivation of Compute-Through-Overflow.....	184
Generating Virtual ECUs for Virtual ECU Testing.....	185
Specifications of the Target Simulation Modules.....	186
Code Generator Options.....	189
New API Commands.....	194
New Hook Functions.....	194
Access Function Changes.....	195
Improvement of Code Efficiency.....	195
General Enhancements and Changes.....	196
New AUTOSAR-Related Features.....	197
Features of the TargetLink AUTOSAR Module.....	197
New TargetLink Data Dictionary Features.....	202
Compare and Merge Features.....	202
Improvements for Data Dictionary File Handling.....	203
Further Improvements of the Data Dictionary Manager.....	205
New DD MATLAB API Commands.....	206
New Features of TargetLink 3.3.....	207
New Production Code Generation Features.....	208
64-Bit Version of TargetLink.....	208
Function Reuse.....	209
Improvements for Incremental Code Generation and Model Referencing.....	210
Lifetime Analysis and Scope Reduction.....	210
Generic Multirate Code Generation.....	211
Using Data Store Memory Variables in Stateflow.....	212
Block Data Clipboard.....	212
Getting and Setting Block Properties via Structures.....	213
Simplified Referencing of Values from Data Dictionary.....	214
Native Autodoc Customization Block.....	214
BlackBox Mask Type.....	215
Enhancements to the Target Simulation Module.....	215
New TargetLink API commands.....	219
Code Generator Options.....	220
Improvement for Online Parameter Modification.....	221
Usability Improvements.....	222

General Enhancements and Changes.....	224
Testing Improvements.....	225
New AUTOSAR-Related Features.....	226
Features of the TargetLink AUTOSAR Module.....	226
New dSPACE Data Dictionary Features.....	232
New Data Dictionary Key Features.....	232
New DD MATLAB API Commands.....	236
New Features of TargetLink 3.2.....	238
New Production Code Generation Features.....	238
Online Parameter Modification.....	238
Debugging in SIL Simulation Mode.....	241
New TargetLink Sqrt Block.....	241
Enhancements to the Target Simulation Module.....	242
New TargetLink API commands.....	243
Code Generator Options.....	244
General Enhancements and Changes.....	245
New AUTOSAR-Related Features.....	248
Features of the TargetLink AUTOSAR Module.....	248
Exchanging Software Component Containers with SystemDesk.....	250
New Features of the dSPACE Data Dictionary.....	253
New Key Features.....	253
New and Modified DD MATLAB API Commands.....	259
New Features of TargetLink 3.1.....	259
New Production Code Generation Features.....	260
Code Generation from the dSPACE Data Dictionary.....	260
New TargetLink Bit Operation Blocks.....	263
Enhancements to the Target Simulation Module.....	264
Requirements Traceability down to Generated Code and Documentation.....	267
Code Generation with Variable Vector Widths.....	268
Enhancements and Changes to the TargetLink Main Dialog.....	268
Code Generator Options.....	272
General Enhancements and Changes.....	274
New AUTOSAR-Related Features.....	278
Features of the TargetLink AUTOSAR Module.....	278
New Features of the dSPACE Data Dictionary.....	282
New Key Features.....	283
New and Modified DD MATLAB API Commands.....	284

General Migration Information	285
Upgrading Models, Libraries, and Data Dictionaries.....	286
Basics on Migrating Between TargetLink Versions.....	286
How to Upgrade a Data Dictionary with Included DD Files.....	289
How to Manually Upgrade Libraries and Models via the API.....	290
Migrating Data Dictionaries to CodeDecorationSets.....	291
 Migration Steps	 295
Migrating from TargetLink 5.0 to 5.1.....	296
Code Generator Options.....	296
Migration Aspects Regarding Code Generator Options.....	296
AUTOSAR.....	298
Migration Aspects Regarding AUTOSAR.....	298
Custom Code.....	299
Migration Aspects Regarding Custom Code.....	299
API Functions and Hook Scripts.....	299
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	299
Messages.....	300
Changes in TargetLink Messages.....	300
Reserved Identifiers.....	301
Migration Aspects regarding reserved identifiers.....	301
Other Migration Aspects.....	302
Various Migration Aspects.....	302
Optimization.....	303
Migration Aspects Regarding Optimization.....	303
Migrating from TargetLink 4.4 to 5.0.....	304
Code Generator Options.....	304
Migration Aspects Regarding Code Generator Options.....	304
AUTOSAR.....	307
AUTOSAR 2.x/3.x no Longer Supported.....	307
Changed Block Property for AUTOSAR Mode.....	308
Custom Code.....	308
Dereferencing of Pointers.....	308

API Functions and Hook Scripts.....	309
Changes in API Functions.....	309
Messages.....	310
Message changes.....	310
Other Migration Aspects.....	311
Various Migration Aspects.....	311
Migrating from TargetLink 4.3 to 4.4.....	314
Code Generator Options.....	314
Migration Aspects Regarding Code Generator Options.....	314
API Functions and Hook Scripts.....	316
Changes in API Functions.....	316
Messages.....	317
Message changes.....	317
Other Migration Aspects.....	317
Various Migration Aspects.....	317
Migrating from TargetLink 4.2 to 4.3.....	318
API Functions and Hook Scripts.....	318
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	318
Code Generator Options.....	319
Migration Aspects Regarding Code Generator Options.....	319
MATLAB-Related Changes.....	321
Modified features in MATLAB R2016b.....	321
Other.....	322
Property Manager - Migration Aspects.....	322
Various Migration Aspects.....	324
Migrating from TargetLink 4.1 to 4.2.....	328
API Functions and Hook Scripts.....	328
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	328
AUTOSAR-Related Migration Aspects.....	330
Migrating Interrunnable Variable Specifications.....	330
Migrating Transformer Error Code Specifications.....	332
Code Generator Options.....	333
Migration Aspects Regarding Code Generator Options.....	333
MATLAB-Related Changes.....	335
Modified features in MATLAB R2016b.....	335

Other.....	336
Various Migration Aspects.....	336
Migrating from TargetLink 4.0 to 4.1.....	337
API Functions and Hook Scripts.....	337
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	337
AUTOSAR-Related Migration Aspects.....	338
AUTOSAR-Related Migration Aspects.....	338
Code Generator Options.....	339
Migration Aspects Regarding Code Generator Options.....	339
Other.....	341
Various Migration Aspects.....	341
Migrating from TargetLink 3.5 to 4.0.....	343
API Functions.....	343
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	343
AUTOSAR-Related Migration Aspects.....	344
2-D Matrix Data Elements and Operation Arguments.....	344
Application Data Types.....	345
AUTOSAR Export.....	345
Constraints of Array and Matrix Types.....	346
Replaced IsQueued Property.....	346
Separate Installation of Container Manager.....	347
Changes With Access Functions.....	347
Changes of ADDRESS_BY_PARAMETER Access Functions.....	348
New Access-Function-Specific Name Macro.....	348
New Default Macro Bodies for Macro Access Functions.....	349
Code Generator Options.....	350
Migration Aspects Regarding Code Generator Options.....	350
Messages.....	351
Message Changes.....	351
Other.....	352
Plot Channel Specification.....	352
Signal Properties Inheritance.....	353
Stricter Settings for Bus Diagnostics.....	353
Various Migration Aspects.....	355
Stateflow-Related Changes.....	357
Exported Graphical Functions.....	357
Stateflow Matrices.....	357

Migrating from TargetLink 3.4 to 3.5.....	359
API Functions and Hook Scripts.....	359
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	359
Hook Script Templates.....	359
AUTOSAR-Related Migration Aspects.....	360
AUTOSAR-Related Migration Aspects.....	360
Documentation.....	361
Documentation Changes.....	361
Other.....	363
Various Migration Aspects.....	363
Migrating from TargetLink 3.3 to 3.4.....	365
API Functions.....	365
Changes in TargetLink and TargetLink Data Dictionary API Functions.....	365
AUTOSAR-Related Migration Aspects.....	365
AUTOSAR-Related Migration Aspects.....	365
Code Generator Options.....	366
Migration Aspects Regarding Code Generator Options.....	366
Other.....	367
Various Migration Aspects.....	367
Migrating from TargetLink 3.2 to 3.3.....	368
API Functions.....	368
Changes in TargetLink and dSPACE Data Dictionary API Functions.....	368
AUTOSAR-Related Migration Aspects.....	370
AUTOSAR-Related Migration Aspects.....	370
Code Generator Options.....	370
Migration Aspects Regarding Code Generator Options.....	370
Other.....	372
Various Migration Aspects.....	372
Migrating from TargetLink 3.1 to 3.2.....	376
API Functions.....	376
Changes in TargetLink API Functions.....	376
AUTOSAR-Related Migration Aspects.....	378
AUTOSAR-Related Migration Aspects.....	378
Code Generator Options.....	380
Migration Aspects Regarding Code Generator Options.....	380

Other.....	383
Various Migration Aspects.....	383
Migrating from TargetLink 3.0 to 3.1.....	387
API Functions.....	387
Modified DD MATLAB API Functions.....	387
Changes in TargetLink API Functions.....	388
AUTOSAR-Related Migration Aspects.....	388
AUTOSAR-Related Migration Aspects.....	388
Code Generator Options.....	391
Migration Aspects Regarding Code Generator Options.....	391
General Migration Aspects.....	392
General Migration Aspects.....	392
Other.....	395
Migrating ClientPort Blocks.....	395
Various Migration Aspects.....	396
Discontinuations	401
Discontinuations as of TargetLink 5.1.....	402
Discontinued TargetLink Features.....	402
Obsolete Limitations.....	402
Discontinuations as of TargetLink 5.0.....	403
Discontinued TargetLink Features	403
Obsolete API Functions.....	404
Obsolete Limitations.....	405
Discontinuations as of TargetLink 4.4.....	406
Discontinued TargetLink Features	406
Obsolete API Functions.....	406
Obsolete Code Generator Options.....	406
Obsolete Limitations.....	407
Discontinuations as of TargetLink 4.3.....	409
Discontinued TargetLink Features	409
Obsolete API Functions.....	409
Obsolete Limitations.....	410
Discontinuations as of TargetLink 4.2.....	411
Discontinued TargetLink Features	411
Obsolete API Functions.....	412

Obsolete Code Generator Options.....	412
Obsolete Limitations.....	413
Discontinuations as of TargetLink 4.1.....	415
Obsolete Code Generator Options.....	415
Obsolete Limitations.....	416
Discontinuations as of TargetLink 4.0.....	418
Discontinued TargetLink Features	418
Discontinued Data Dictionary Features	418
Obsolete API Functions.....	419
Obsolete Code Generator Options.....	420
Obsolete Limitations.....	420
Discontinuations as of TargetLink 3.5.....	422
Discontinued Data Dictionary Features	422
Obsolete Limitations.....	423
Discontinuations as of TargetLink 3.4.....	424
Discontinued Data Dictionary Features	424
Obsolete Limitations.....	424
Discontinuations as of TargetLink 3.3.....	426
Discontinued Data Dictionary Features	426
Obsolete Code Generator Options.....	426
Obsolete Limitations.....	427
Discontinuations as of TargetLink 3.2.....	429
Obsolete Code Generator Options.....	429
Discontinuations as of TargetLink 3.1.....	430
Obsolete Code Generator Options.....	430
Code Changes	431
Code Changes Between TargetLink 5.0 and TargetLink 5.1.....	432
AUTOSAR.....	432
Auxiliary Variables.....	433
Basetype-Related Code Changes.....	434
Comment-Related Code Changes.....	435
Control Flows.....	437
Custom Code.....	438
Function Inlining.....	439
Interprocedural Analysis.....	439
Initialization.....	446
Logical Expressions.....	448
Range Propagation.....	449

Shift Operations.....	450
Stateflow and MATLAB Code.....	450
Variable Optimization.....	452
Variable Scope.....	453
Other.....	453
Code Changes Between TargetLink 4.4 and TargetLink 5.0.....	456
Shift Operations.....	456
Extern C Encapsulation.....	459
Access Functions.....	459
AUTOSAR.....	460
Global Bitfields.....	461
MATLAB Code.....	461
Name Template of DD VariableTemplate Objects.....	462
Look-Up Tables.....	462
Variable Vector Width.....	463
Stateflow.....	464
Other.....	466
Code Changes Between TargetLink 4.3 and TargetLink 4.4.....	470
Access Functions.....	470
AUTOSAR.....	473
Block-Specific Code Changes.....	473
Efficiency.....	478
Inheritance of Struct Data Types.....	479
Mixed Operations (Floating-Point and Fixed-Point Types).....	481
Stateflow.....	482
Other.....	483
Code Changes between TargetLink 4.2 and TargetLink 4.3.....	487
64-Bit Multiplication.....	487
AUTOSAR.....	488
Efficiency.....	491
Function Reuse.....	494
MISRA Compliance.....	495
Mixed Operations (Floating-Point and Fixed-Point Types).....	496
State Reset.....	498
Other.....	504
Code Changes between TargetLink 4.1 and TargetLink 4.2.....	510
Code Changes between TargetLink 4.1 and TargetLink 4.2.....	510
Code Changes between TargetLink 4.0 and TargetLink 4.1.....	521
Code Changes between TargetLink 4.0 and TargetLink 4.1.....	521

Code Changes between TargetLink 3.5 and TargetLink 4.0.....	535
Code Changes between TargetLink 3.5 and TargetLink 4.0.....	535
Code Changes between TargetLink 3.4 and TargetLink 3.5.....	547
Code Changes between TargetLink 3.4 and TargetLink 3.5.....	547
Code Changes between TargetLink 3.3 and TargetLink 3.4.....	551
Code Changes between TargetLink 3.3 and TargetLink 3.4.....	551
Changes in Future TargetLink Versions	557
Features to Be Discontinued.....	557
API Functions to Be Discontinued.....	558
Deprecated Code Generator Options.....	558
Glossary	559
Index	589

About This Guide

Contents

This guide contains an overview of new features and information on migration of the current and previous TargetLink releases. All the information on the current TargetLink release is identical to the information provided in the New Features and Migration Guide (refer to  [New Features and Migration](#)).

To migrate libraries or models from TargetLink versions earlier than 4.2, you also have to perform the migration steps of the TargetLink versions in between. For detailed information, refer to [Migration Steps](#) on page 295.

Orientation and Overview Guide For an introduction to the use cases and the TargetLink features that are related to them, refer to the  [TargetLink Orientation and Overview Guide](#).

Required knowledge

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

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

New Features

Where to go from here

Information in this section

Current TargetLink Version.....	26
Prior TargetLink Versions.....	41

Current TargetLink Version

New Features of TargetLink 5.1

Where to go from here

Information in this section

Modeling in Simulink or Stateflow.....	26
MATLAB Code.....	28
Adaptive AUTOSAR.....	29
Classic AUTOSAR.....	29
Tool Chain Support.....	31
Code Generation Core Functionality.....	33
Target Simulation (PIL).....	35
Usability.....	36
Code Generator Options.....	37
API Functions and Hook Scripts.....	38
Other.....	38

Modeling in Simulink or Stateflow

Where to go from here

Information in this section

Referenced Subsystems.....	27
Array-of-Struct Support.....	27
Support of Simulink Functions.....	28
Probe Block.....	28

Referenced Subsystems

Referenced subsystems

TargetLink 5.1 supports Simulink® Subsystem Reference. You can use Subsystem Reference for component-based modeling to reference a reusable group of blocks.

Via Subsystem Reference you can save the contents of a subsystem in a separate SLX file. You can reference this subsystem by using a Subsystem Reference block.

Related documentation

- [How to Make Referenced Subsystems TargetLink-Compliant \(TargetLink Preparation and Simulation Guide\)](#)

Related topics

HowTos

[How to Make Referenced Subsystems TargetLink-Compliant \(TargetLink Preparation and Simulation Guide\)](#)

Array-of-Struct Support

SIL simulation

TargetLink 5.1 supports the access of array-of-struct variables during simulation via signal injection and signal tunneling.

Related documentation

- [Basics on Injecting or Tunneling Signals During Simulation \(TargetLink Preparation and Simulation Guide\)](#)

Dynamic access via Custom Code (type II) block

TargetLink 5.1 supports the dynamic access to array-of-struct variables that are specified at data stores via Custom Code (type II) blocks.

Related documentation

- [Example of Dynamically Accessing Array-of-Struct Variables via Data Store Blocks and Custom Code \(Type II\) Blocks \(TargetLink Preparation and Simulation Guide\)](#)

Support of Simulink Functions

You can now define global Simulink Function subsystems in a TargetLink subsystem that can be called from Function Caller blocks in the same code generation unit.

Related documentation [Working With Simulink Function Subsystems in TargetLink](#) ([TargetLink Preparation and Simulation Guide](#))

Probe Block

Newly supported TargetLink simulation block: Probe block

TargetLink now supports the Probe block to output the width and/or signal dimensions of the input signal.

Related documentation

- [Probe Block Functionality](#) ([TargetLink Model Element Reference](#))
- [Probe block](#) ([TargetLink Limitation Reference](#))

MATLAB Code

MATLAB Code Improvements

Constant variables in MATLAB code

TargetLink 5.1 lets you use constant variables in MATLAB code to increase code readability. In addition, you can declare a constant [local MATLAB variable](#) as a macro or constant C variable.

Related documentation

- [Basics on Constant Variables in MATLAB® Code](#) ([TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models](#))

Related topics

Basics

[Basics on Constant Variables in MATLAB® Code](#) ([TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models](#))

Adaptive AUTOSAR

Enhanced Support for Adaptive AUTOSAR

Enhanced Adaptive AUTOSAR support

TargetLink 5.1 supports additional features of Adaptive AUTOSAR.

The following additional features are supported:

- Import of Adaptive AUTOSAR ARXMLs with elements defined by ara::per
- Modeling of select parts of a service-based communication as described by ara::com:
 - Sending and receiving events as defined in Adaptive AUTOSAR release 18-10.
 - SIL simulation of Adaptive AUTOSAR components.
- Modeling of select parts of access to persistent memory as described by ara::per:
 - Read and write access to key-value pairs with an AdaptivePlatformType from key-value storages via Data Store blocks.

A demo model that showcases the different modeling styles for Adaptive AUTOSAR communication is provided. Refer to [AAR_COMMUNICATION](#) ([TargetLink Demo Models](#)).

Additionally, a new guide for the user documentation is available. Refer to [TargetLink Adaptive AUTOSAR Modeling Guide](#).

Related documentation

- [TargetLink Adaptive AUTOSAR Modeling Guide](#)
- [AAR_COMMUNICATION](#) ([TargetLink Demo Models](#))

Classic AUTOSAR

Where to go from here

Information in this section

Dynamically Accessing Array-of-Struct Variables Specified at Data Stores.....	30
General Enhancements and Changes for Classic AUTOSAR.....	30
AUTOSAR Import and Export Improvements.....	30

Dynamically Accessing Array-of-Struct Variables Specified at Data Stores

TargetLink now lets you dynamically access array-of-struct variables that are specified at data stores via Custom Code (type II) blocks.

This is possible for the following Classic AUTOSAR data prototypes:

- Data elements
- Interrunnable variables
- NvData elements
- Static memories

Related documentation

- [Dynamically Accessing AUTOSAR Data Stores via Custom Code \(Type II\) Blocks](#)
([TargetLink Classic AUTOSAR Modeling Guide](#))
- [AR_ARRAY_OF_STRUCT_DATA](#) ([TargetLink Demo Models](#))

General Enhancements and Changes for Classic AUTOSAR

KPR.2014.09.23.004 resolved

With TargetLink 5.1, the known problem report KPR.2014.09.23.004 is resolved.

It is now possible to compile the production code for AUTOSAR models that use the fixed-point library regardless of the module names used.

The handling of data types used for the fixed-point library has changed. As a result, the `t1_types.h` header file is no longer required and thus no longer generated.

Related documentation

- [Migration Aspects Regarding Custom Code](#) on page 299

AUTOSAR Import and Export Improvements

Support of new data types

TargetLink now lets you import ARXML files that use the following data types into the Data Dictionary:

- `Int64` and `UInt64` whose encoding is `None` or `2C`
- Data types whose Category is `DATA_REFERENCE` (Pointer)
- Optimized integer data types as described in the `Platform_Types` header

The export of all these data types is also supported.

Note

You cannot use these data types in code generation.

Look-up-table-related elements

TargetLink now lets you import and export the following:

- Primitive application data types of the categories CURVE, MAP, and COM_AXIS.
- Elements that are related to the calibration of look-up tables.

You can use these when modeling look-up tables for AUTOSAR and to prepare measurement and calibration.

Related documentation

- [Basics on Preparing Look-Up Tables for Measurement and Calibration \(Classic AUTOSAR\)](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))
- [Example of Preparing a Look-Up Table for Measurement and Calibration](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))

Tool Chain Support

Improved Container Export

Improvements to FMU containers

With TargetLink 5.1, the following improvements to FMU containers are available:

- FMU containers that include the following can be generated:
 - Source code only
 - Binaries only (new)
 - Source code and binaries
- Binaries for the Linux 64-bit platform can be built. This requires a suitable compiler, which you must provide.
- By default, the source code of the generated FMUs is compatible with the following platforms:
 - If system files are not included: all 32-bit and 64-bit platforms (QNX, Windows and Linux)
 - If system files are included: all 32-bit and 64-bit platforms (QNX, Windows and Linux) with little endian byte order

Related documentation

- [Basics on Exporting FMUs from TargetLink](#) ([TargetLink Interoperation and Exchange Guide](#))
 - [tl_generate_fmu](#)
 - [TargetLink FMU Manager](#) ([TargetLink Tool and Utility Reference](#))
-

SIC compatibility

With TargetLink 5.1, the SIC files generated by TargetLink are compatible with the following dSPACE platforms:

- Host PC: Windows 32-bit, Windows 64-bit and Linux 64-bit (VEOS)
- SCALEXIO: QNX/Linux 32-bit (ConfigurationDesk)
- MicroAutoBox III: Linux 32-bit (ConfigurationDesk)

Related documentation

- [TargetLink SIC Manager](#) ([TargetLink Tool and Utility Reference](#))
 - [tlGenerateSic](#) ([TargetLink API Reference](#))
-

V-ECU implementation container

With TargetLink 5.1, the V-ECU containers generated by TargetLink comply with V-ECU Version 3.0 and are compatible with the following dSPACE platforms:

- Host PC: Windows 32-bit, Windows 64-bit and Linux 64-bit (VEOS)
- SCALEXIO: QNX/Linux 32-bit (ConfigurationDesk)
- MicroAutoBox III: Linux 32-bit (ConfigurationDesk)

Related documentation

- [Basics on Interoperating with Other dSPACE Tools for Virtual Validation](#) ([TargetLink Interoperation and Exchange Guide](#))
 - [tl_generate_vecu_implementation](#)
-

V-ECU implementation with fixed-point library

With TargetLink 5.1, the dSPACE fixed-point library is included in a V-ECU implementation container if required. Therefore, it is no longer necessary to export the library as a ZIP archive when using V-ECU implementation containers in ConfigurationDesk.

Related documentation

- [TargetLink V-ECU Manager](#) ([TargetLink Tool and Utility Reference](#))
 - [tl_generate_vecu_implementation](#)
-

Related topics**References**

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

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

Code Generation Core Functionality

Where to go from here

Information in this section

Unicode.....	33
C99 Support Enhancements.....	33
Floating-Point Limit Handling.....	34
Improved Data Flow Analysis.....	34

Unicode

Unicode support

TargetLink 5.1 supports Unicode in the following areas:

- Descriptions in Simulink blocks.
- Descriptions in TargetLink blocks.
- Descriptions in TargetLink Data Dictionary entries.
- Comments in A2L files.
- Descriptions in AUTOSAR XML files.

You can specify a character encoding via the **CharacterSet** property in the Data Dictionary Manager. With TargetLink 5.1 the default is changed from **LocalDefault** to **UTF-8**.

Related documentation

- Overview of the Supported Character Sets ([TargetLink Interoperation and Exchange Guide](#))

C99 Support Enhancements

Fixed-width integer data types as defined by C99

TargetLink 5.1 supports fixed-width integer data types with exact widths as defined by the language standard C99. This support includes signed and unsigned types. You can use the data types via the **Generic C99 Code Generation** target.

Related documentation

- Basics on Generating Code Compliant with C99/C11/C++ ([TargetLink Customization and Optimization Guide](#))

Floating-Point Limit Handling

Macros and DD objects for floating-point limits

With TargetLink 5.1, floating-point type limits are handled via macros in the generated code. These macros are defined in separate, platform-specific header files that can be automatically generated again if required. This ensures that the generated code that uses the floating-point limits can be compiled for different platforms.

TargetLink now provides predefined DD Variable objects for the floating-point limits of the `Float32` and `Float64` base types. These objects can be used to model the use of floating-point limits.

Related documentation

- [Basics on Floating-Point Limits in TargetLink \(TargetLink Preparation and Simulation Guide\)](#)

Improved Data Flow Analysis

Introduction

TargetLink performs optimizations based on data flow analysis. Examples are:

- Elimination of intermediate variables.
- Scope reduction of variables.
- Removal of unnecessary computations.
- Moving code into conditionally executed control flow branches.

Improved data flow analysis

With TargetLink 5.1, data flow analysis is improved in the following ways:

- Consistent treatment of the influence of the DD VariableClass object's Optimization property values, especially `MOVABLE`.
- Consistent treatment of variables and functions accessible across code generation unit (CGU) boundaries.
- Consistent scheduling of less-used custom-code-section code.
- Scope reduction of variables used in the initialization of other variables.
- Interprocedural data flow analysis for cyclic function call relationships.
- Increased analysis depth for interprocedural data flow analysis.
- Utilization of data flow properties of arguments of function calls and the newly introduced user assertions for Data Store variables of the Custom Code (type II) block.
- Improved utilization of alias relationships for accesses via pointer dereference.

This leads to several major and minor changes in generated code, usually contributing either to code efficiency or matching user expectations with regard to consistent optimization behavior or to correct code with respect to data flow across function, CGU, and external functionality borders. For details, see

[Interprocedural Analysis](#) on page 439 in [Code Changes Between TargetLink 5.0 and TargetLink 5.1](#) on page 432.

Target Simulation (PIL)

Changes in the Target Simulation Modules

New and discontinued compiler versions

The following table shows the compiler versions that are now supported by TargetLink 5.1. Refer to the New and No changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.

Microcontroller Family	Compiler	New	No Changes	Discontinued
ARM CortexM3	Keil	—	5.2	—
C16x	TASKING	—	8.6	—
MPC57xxVLE	Diab	—	5.9	—
	GreenHill	—	2019	—
RH850	GreenHill	—	2019	—
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3	—
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	—	3.2	—
TriCore2xx	TASKING	—	6.3	—
	GCC	—	4.9	—
TMS570 (ARM)	CCS	—	7.0	—
V850	GreenHill	—	2019	—
XC22xx	TASKING	—	3.0	—

For more information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ( [Evaluation Board Reference](#)).

Note

For more PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to the dSPACE [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Usability

Where to go from here

Information in this section

Property Manager.....	36
Data Dictionary.....	36

Property Manager

Usability improvements

TargetLink 5.1 provides several usability improvements in the Property Manager, for example:

- It is now possible to use the breadcrumb navigation to copy, paste, or edit the path of a model element.
- There are several improvements concerning the display of hints and tooltips in the Property Manager.

Related documentation

- [Display of Validation Errors](#) ([TargetLink Preparation and Simulation Guide](#))
- [Breadcrumb](#) ([TargetLink Tool and Utility Reference](#))
- [Reset View Set](#) ([TargetLink Tool and Utility Reference](#))
- [Search \(Property View\)](#) ([TargetLink Tool and Utility Reference](#))

Data Dictionary

Usability improvements

TargetLink 5.1 provides several usability improvements in the Data Dictionary Manager. For example:

- In the [Edit Matrix](#) dialog, you can now use the keys and key combinations known from spreadsheet applications.
- Properties containing a list of file paths can now be edited in the [StringList](#) dialog.
- An improved design of the property value list improves orientation and ease of use.

Related documentation

- [Edit Matrix](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Edit StringList](#) ([TargetLink Data Dictionary Manager Reference](#))

DD three-way merge

TargetLink 5.1 provides a DD three-way merge that can compare and merge two different versions of a DD file derived from a common ancestor.

Related documentation

- [Basics on Comparing and Merging DD Files via DD Three-Way Merge](#)
([TargetLink Data Dictionary Basic Concepts Guide](#))

Code Generator Options

New Code Generator Options

Overview of new Code Generator options

The following new Code Generator options are available with TargetLink 5.1:

- **DefaultCppHeaderFileExtension**
Lets you define the default header file name extension for modules whose programming language is set to C++. You can set the programming language of a module via the `Language` property of the DD `ModuleInfo` object.
- **DefaultCppSourceFileExtension**
Lets you define the default source file name extension for modules whose programming language is set to C++. You can set the programming language of a module via the `Language` property of the DD `ModuleInfo` object.
- **UsePlatformTypeAbstractionLayer**
Lets you generate an additional, central header file for mapping `typedef` names to platform data types.
- **MapTypeLimitMacrosToTargetConfigurationValues**
Lets you use values that are defined in the `TargetConfig.xml` for the macros for floating-point limits in the production code.
- **ReportDataStoreUserAssertionsForCustomCode**
Lets you generate a report that lists the user assertions specified for a Data Store access via Custom Code (type II) block.
- **AssumePossibleSideEffectsForReadAccessToNonErasableVariables**
Lets you specify whether TargetLink has to assume potential side effects on read accesses to non-ERASABLE variables.

Related documentation

- [DefaultCppHeaderFileExtension](#) ([TargetLink Model Element Reference](#))
- [DefaultCppSourceFileExtension](#) ([TargetLink Model Element Reference](#))
- [UsePlatformTypeAbstractionLayer](#) ([TargetLink Model Element Reference](#))
- [MapTypeLimitMacrosToTargetConfigurationValues](#) ([TargetLink Model Element Reference](#))

- [ReportDataStoreUserAssertionsForCustomCode](#) (TargetLink Model Element Reference)
- [AssumePossibleSideEffectsForReadAccessToNonErasableVariables](#) (TargetLink Model Element Reference)

For reference information on all Code Generator options, refer to [Alphabetical List of Code Generator Options](#) (TargetLink Model Element Reference).

Migration aspects of Code Generator options

For more information, refer to [Migration Aspects Regarding Code Generator Options](#) on page 296.

API Functions and Hook Scripts

New API Functions

List of new API functions	API Function	Purpose
	tIPromoteProperty	Provides TargetLink properties in TargetLink block masks for Simulink® promote mechanism. Refer to Basics on Adding Mask Parameters for TargetLink Block Properties (TargetLink Preparation and Simulation Guide).

Related topics

References

[tIPromoteProperty](#) (TargetLink API Reference)

Other

Where to go from here

Information in this section

General Enhancements and Changes.....	39
TargetLink Demos.....	39

General Enhancements and Changes

Programming language of a module

With TargetLink 5.1, you can set the programming language of a module via the Language property of the DD ModuleInfo object. By default, the programming language depends on the code generation mode. For Adaptive AUTOSAR, it is C++, otherwise C.

Related documentation

- [Basics on Generating Code Compliant with C99/C11/C++ \(TargetLink Customization and Optimization Guide\)](#)

Adding mask parameters for TargetLink block properties

With TargetLink 5.1, you can provide mask parameters at TargetLink blocks for the Simulink® promote mechanism..

Related documentation

- [Basics on Adding Mask Parameters for TargetLink Block Properties \(TargetLink Preparation and Simulation Guide\)](#)

Dynamically assigning new values to data store variables

With TargetLink 5.1, TargetLink can detect block patterns that are used to dynamically assign new values to data store variables.

Related documentation

- [Basics on Dynamically Assigning New Values to Data Store Variables \(TargetLink Preparation and Simulation Guide\)](#)

Support of Docker for Windows

With TargetLink 5.1, TargetLink can be used for tasks without user interaction on a Docker for Windows container.

For more information, contact dSPACE Support (www.dspace.com/go/supportrequest) for more information.

TargetLink Demos

New demo models

New Demo Model	Description
AAR_COMMUNICATION (TargetLink Demo Models)	The model shows you how to model communication according to Adaptive AUTOSAR.
ENUM_BLINKER (TargetLink Demo Models)	The model shows you how to implement a turn signal by using Simulink enumerations.

Modified demo models	Changed Demo Model	Description
	AR_COLLISION_DETECTION (TargetLink Demo Models)	The model is extended with a model part concerning Rte_result error handling.
	AR_ARRAY_OF_STRUCT_DATA (TargetLink Demo Models)	The model now uses a generic modeling pattern which allows TargetLink to generate direct accesses to the RTE variables, or the correct RTE API functions and access points.
	PRELOOKUP_USR (TargetLink Demo Models)	The model is extended with the specification of table data via table input ports.

Prior TargetLink Versions

Where to go from here

Information in this section

New Features of TargetLink 5.0.....	41
New Features of TargetLink 4.4.....	57
New Features of TargetLink 4.3.....	72
New Features of TargetLink 4.2.....	87
New Features of TargetLink 4.1.....	109
New Features of TargetLink 4.0.....	133
New Features of TargetLink 3.5.....	156
New Features of TargetLink 3.4.....	182
New Features of TargetLink 3.3.....	207
New Features of TargetLink 3.2.....	238
New Features of TargetLink 3.1.....	259

New Features of TargetLink 5.0

Where to go from here

Information in this section

Modeling in Simulink or Stateflow.....	42
MATLAB Code.....	44
Adaptive AUTOSAR.....	45
Classic AUTOSAR.....	45
Tool Chain Support.....	47
Code Generation Core Functionality.....	48
Target Simulation (PIL).....	50
Usability.....	51
Code Generator Options.....	53
API Functions and Hook Scripts.....	55

Other.....	56
------------	----

Modeling in Simulink or Stateflow

Where to go from here

Information in this section

Improved Support of Array-of-Structs.....	42
Data Stores Usable Across Code Generation Units.....	42
N-Dimensional Look-Up Tables.....	43
Other MATLAB/Simulink/Stateflow Features.....	43

Improved Support of Array-of-Structs

Element-wise access to array-of-structs variables via Data Store Read/Write blocks

TargetLink Data Store Memory blocks can reference DD Variable objects holding array-of-structs C variables.

Element-wise access to array-of-structs C variables can be specified in the model via TargetLink Data Store Read and Data Store Write blocks.

Related documentation

- [Working with Array-of-Bus Signals](#) ([TargetLink Preparation and Simulation Guide](#))

Data Stores Usable Across Code Generation Units

Using Data Stores across CGUs

TargetLink can now consider data stores during code generation even though they are specified outside the subsystem configured for incremental code generation or referenced model for which code is being generated.

This is possible by specifying global data stores in the Data Dictionary via DD Block objects whose BlockType property is set to `TL_DataStoreMemory`. TargetLink Data Store Memory blocks can reference these objects and `Simulink.Signal` objects can be associated with these objects.

Related documentation [Working with Array-of-Bus Signals \(TargetLink Preparation and Simulation Guide\)](#)

N-Dimensional Look-Up Tables

Using n-dimensional Look-Up tables for DD-based code generation

Via Interpolation DD Look-Up table objects you can centrally specify look-up tables with up to five dimensions in the Data Dictionary. You can use these objects for DD-based code generation.

Using them in DD-based code generation lets you generate Look-Up table arrays with n dimensions in code modules that are independent of model elements.

You can use these generated arrays in combination with external or legacy implementations of n-dimensional table functions for three or more dimensions.

Related documentation

- [How to Specify an N-Dimensional DD Look-Up Table Object for Code Generation \(TargetLink Preparation and Simulation Guide\)](#)

Other MATLAB/Simulink/Stateflow Features

Bitwise not for numerical values

For MATLAB code and Stateflow® action language, TargetLink 5.0 additionally supports bitwise not-operations applied directly to numerical values, if the operation is the last operation on the right-hand side of an assignment. Additionally, all of the following conditions for the left-hand side of the assignment must be fulfilled:

- The left-hand side is not scaled.
- The left-hand side is not saturated.
- The data type of the left-hand side is not a floating-point type.

Related documentation

- None

Stateflow chart blocks available in library

With TargetLink 5.0, Stateflow® Chart blocks are available in the TargetLink block library for supported Simulink blocks. The Action Language property of these blocks has been pre-set to C.

Related documentation

- [Code-Relevant Simulink Blocks \(TargetLink Model Element Reference\)](#)

SPI blocks without preceding blocks	For SPI blocks that are not connected to preceding blocks or that are connected to a Ground block, the block output inherits the data type <code>Int32</code> and the default <code>VOID_SCALING</code> scaling. TargetLink displays W15295.
--	--

Related documentation

- [W15295](#)

MATLAB Code

MATLAB Code Improvements

MATLAB code struct support	TargetLink 5.0 supports the <code>Simulink.Bus</code> data type for MATLAB variables except local MATLAB variables. As with buses in Stateflow, these variables are represented by structured types in the production code.
-----------------------------------	---

Related documentation

- [Basics on Code Generation for MATLAB® Code \(TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models\)](#)
- [Basics on Working with MATLAB Functions \(TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models\)](#)

Comments in MATLAB code	TargetLink 5.0 lets you transfer code comments from your MATLAB code to the generated production code. This improves the readability and traceability of the generated production code.
--------------------------------	---

Related documentation

- [Example of Transferring MATLAB Function Comments to Production Code \(TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models\)](#)

Automatic use of Simulink data types in MATLAB code functions	For any local MATLAB variables you did not specify explicitly as Data Dictionary objects, TargetLink 5.0 provides new Code Generator options that let you use the specified Simulink® data type as the basis for code generation. This increases the efficiency of production code that was generated with default settings.
--	--

Related documentation

- [Basics on Mapping of Local MATLAB® Variable Data Types \(TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models\)](#)
- [New Code Generator Options on page 53](#)

Adaptive AUTOSAR

Adaptive AUTOSAR

Support of ara::com

AUTOSAR Adaptive Platform, also called Adaptive AUTOSAR, is a standard based on a service-oriented architecture that aims at on-demand software updates and high-end functionalities.

TargetLink 5.0 supports select features of AUTOSAR Adaptive Platform Release 19-03 . The modeling constructs and Data Dictionary workflow for Adaptive AUTOSAR closely resemble the ones used for Classic AUTOSAR.

TargetLink 5.0 supports the Adaptive AUTOSAR standard as follows:

- Import and export of Adaptive AUTOSAR ARXMLs
- Modeling of selected parts of a service-based communication as described by ara::com:
 - Behavior of methods on server side via subsystems and the TargetLink Function block
 - Method calls on client side via subsystems and the TargetLink Function block
 - Access to fields of service interfaces
 - Configurable service discovery
 - Generation of C/C++ code to use in adaptive applications

Related documentation

- *Adaptive AUTOSAR-compliant models with TargetLink:*

http://www.dspace.com/go/tl_aar

Classic AUTOSAR

Where to go from here

Information in this section

Supported AUTOSAR Releases.....	46
Improved Support of Data Store Blocks for AUTOSAR Communication.....	46
Improved Import and Export for AUTOSAR 4.4.....	46

Supported AUTOSAR Releases

Supported AUTOSAR Releases

The following AUTOSAR Releases are supported:

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

AUTOSAR 2.x/3.x no longer supported TargetLink no longer supports code generation for AUTOSAR 2.x/3.x. For more information, refer to [AUTOSAR 2.x/3.x no Longer Supported](#) on page 307.

Improved Support of Data Store Blocks for AUTOSAR Communication

AUTOSAR communication with array-of-struct data types is now possible via data store blocks for the following communication kinds:

- Interrunnable communication
- Sender-receiver communication
- NvData communication

Related documentation

- [Working with Array-of-Bus Signals](#) ( [TargetLink Preparation and Simulation Guide](#))

Improved Import and Export for AUTOSAR 4.4

IsOptional element for struct components

TargetLink can now import and export the `isOptional` attribute of `APPLICATION-RECORD-ELEMENT` and `IMPLEMENTATION-DATA-TYPE-ELEMENT` elements. This attribute provides information on whether the `APPLICATION-`

RECORD-ELEMENT or **IMPLEMENTATION-DATA-TYPE-ELEMENT** is available at runtime.

In the Data Dictionary, this attribute is represented by the **IsOptional** property of DD **ApplicationDataTypeComponent** and **TypedefComponent** objects. The property can be specified if these objects specify a primitive data type or an array data type.

DisplayPresentation element	TargetLink can import and export the DisplayPresentation attribute of the SW-DATA-DEF-PROPS . This attribute controls the presentation of the related data in a measurement and calibration tool. In the Data Dictionary, this attribute is represented by the DisplayPresentation property of DD ApplicationDataTypeComponent , TypedefComponent objects. The property can be specified if these objects specify a primitive data type or an array data type.
Offset element	TargetLink can import and export the offset attribute of TIMING-EVENT elements. In the Data Dictionary, this attribute is represented by the Offset property of DD RteEvent objects. The property can be specified if these objects specify a timing event.

Tool Chain Support

Using Several TargetLink-Generated SIC Files in the Same Application Process

You can now assign TargetLink-generated SIC files together with Simulink models, bus simulation container files and other SICs to the same application process in ConfigurationDesk.

Related documentation

- [Assigning multiple model implementations to the same application process in ConfigurationDesk \(TargetLink Interoperation and Exchange Guide\)](#)

Code Generation Core Functionality

Where to go from here

Information in this section

Support of Modern Standards of C.....	48
Void casts in function calls.....	49
Generating Code for 64-Bit Linux Targets.....	49
Improved Code Efficiency.....	49

Support of Modern Standards of C

Support of C99/C11

TargetLink 5.0 supports the common subset of C90 and C99/C11. It is now possible to generate production code that fully complies with the C99 and C11 standard. Additionally, select features of C99 are supported.

TargetLink 5.0 supports the specification of custom restrictions for identifiers. This lets you prevent or detect unintended use of reserved identifiers.

Related documentation

- [Basics on Generating Code Compliant with C99/C11/C++ \(TargetLink Customization and Optimization Guide\)](#)
- [How to Specify Custom Identifier Restrictions \(TargetLink Customization and Optimization Guide\)](#)

Common subset of C90 and C++11/C++14/C++17

TargetLink 5.0 supports the common subset of C90 and C++11/14/17.

TargetLink 5.0 supports the generation of `extern C` encapsulation via the new `EmitEncapsulationByExternC` Code Generator option. This allows for integration of C production code into C++ projects.

Related documentation

- [Basics on Generating Code Compliant with C99/C11/C++ \(TargetLink Customization and Optimization Guide\)](#)
- [EmitEncapsulationByExternC \(TargetLink Model Element Reference\)](#)

Void casts in function calls

Void casts in function calls - Generation of void casts for the suppression of return values

Via the new InsertVoidCastForUnusedFcnCallReturnValue Code Generator option, TargetLink 5.0 lets you specify whether a void cast is inserted in front of function calls if the return value of the function is not used.

Related documentation

- [InsertVoidCastForUnusedFcnCallReturnValue](#) ( [TargetLink Model Element Reference](#))

Generating Code for 64-Bit Linux Targets

TargetLink can now generate generic  [ANSI C](#) production code for 64-Bit Linux targets. In this context the base types of 32-Bit integer data types are defined as follows:

- `Int32`: `signed int` instead of `signed long int`.
- `UInt32`: `unsigned int` instead of `unsigned long int`.

The reason is that on 64-bit Linux targets the length of the long data type is 8 byte, on 32-bit Linux and Windows targets 4 bytes. Therefore, on 64-bit Linux `signed long int` and `unsigned long int` do not define 32-bit integer data types.

Improved Code Efficiency

Combining Code of If Statements

TargetLink can now combine the code of several `if` statements that depend on the same or negated condition.

Related documentation

- [Combining Code of If Statements](#) ( [TargetLink Customization and Optimization Guide](#))
- [CombineControlFlowStatements](#) ( [TargetLink Model Element Reference](#))

Merging of nonconsecutive loops

TargetLink now tries to merge nonconsecutive loops. This even works when logging via log macros is enabled.

Related documentation

- [Merging Loops](#) ( [TargetLink Customization and Optimization Guide](#))
- [CombineControlFlowStatements](#) ( [TargetLink Model Element Reference](#))

Target Simulation (PIL)

Changes in the Target Simulation Modules

Support for new evaluation board

The Target Simulation Module of TargetLink 5.0 supports the Texas Instruments LAUNCHXL2570LC43 evaluation board.

Related documentation

- [Combinations of Evaluation Boards and Compilers \(Evaluation Board Reference\)](#)

Availability of support for Freescale MPC5604B

As of TargetLink 5.0, the Freescale MPC5604B evaluation board is supported via a Target Simulation Module (TSM) Extension instead of the TSM. TSM Extensions are free of charge if you have a valid Software Maintenance Service (SMS) contract.

Related documentation [Combinations of Evaluation Boards and Compilers \(Evaluation Board Reference\)](#)

New and discontinued compiler versions

The following table shows the compiler versions that are now supported by TargetLink 5.0. Refer to the New and No changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.

Microcontroller Family	Compiler	New	No Changes	Discontinued
ARM CortexM3	Keil	—	5.2	—
C16x	TASKING	—	8.6	—
MPC57xxVLE	Diab	—	5.9	—
	GreenHill	2019	—	2018
MPC560xVLE	Diab	—	—	5.9
	GreenHill	—	—	2018
RH850	GreenHill	2019	—	2018
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3	—
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	—	3.2	6.2
TriCore2xx	TASKING	6.3	—	6.2
	GCC	4.9	—	4.6
TMS570 (ARM)	CCS	7.0	—	—

Microcontroller Family	Compiler	New	No Changes	Discontinued
V850	GreenHill	2019	—	2018
XC22xx	TASKING	—	3.0	—

For more information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ([Evaluation Board Reference](#)).

Note

For more PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Usability

Where to go from here

Information in this section

Data Dictionary and Data Dictionary Manager.....	51
Property Manager.....	53

Data Dictionary and Data Dictionary Manager

Data Dictionary Manager Dialogs

TargetLink 5.0 provides the new Advanced Edit String List dialog in the Data Dictionary Manager.

In addition, the following Data Dictionary Manager dialogs are optimized in TargetLink 5.0:

- Edit String List
- Plain Variable
- Struct or Implicit Struct Variable

Related documentation

- Plain Variable ([TargetLink Data Dictionary Manager Reference](#))
- Advanced Edit String List ([TargetLink Data Dictionary Manager Reference](#))
- Edit StringList ([TargetLink Data Dictionary Manager Reference](#))
- Struct or Implicit Struct Variable ([TargetLink Data Dictionary Manager Reference](#))

Import and Export Dialogs

TargetLink 5.0 provides improved DD import and export dialogs that stay open for repeated import and export. This lets you keep settings you already specified in the dialog.

Related documentation

- [Import from AUTOSAR File](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Import from XML File](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Import from OIL File](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Export as AUTOSAR File](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Export as A2L File](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Export as XML File](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Export as OIL File](#) ([TargetLink Data Dictionary Manager Reference](#))

Usability improvements

TargetLink 5.0 provides several usability improvements in the Data Dictionary Manager, for example:

- Improved menu extensions
- A new details pane for objects and properties
- A symbol for custom properties
- An improved saving of DDs, write-protected DDs and DD include files

Related documentation

- [Details](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Save As](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Save](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Save All](#) ([TargetLink Data Dictionary Manager Reference](#))
- [Add Custom Property](#) ([TargetLink Data Dictionary Manager Reference](#))
- [How to Include Partial Data Dictionary Files](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))
- [Basics on Adding Custom Functionality to the Data Dictionary Manager](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

Data Model Revision

TargetLink 5.0 provides data model revision numbers for included DD files and for XML import and export.

Related topics

- [Basics on Importing XML Files](#) ([TargetLink Interoperation and Exchange Guide](#))
- [Basics of Exporting XML Files](#) ([TargetLink Interoperation and Exchange Guide](#))

Adjust to Typedef for multiple DD objects at once

For a more efficient operation, the Adjust to Typedef functionality can now be used on multiple DD objects at once as follows:

- On DD VariableGroup objects to adjust all child objects.
- On multiple selected DD Variable objects and DD InterRunnableVariables objects to adjust all selected objects.

Related documentation

- None.

Property Manager

Validation Summary

TargetLink 5.0 provides a Validation Summary in the Property Manager. It displays model element data validation errors from all model element variables of the Property View. It lets you search, filter, and group errors.

Related documentation

- [Display of Validation Errors](#) ([TargetLink Preparation and Simulation Guide](#))
- [Validation Summary](#) ([TargetLink Tool and Utility Reference](#))
- [How to Search for Model Elements and Validation Errors](#) ([TargetLink Preparation and Simulation Guide](#))
- [How to Filter for Property Values and Validation Errors](#) ([TargetLink Preparation and Simulation Guide](#))
- [How to Customize Columns in the Property View and Validation Summary](#) ([TargetLink Preparation and Simulation Guide](#))

Code Generator Options

New Code Generator Options

Overview of new Code Generator options

The following new Code Generator options are available with TargetLink 5.0:

- **CombineControlFlowStatements**
Combine two or more control flow statements with compatible controlling expression into one control flow statement. Refer to [Improved Code Efficiency](#) on page 49.
- **EmitEncapsulationByExternC**
Emit `extern C` encapsulation via `#ifdef __cplusplus`.

- **UtilizeBitwiseShiftOperations**
Use bitwise shift operations to implement efficient multiplications and divisions in a portable manner.
- **EmitAllTagNameConflictsAsError**
Emit an error if a tag name is simultaneously used as another identifier.
- **InsertVoidCastForUnusedFcnCallReturnValue**
Enable the insertion of void casts in front of function calls if the return value is not used.
- The following Code Generator options can be used to map Simulink® data types to TargetLink data types for local MATLAB® variables whose data type is not explicitly specified:
 - **AutoSelectBoolTypesForLocalMATLABVariables**
Map the Simulink® data type `logical` to the TargetLink data type `Bool`.
 - **AutoSelectIntegerTypesForLocalMATLABVariables**
Map the Simulink® data types `int*` and `uint*` to the equivalent TargetLink data type, e.g., `int16 <-> Int16`, `uint8 <-> UInt8`.
 - **AutoSelectFloatTypesForLocalMATLABVariables**
Map the Simulink® data types `single` and `double` to the equivalent TargetLink data types, e.g., `single <-> Float32`, `double <-> Float64`.

Refer to [Basics on Mapping of Local MATLAB® Variable Data Types](#) ([TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models](#)).

Related documentation

- [CombineControlFlowStatements](#) ([TargetLink Model Element Reference](#))
- [EmitEncapsulationByExternC](#) ([TargetLink Model Element Reference](#))
- [UtilizeBitwiseShiftOperations](#) ([TargetLink Model Element Reference](#))
- [EmitAllTagNameConflictsAsError](#) ([TargetLink Model Element Reference](#))
- [InsertVoidCastForUnusedFcnCallReturnValue](#) ([TargetLink Model Element Reference](#))
- [AutoSelectBoolTypesForLocalMATLABVariables](#) ([TargetLink Model Element Reference](#))
- [AutoSelectIntegerTypesForLocalMATLABVariables](#) ([TargetLink Model Element Reference](#))
- [AutoSelectFloatTypesForLocalMATLABVariables](#) ([TargetLink Model Element Reference](#))

For reference information on all Code Generator options, refer to [Alphabetical List of Code Generator Options](#) ([TargetLink Model Element Reference](#)).

Migration aspects of Code Generator options

For more information, refer to [Migration Aspects Regarding Code Generator Options](#) on page 304.

API Functions and Hook Scripts

Where to go from here

Information in this section

New API Functions.....	55
Improved API Functions and Hook Scripts.....	55

New API Functions

List of new API functions

API Function	Purpose
<code>tlFindHook</code> (TargetLink API Reference)	Searches for TargetLink hook scripts whose names match the specified pattern.
<code>tlRebuildFixedPointLibrary</code> (TargetLink API Reference)	Rebuilds the Fixed-Point Library.
<code>dsdd('New'[,<template>])</code>	Creates a new DD workspace using a pre-defined or user template.

Improved API Functions and Hook Scripts

Improved API functions and hook scripts

With TargetLink 5.0, the following API functions and hook scripts have been improved.

`tl_build_standalone` and `tl_tl2rti`

With TargetLink 5.0, you can create a stand-alone TargetLink S-function without starting code generation via the new property `SkipCodeGeneration` of the `tl_build_standalone` and `tl_tl2rti` API function. By default, the new property is set to `off`.

To use this new property, you have to ensure that the required production code is available. The production code's description, i.e., the DD Subsystem object and the DD Application object and their child objects, must exist in the open Data Dictionary.

Related documentation

- [tl_build_standalone](#) ([TargetLink API Reference](#))
- [tl_tl2rti](#) ([TargetLink API Reference](#))

tlSimulinkBusObject With TargetLink 5.0, the `tlSimulinkBusObject` API function can create a DD Typedef object from a Simulink.Bus object.

Related documentation

- [tlSimulinkBusObject](#) ([TargetLink API Reference](#))
- [Basics on Preparing Simulink Systems for TargetLink Code Generation](#) ([TargetLink Preparation and Simulation Guide](#))
- [Basics on Modeling Buses via Data Store Blocks](#) ([TargetLink Preparation and Simulation Guide](#))

Other

Where to go from here

Information in this section

General Enhancements and Changes.....	56
TargetLink Demos.....	57

General Enhancements and Changes

Navigating TargetLink menus without using a mouse

In the Simulink Model Editor, you can now navigate in TargetLink menus via keyboard without using a mouse. First, press the **Alt** key to display the current menu with a unique letter underlined in each menu item. Then enter the related letter to select a specific menu item.

You can abort keyboard menu navigation by pressing **Alt** again.

Extracting model referencing subsystem via block dialog

With TargetLink 5.0, you can extract model referencing subsystems via the Extract Subsystem button on the Incremental page of the Function Block block. This was already possible via the `tlExtractSubsystem` API command.

Related documentation

- [Function Block](#) ([TargetLink Model Element Reference](#))
- [Basics on Model Referencing](#) ([TargetLink Customization and Optimization Guide](#))
- [How to Extract Subsystems to a Test Frame in a Separate Model](#) ([TargetLink Customization and Optimization Guide](#))

Messages for conflicting specifications

The message E17299 is displayed if an identifier is not unique and does not meet the requirements for variable merging.

However, in TargetLink < 5.0 this message is displayed only once per code generation run, even if the model contains several identifiers where merging is not possible.

With TargetLink 5.0, this message is displayed multiple times if required. This reduces the number of code generation attempts, especially when merging variables.

TargetLink Demos

New demo models

The following new demo models are available for TargetLink 5.0:

- [AR_INCREMENTAL_CODEGEN](#) ([TargetLink Demo Models](#))
- [VECU_EVENT_SIMULATION](#) ([TargetLink Demo Models](#))

New Features of TargetLink 4.4

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

Tool Chain Support	58
Modeling in Simulink or Stateflow	58
Code Generation Core Functionality	62
AUTOSAR	63
Target Simulation (PIL)	65
Data Dictionary and Data Management	67
Code Generator Options	68
API Functions and Hook Scripts	69
Other	70

Tool Chain Support

Executing TargetLink Code on dSPACE Real-Time Hardware

TargetLink can now generate SIC files to interoperate with ConfigurationDesk. This lets you execute the production code generated by TargetLink on dSPACE hardware for real-time simulations.

Offline simulation on VEOS is also possible.

Related documentation

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

Modeling in Simulink or Stateflow

Where to go from here

Information in this section

Support of Arrays of Struct.....	58
Bus Support of the Constant Block.....	59
Other MATLAB/Simulink/Stateflow Features.....	60
TargetLink Module for MATLAB Code.....	61

Support of Arrays of Struct

DD-based code generation

TargetLink supports arrays of struct in the generated code as follows:

- Specifying array-of-struct variables in the Data Dictionary
- Specifying initial values in the Data Dictionary
- Generating C modules with global array-of-struct interface variables from the Data Dictionary
- Accessing array elements and leaf struct components in the model via access functions or custom code
- Address-based logging of leaf struct components during SIL/PIL simulation

Related documentation

- [Working with Array-of-Bus Signals](#) ([TargetLink Preparation and Simulation Guide](#))

Bus Support of the Constant Block

Bus-capable Constant Block

TargetLink now supports bus signals in Constant blocks.

You can easily generate a complete Simulink bus by creating a Simulink.Bus object and specifying it as the data type of a Constant block. During model initialization, the Constant block outputs a Simulink bus.

You can also create predefined structs in the TargetLink Data Dictionary, and assign these DD Typedef objects or DD Variable objects at Constant blocks. The mapping is done in the same way as for all bus-capable blocks. For more information, refer to [Mapping Entire Buses to Explicit Structs in the Code](#) ([TargetLink Customization and Optimization Guide](#)).

To specify an initial condition, you can set the Constant value on the Constant page of the Constant block to one of the following options:

- To use the default value for every signal, enter `0` (or a MATLAB expression that evaluates to 0).
For numerical data types, the default value is always '0'.
For enumerations, the default value (if specified) or the first value is used.
- To specify a specific initial condition for each bus element, specify a complete initial condition structure. Partial IC structures are not supported at Constant blocks. For more information, refer to [Basics on Initializing Buses via Initial Condition Structures](#) ([TargetLink Preparation and Simulation Guide](#)).

Obsolete limitation A limitation concerning Constant blocks has become obsolete. Refer to [Obsolete Limitations](#) on page 407.

Constant blocks whose data type is specified by using a Simulink.Bus object are now supported by TargetLink.

Related documentation

- [Basics on Defining Buses via Constant Blocks](#) ([TargetLink Preparation and Simulation Guide](#))
- [Constant Block](#) ([TargetLink Model Element Reference](#))
- [Basics on the Representation of Buses in the Production Code](#) ([TargetLink Preparation and Simulation Guide](#))
- [Basics on Initializing Buses via Initial Condition Structures](#) ([TargetLink Preparation and Simulation Guide](#))
- [tlSimulinkBusObject](#) ([TargetLink API Reference](#))

Other MATLAB/Simulink/Stateflow Features

Fast Restart mode for iterative model simulations

In Fast Restart mode, you can simulate a model without initializing it again. For more information on Fast Restart, refer to the Simulink documentation.

TargetLink now supports Fast Restart for all three simulation modes (MIL, SIL and PIL), including signal logging, Min/Max logging, plotting, and overflow detection.

The Simulink limitations regarding the modification of models between two simulation runs also apply to TargetLink. However, in MIL simulation mode, you can change TargetLink data, such as the scaling or the data type, before you start a new simulation.

To enable or disable the Fast Restart mode, click the Fast Restart button on the Simulink Editor toolbar or execute the following API commands:

- `set_param(<model>, 'FastRestart', 'on')`
- `set_param(<model>, 'FastRestart', 'off')`

Obsolete limitation A limitation concerning the fast restart behavior has become obsolete. Refer to [Obsolete Limitations](#) on page 407.

Simulink no longer automatically disables the Fast Restart mode when starting a simulation if at least one TargetLink subsystem is in SIL or PIL mode.

Related documentation

- None

String array support

In addition to character arrays (short sequence of characters (text) enclosed in single quotes), the TargetLink API now also supports text processing for string arrays in MATLAB®. A string array is a container used for larger text sequences (enclosed in double quotes). For details on working with text as data, refer to the MathWorks documentation: [Represent Text with Character and String Arrays](#).

TargetLink 4.4

```
tl_get(<block>, "output.type")
OR
tl_set(<block>, 'output.description', "My description")
```

Related documentation

- None

Initialization of function return values for iterated subsystems

You can now specify the block variables of TargetLink OutPort blocks of iterated subsystems as function return values that are initialized variables as well. To achieve this, the variable must have a variable class with the Scope property of the DD VariableClass object set to `fcn_return` and the InitAtDefinition property to `on`.

TargetLink 4.4 generates initialized variables that are returned by the function. In TargetLink ≤ 4.3, the InitAtDefinition property was ignored and uninitialized

variables were returned. Furthermore, TargetLink displayed message E06514 informing about an invalid DD VariableClass object.

TargetLink ≤ 4.3	TargetLink 4.4
<pre>Int16 Sa2_IterExtern(Int16 Sa2_In1) { Int16 Sa2_Out1; Int32 Sa2_For_Iterator_it; for (Sa2_For_Iterator_it = 1; Sa2_For_Iterator_it <= ((Int32) Sa2_In1); Sa2_For_Iterator_it++) { Sa2_Out1 = [...] } return Sa2_Out1; }</pre>	<pre>Int16 Sa2_IterExtern(Int16 Sa2_In1) { Int16 Sa2_Out1 = 0; Int32 Sa2_For_Iterator_it; for (Sa2_For_Iterator_it = 1; Sa2_For_Iterator_it <= ((Int32) Sa2_In1); Sa2_For_Iterator_it++) { Sa2_Out1 = [...] } return Sa2_Out1; }</pre>

Related documentation

- None

TargetLink Module for MATLAB Code

MATLAB code

MATLAB® code provides functions for a script-based implementation of function algorithms, which is useful for coding algorithms on a textual basis (procedural language) instead of a graphical language. However, not all MATLAB functions are supported by TargetLink. All functions that are fully or partially supported can be used for implementing the function algorithm.

The TargetLink Module for MATLAB Code provides the following features:

- Generate code from MATLAB Function blocks in Simulink® models to extend model-based development with procedural language functions and operators
- Generate code from Stateflow® MATLAB Functions in Stateflow to easily access MATLAB expressions from Stateflow charts
- Full specification of fixed-point and other code properties to generate optimized fixed-point code that complies with the applicable coding guidelines
- AUTOSAR support to create standard-compliant AUTOSAR Software Components from models containing MATLAB code components
- Tight integration with TargetLink features, so you do not have to modify existing workflows
- Call legacy functions from MATLAB code to reuse C code that has already been verified and tested

All features are natively supported by TargetLink, so no separate TargetLink block is required.

Related documentation

- [Basics on Code Generation for MATLAB® Code \(TargetLink Code Generation Guide for MATLAB® Code in Simulink® Models\)](#)

- Supported MATLAB® Code Function Statements and Function Operators
( [TargetLink Code Generation Reference for MATLAB® Code in Simulink® Models](#))

Code Generation Core Functionality

New Value Copy Access Functions

TargetLink now lets you specify value copy access functions as either system inputs or system outputs.

Example

```
Int16 Aux_MyInput[56];
Int16 Aux_MyOutput[20];
GetMyInput(Aux_MyInput);
/* copies contents of MyInput to Aux_MyInput */
...
SetMyOutput(Aux_MyOutput);
/* copies contents of Aux_MyOutput to MyOutput */
```

Related documentation

- [Encapsulating Access to Data Signals and Parameters via Access Functions](#)
( [TargetLink Customization and Optimization Guide](#))
- [AccessFunction](#)
- [Overview of Predefined Access Function Templates](#)

AUTOSAR

Where to go from here

Information in this section

Supported AUTOSAR Releases.....	63
Support of Provide-Require Ports for Sender-Receiver Communication.....	64
Support of Data Store Blocks for Sender-Receiver Communication.....	64
Support of Explicit NvData Communication.....	64
Support of Rte_IWrite and Rte_IWriteRef in Sender-Receiver and NvData Communication.....	64
Exporting Splittable Elements.....	65

Supported AUTOSAR Releases

Supported AUTOSAR Releases

The following AUTOSAR Releases are supported:

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

Support of Provide-Require Ports for Sender-Receiver Communication

You can now use provide-require ports when modeling sender-receiver communication.

Related documentation

- [Modeling Sender-Receiver Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Support of Data Store Blocks for Sender-Receiver Communication

You can now use data store blocks to model sender-receiver communication.

Related documentation

- [Modeling Sender-Receiver Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Support of Explicit NvData Communication

You can now model explicit NvData communication.

Related documentation

- [Modeling NvData Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Support of Rte_IWrite and Rte_IWriteRef in Sender-Receiver and NvData Communication

Support of Rte_IWrite and Rte_IWriteRef

TargetLink now supports the `Rte_IWrite` and `Rte_IWriteRef` RTE API functions for both SenderReceiver and NvData communication. You can select the API to generate in the related block dialog.

Related documentation

- [Modeling NvData Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [Modeling Sender-Receiver Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Exporting Splittable Elements

Exporting Splittable Elements

TargetLink now lets you export the following elements to separate AUTOSAR (ARXML) files:

- <AR-TYPED-PER-INSTANCE-MEMORYS>
- <CONSTANT-MEMORYS>
- <EXPLICIT-INTER-RUNNABLE-VARIABLES>
- <SWC-INTERNAL-BEHAVIOR>
- <IMPLICIT-INTER-RUNNABLE-VARIABLES>
- <PER-INSTANCE-PARAMETERS>
- <SHARED-PARAMETERS>
- <STATIC-MEMORYS>

Related documentation

- [Introduction to Importing and Exporting AUTOSAR Files \(TargetLink Interoperation and Exchange Guide\)](#)
- [AR_POSCONTROL \(TargetLink Demo Models\)](#)

Target Simulation (PIL)

Where to go from here

Information in this section

Support for Virtual Evaluation Boards.....	65
Changes in the Target Simulation Modules.....	66

Support for Virtual Evaluation Boards

Virtual evaluation boards

TargetLink lets you run a processor-in-the-loop simulation on virtual evaluation boards (instruction set simulators (ISS)), such as the TRACE32 Simulator by Lauterbach.

Related documentation

- [Processor-in-the-Loop Simulation Mode \(TargetLink Preparation and Simulation Guide\)](#)
- [Virtual Evaluation Boards \(Evaluation Board Reference\)](#)

Changes in the Target Simulation Modules

New and discontinued compiler versions

The following table shows the compiler versions that are now supported by TargetLink 4.4. Refer to the New and No changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.

Microcontroller Family	Compiler	New	No Changes	Discontinued
ARM CortexM3	Keil	—	5.2	—
C16x	TASKING	—	8.6	—
MPC57xxVLE	Diab	—	5.9	—
	GreenHill	2018	—	2016
MPC560xVLE	Diab	—	5.9	—
	GreenHill	2018	—	2016
RH850	GreenHill	2018	—	2016
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3	—
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	—	3.2, 6.2	—
TriCore2xx	TASKING	—	6.2	—
	GCC	4.9	—	4.6
V850	GreenHill	2018	—	2016
XC22xx	TASKING	—	3.0	—

For more information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ([Evaluation Board Reference](#)).

Note

For more PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Data Dictionary and Data Management

Where to go from here

Information in this section

Support of Show Masked Content and Show Library Content in the Property Manager.....	67
New Import and Export of View Sets in the Property Manager.....	67
Support for User Mode per Workspace in the Data Dictionary Manager.....	68

Support of Show Masked Content and Show Library Content in the Property Manager

Show masked content and Show library content in the Property Manager

The TargetLink Property Manager now supports showing or hiding the content of the following model elements in the Model Navigator and the properties of the underlying model elements in the Property View:

- The content of masked subsystems and charts.
- The content of model elements with active library links, including LinkCharts.

Related documentation

- [Show Masked Content](#) ([TargetLink Tool and Utility Reference](#))
- [Show Library Content](#) ([TargetLink Tool and Utility Reference](#))
- [Basics on Working with Libraries](#) ([TargetLink Preparation and Simulation Guide](#))

New Import and Export of View Sets in the Property Manager

Import and export of view sets

The TargetLink Property Manager now supports the import and export of custom property view sets via View Set XML (VSML) files.

Related documentation

- [Import View Sets Dialog](#) ([TargetLink Tool and Utility Reference](#))
- [Export View Sets Dialog](#) ([TargetLink Tool and Utility Reference](#))
- [How to Create Property View Sets](#) ([TargetLink Preparation and Simulation Guide](#))

Support for User Mode per Workspace in the Data Dictionary Manager

One user mode and password per workspace TargetLink lets you assign a user mode and password per workspace in the Data Dictionary.

Related documentation

- [Users](#)
- [GetUser](#)
- [SetUser](#)
- [SetPassword](#)

Code Generator Options

New Code Generator Options

Overview of new Code Generator options

The following new Code Generator options are available with TargetLink 4.4.

- [InitializeVariablesViaRestartFunction](#)
Lets you globally change restart initialization behavior.
- [RealTimePlatformSupport](#)
Causes the Code Generator to consider specifications for use with dSPACE ConfigurationDesk and VEOS Player.

Related documentation

- [InitializeVariablesViaRestartFunction](#) ( [TargetLink Model Element Reference](#))
- [RealTimePlatformSupport](#) ( [TargetLink Model Element Reference](#))

For reference information on all Code Generator options, refer to [Alphabetical List of Code Generator Options](#) ( [TargetLink Model Element Reference](#)).

Migration aspects of Code Generator options

For more information, refer to [Migration Aspects Regarding Code Generator Options](#) on page 314.

API Functions and Hook Scripts

Where to go from here

Information in this section

New API Functions.....	69
Improved Hook Scripts.....	69

New API Functions

List of new API functions

API Function	Purpose
<code>tlGenerateSic</code>	Generates a dSPACE Simulink implementation container using the TargetLink Code Generator.

Related documentation:

- [tlGenerateSic](#) ([TargetLink API Reference](#))
- [Interoperating with ConfigurationDesk](#) ([TargetLink Interoperation and Exchange Guide](#))

With TargetLink 4.4, API functions have changed. Refer to [Changes in API Functions](#) on page 316.

Improved Hook Scripts

`tl_pre_compile_host`

TargetLink supports parallel compiling for SIL simulation. You can disable or enable the parallel compilation by means of the `EnableParallelCompiling` option of the `tl_pre_compile_host` hook script.

Related documentation

- [tl_pre_compile_host_hook](#) ([TargetLink File Reference](#))

Other

Where to go from here

Information in this section

General Enhancements and Changes.....	70
TargetLink Demos.....	70
Synchronizing Simulink and TargetLink.....	71

General Enhancements and Changes

Blocks not supporting matrix signals

In TargetLink 4.4, blocks that do not support matrix signals reduce a [1 x 1] signal to a scalar. In this case, code generation is now possible.

Related documentation

- [Block-Specific Limitations](#) ([TargetLink Limitation Reference](#))

Simulink parameter-based differences between MIL and SIL simulations

If you enable the Use algorithms optimized for row-major array layout Simulink configuration parameter to enable efficient algorithms, this might result in numerical differences between simulation and generated code. However, TargetLink does not support the Use algorithms optimized for row-major array layout Simulink configuration parameter.

Block and Object Reference renamed and reorganized

The *Block and Object Reference* user documentation was renamed to *Model Element Reference*. Additionally, it now focuses on the TargetLink properties of TargetLink blocks and Stateflow objects. TargetLink blocks and Stateflow objects are model elements that have TargetLink-specific properties. These properties influence how TargetLink generates production code from the model.

TargetLink Demos

New demo models

The following new demo models are available for TargetLink 4.4:

MATLAB_CODE_STATEFLOW The MATLAB_CODE_STATEFLOW demo model demonstrates fixed-point code generation of MATLAB functions.

Refer to [MATLAB_CODE_STATEFLOW](#) ([TargetLink Demo Models](#)).

MATRIX_INVERSE The **MATRIX_INVERSE** demo model shows two possibilities for matrix inversion:

- A block with only one output, which simply calculates and returns a matrix - without any error check.
- A block with an additional second output that returns a status indicating whether it was possible to calculate the inverse matrix.

Refer to [MATRIX_INVERSE](#) ([TargetLink Demo Models](#)).

REAL_TIME_PLATFORM_SUPPORT The **REAL_TIME_PLATFORM_SUPPORT** demo model shows how to prepare a TargetLink subsystem so you can use it as a behavior model (i.e., a SIC file) in ConfigurationDesk. In particular, it shows how to model the subsystem's interface via model port variables, how to model multiple SIC runnable functions that can be connected to different tasks (asynchronous or time-triggered) in ConfigurationDesk, and how to ensure a secure communication between the tasks in a real-time simulation.

Refer to [REAL_TIME_PLATFORM_SUPPORT](#) ([TargetLink Demo Models](#)).

Extended demo models

The following demo models have been extended for TargetLink 4.4:

AR_POSCONTROL New feature: Settings for AUTOSAR export, i.e., files are exported as splittables.

Refer to [AR_POSCONTROL](#) ([TargetLink Demo Models](#)).

BUS_CC New feature: The specification of array-of-struct variables as DD-based objects is explained. You can access field elements of array-of-struct variables by means of access functions for a model-based code generation unit.

Refer to [BUS_CC](#) ([TargetLink Demo Models](#)).

Synchronizing Simulink and TargetLink

Preparing, synchronizing, and clearing Simulink models

TargetLink 4.4 lets you synchronize referenced Simulink library blocks with TargetLink data from a specific library block instances.

Related documentation

- [System Preparation Tool](#) ([TargetLink Tool and Utility Reference](#))
- [How to Synchronize Simulink and TargetLink Data](#) ([TargetLink Preparation and Simulation Guide](#))

New Features of TargetLink 4.3

New licensing for dSPACE products	With dSPACE Release 2017-B, dSPACE introduces a new licensing technology for protecting dSPACE software. dSPACE has also improved the software installation process. As a consequence, some parts of the dSPACE software, for example, the user documentation, are installed in encrypted archives on the host PC. These license-protected archives must be decrypted before you can use the contained files, for example, to view the user documentation.
--	--

Where to go from here	Information in this section
	Modeling in Simulink or Stateflow..... 72
	Code Generation Core Functionality..... 74
	Modular Development..... 77
	AUTOSAR..... 78
	Target Simulation (PIL)..... 81
	Data Dictionary and Data Management..... 82
	Code Generator Options..... 82
	API Functions and Hook Scripts..... 83
	Other..... 83

Modeling in Simulink or Stateflow

Where to go from here	Information in this section
	Newly Supported Simulink Block..... 73
	Bus-Capable Custom Code Block (Type II)..... 73
	Other Simulink/Stateflow Features..... 73

Newly Supported Simulink Block

Support of the Simulink Delay block

TargetLink now supports the Simulink Delay block, which can be used to delay a signal by multiple fixed or variable sample steps.

Related documentation [Delay Block \(TargetLink Model Element Reference\)](#)

Bus-Capable Custom Code Block (Type II)

Bus-Capable Custom Code Block (Type II)

TargetLink now supports the use of Simulink buses and TargetLink structured data types with the Custom Code (type II) block.

Related documentation

- [Basics on Using Simulink Buses and TargetLink Structured Data Types with the Custom Code \(Type II\) Block \(TargetLink Preparation and Simulation Guide\)](#)
- [Overview of Methods for Preparing Unsupported Simulink Blocks for TargetLink Code Generation \(TargetLink Preparation and Simulation Guide\)](#)
- [Custom Code Block \(TargetLink Model Element Reference\)](#)
- [BUS_CC \(TargetLink Demo Models\)](#)

Other Simulink/Stateflow Features

Support for resettable subsystems

TargetLink now supports resettable subsystems. You can either use a Simulink Resettable Subsystem or copy&paste the Reset Port block in a subsystem. The subsystem then becomes resettable. The reset trigger signal resets all state variables in the blocks of the subsystem.

Related documentation

- [Basics on Resettable Subsystems \(TargetLink Customization and Optimization Guide\)](#)
- [Code-Relevant Simulink Blocks \(TargetLink Model Element Reference\)](#)

Full support for States when enabling behavior of function-call-triggered subsystems

TargetLink now fully supports the `held` and `reset` settings of the `States when enabling` property for the Trigger block when the function-call trigger type is selected. It lets you decide the way states are dealt with in function-call-triggered subsystems.

Until now, only the `inherit` setting was supported.

Support of NXOR at Logical Operator block TargetLink now supports the NXOR operation at the Logical Operator block.

Support of Min/Max at bit operation blocks The Code Generator now supports Min/Max constraints at the input and output signals of the following blocks:

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

Improved TL_Blackbox block mask type TargetLink now supports masked subsystems with the TL_BlackBox mask type to be placed anywhere in a model. TL_BlackBox subsystems no longer have to reside in subsystems specified as external functions. In addition, the new `t1IsCodeGenerationInProgress` API function lets you model TL_BlackBox subsystems.

Related documentation

- Overview of Methods for Preparing Unsupported Simulink Blocks for TargetLink Code Generation ( [TargetLink Preparation and Simulation Guide](#))

Code Generation Core Functionality

Where to go from here

Information in this section

Code Decorations (Declaration Statements and Section Names)..... 74

MISRA C Compliance..... 75

Improved Support of Variable Vector Widths..... 76

Code Decorations (Declaration Statements and Section Names)

Code Decorations

TargetLink's mechanism of generating declaration statements and section names (i.e., code decorations) was improved.

You can now specify these decorations via dedicated objects in the Data Dictionary. This lets you filter the code elements to decorate by their characteristics (data type, width, initialization, scope, and storage).

Via the `AvoidStaticLocalScope` Code Generator option you can avoid variables of local scope with static storage duration.

Related documentation

- [Migrating Data Dictionaries to CodeDecorationSets on page 291](#)
- [Decorating Production Code \(TargetLink Customization and Optimization Guide\)](#)
- [CodeDecoration](#)
- [AvoidStaticLocalScope \(TargetLink Model Element Reference\)](#)

MISRA C Compliance

Rule compliance

It is now possible to generate code that complies with all rules classified as **mandatory** or **required** according to autocoding classification.

Related documentation

- [TargetLink's MISRA C:2012 Compliance Documentation document](#)

Enable/disable pointer arithmetic in look-up table functions

TargetLink lets you enable/disable pointer arithmetic when generating code from Prelookup and Interpolation Using Prelookup blocks. If you set the `ImplementLookUpTableFunctionsWithPointerArithmetics` Code Generator option to `off`, pointer arithmetic is deactivated according to MISRA C:2012 rules. Instead, index accesses are used.

Keep in mind that this is not possible for 1D and 2D Look-Up Table blocks, because they can also be mapped with the Prelookup and Interpolation using Prelookup blocks.

Related documentation

- [Prelookup Block \(TargetLink Model Element Reference\)](#)
- [Interpolation Using Prelookup Block \(TargetLink Model Element Reference\)](#)
- [ImplementLookUpTableFunctionsWithPointerArithmetics \(TargetLink Model Element Reference\)](#)

Improvements to TargetLink's fixed-point library

Several improvements were made to TargetLink's Fixed-Point Library (DsFXP) to improve the compliance with individual MISRA C rules. The improvements include the following:

- TargetLink now supports 64-bit divisions without violating mandatory or required MISRA-C:2012 rules.
- Implementation-defined behavior messages reduced.

- The remaining `Implementation-defined behavior` messages are now documented.
- Less unreachable code (improvements to MISRA C:2012 - 2.1, 2.2, 14.3 rules).
- No more superfluous determination of loop variables (MISRA C:2012 18.1 rule).

Related documentation

- None
-

Explicit casts and floating point auxiliary variables for look-up table functions

To improve MISRA C compliance, the following changes were made to the code of look-up table functions in scenarios mixing fixed-point and floating point code:

- Implicit type conversions were made explicit via casts
-

Related topics

References

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

Improved Support of Variable Vector Widths

Supported blocks

TargetLink now supports variable vector widths for the Assignment, Custom Code, Delay, and Selector blocks. Specific settings need to be set.

Related documentation

- [Details on Variable Vector Width Implementation](#) (TargetLink Customization and Optimization Guide)
-

Width macros referenced in the model

Width macros specified as DD variable objects can now be referenced in TargetLink blocks. This allows you to make the active width value of a width-variantated DD variable explicitly available in the model.

Related documentation

- [Details on Variable Vector Width Implementation](#) (TargetLink Customization and Optimization Guide)
 - [Example of a Width Macro Referenced in the Model](#) (TargetLink Customization and Optimization Guide)
-

Report on terminated width propagation

By setting a specific code generator option, TargetLink generates a report on issues related to the propagation of variable vector widths throughout the model. The terminating blocks and ports are described.

Related documentation

- [How to Generate a Report on the Propagation Failures \(TargetLink Customization and Optimization Guide\)](#)
- [ReportVariableVectorWidthDetails \(TargetLink Model Element Reference\)](#)

Related topics**References**

[ReportVariableVectorWidthDetails \(TargetLink Model Element Reference\)](#)

Modular Development

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

Improved A2L File Generation.....	77
Improved Workflow for Distributed Development.....	77

Improved A2L File Generation

Incremental A2L File Generation

TargetLink can now incrementally generate A2L files. These files contain only the variable descriptions defined by the [Code generation unit \(CGU\)](#), which is used to generate production code.

You can also generate an A2L file for several CGUs in one step. This file then contains the descriptions of the variables that are defined in the production code of any of the CGUs.

Related documentation [Exchanging A2L Files \(TargetLink Interoperation and Exchange Guide\)](#)

Improved Workflow for Distributed Development

Improved workflow

TargetLink now supports an improved workflow for distributed development.

This lets you perform the following actions:

- Use the same workflow with incremental code generation and model referencing
- Specify the location of the artifacts generated by TargetLink as required by your project
- Integrate existing artifacts into your project

Related documentation

- [Specifying the Location of Artifacts Generated or Used by TargetLink \(TargetLink Customization and Optimization Guide\)](#)
- [Principles of Partitioning Models and Code \(TargetLink Customization and Optimization Guide\)](#)

AUTOSAR

Where to go from here

Information in this section

Supported AUTOSAR Releases.....	78
Memory Mapping.....	79
Static Memories and Constant Memories for Measurement and Calibration.....	79
Support for Rte_IsUpdated.....	80
AUTOSAR Import and Export Improvements.....	80

Supported AUTOSAR Releases

Supported AUTOSAR Releases

The following AUTOSAR Releases are supported:

AUTOSAR Release	Revision
4.3	4.3.0 ¹⁾
4.2	4.2.2 4.2.1
4.1	4.1.3 4.1.2 4.1.1
4.0	4.0.3 4.0.2

AUTOSAR Release	Revision
3.2	3.2.3
	3.2.2
	3.2.1
3.1	3.1.5
	3.1.4
	3.1.2
	3.1.0
3.0	3.0.7
	3.0.6
	3.0.4
	3.0.2
2.1	2.1.4

¹⁾ New in TargetLink 4.3

Memory Mapping

SwAddressMethods and MemorySections

You can use `SwAddressMethods` and `MemorySections` for memory mapping as defined by AUTOSAR.

Related documentation

- [Mapping Variables and Functions to Memory Sections \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [Decorating Production Code \(TargetLink Customization and Optimization Guide\)](#)
- [AR_MEMORY_MAPPING \(TargetLink Demo Models\)](#)

Static Memories and Constant Memories for Measurement and Calibration

Static memories and constant memories

TargetLink now supports static memories and constant memories for measurement and calibration as defined by AUTOSAR 4.x.

Related documentation:

- [Basics on Static and Constant Memories \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [How to Model Static Memories for Measurement \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

- [How to Model Constant Memories for Calibration \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Support for Rte_IsUpdated

Rte_IsUpdated

TargetLink now supports the **Rte_IsUpdated** RTE API function.

Related documentation

- [Basics on Checking the Update Flag of Data Elements in Explicit Sender-Receiver Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [How to Model Checks of the Update Flag in Explicit Sender-Receiver Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

AUTOSAR Import and Export Improvements

Arrays of structs

TargetLink can import and export AUTOSAR data prototypes that reference array of struct data types. The following AUTOSAR data prototypes are supported:

- Data elements (AUTOSAR 3.x/4.x)
- Interrunnable variables (AUTOSAR 4.x)
- Static memories (AUTOSAR 4.x)

Direct support for Array of Structs is available only for import and export. They are not supported in the model (as arrays of buses) and in code generation. For an example of how to use the imported data prototypes for modeling and code generation, see the [AR_ARRAY_OF_STRUCT_DATA](#) demo model.

Related documentation [AR_ARRAY_OF_STRUCT_DATA \(TargetLink Demo Models\)](#)

Related topics

Basics

[AR_ARRAY_OF_STRUCT_DATA \(TargetLink Demo Models\)](#)

Target Simulation (PIL)

Changes in the Target Simulation Modules

New and discontinued compiler versions

The following table shows the compiler versions that are now supported by TargetLink 4.3, refer to the New and No changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.

Microcontroller Family	Compiler	New	No Changes	Discontinued
ARM CortexM3	Keil	—	5.2	—
C16x	TASKING	—	8.6	—
MPC57xxVLE	Diab	—	5.9	—
	GreenHill	2016	—	2014
MPC560xVLE	Diab	—	5.9	—
	GreenHill	2016	—	2012,2014
RH850	GreenHill	2016	—	2015
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3	—
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	6.2	3.2	6.0
TriCore2xx	TASKING	6.2	—	6.0
	GCC	—	4.6	—
V850	GreenHill	2016	—	2015
XC22xx	TASKING	—	3.0	—

For more information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ( [Evaluation Board Reference](#)).

Note

For more PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Data Dictionary and Data Management

Further Improvements to the Data Dictionary

Empty Class property of a DD Variable object always causes an error in DD validation checks	The default <code>dsdd_validate</code> API command and the corresponding <code>Validate</code> function of the Data Dictionary Manager now always detect DD Variable objects with an empty <code>Class</code> property and return an error. In earlier versions of TargetLink, this might have been detected only during code generation.
--	---

Code Generator Options

New Code Generator Options

Overview of new Code Generator options	<p>The following new Code Generator options are available with TargetLink 4.3.</p> <ul style="list-style-type: none">▪ <code>AvoidStaticLocalScope</code> Specifies to avoid static storage duration for function-local variables.▪ <code>ImplementLookUpTableFunctionsWithPointerArithmetics</code> Enables the generation of pointer arithmetic in Look-Up Table functions, otherwise an index access is used where supported.▪ <code>OutputMATLABCodeInfo</code> Outputs XML files that contain information about MATLAB subfunctions and internal variables.▪ <code>ReportVariableVectorWidthDetails</code> When code is generated, a report on variable vector widths is generated. This report describes the blocks and ports where the propagation of <code>ExchangeableWidth</code> objects terminates and explains the reasons.
	<p>Related documentation For reference information on all Code Generator options, refer to Alphabetical List of Code Generator Options ( TargetLink Model Element Reference).</p>

Migration aspects of Code Generator options	For more information, refer to Migration Aspects Regarding Code Generator Options on page 319.
--	--

API Functions and Hook Scripts

New API Functions

List of new API functions	API Function	Purpose
	<code>t1CodeGenerationMetadata</code>	Saves and loads code generation meta data (DD Subsystem object).
	<code>t1GetArtifactLocation</code>	Returns the location of the code generation unit's generated artifacts .
	<code>t1IsCodeGenerationInProgress</code>	Determines if the code generation process is active.
	<code>t1Propman</code>	Command-line interface to the revised TargetLink Property Manager.

Related documentation:

- [t1Propman](#) ([TargetLink API Reference](#))
- [t1GetArtifactLocation](#) ([TargetLink API Reference](#))
- [t1CodeGenerationMetadata](#) ([TargetLink API Reference](#))
- [t1IsCodeGenerationInProgress](#) ([TargetLink API Reference](#))

Other

Where to go from here	Information in this section
	<p>New Property Manager..... 83</p> <p>General Enhancements and Changes..... 85</p> <p>TargetLink Demos..... 86</p>

New Property Manager

New version of the Property Manager	TargetLink 4.3 provides the revised Property Manager to support you in exploring and changing the TargetLink properties of large models.
--	--

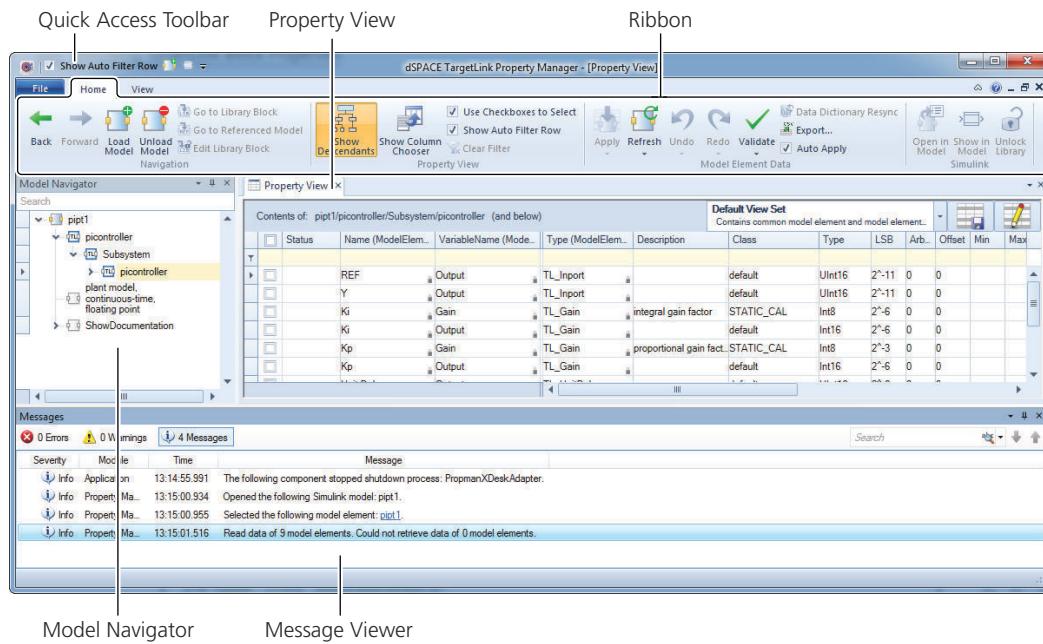
The Property Manager provides views for navigating through the Simulink model and for displaying and editing TargetLink properties of model elements.

Model elements can be, for example:

- Simulink subsystems
- TargetLink blocks
- Stateflow/MATLAB function objects

The Property Manager allows you to view and filter the properties of model elements and lets you simultaneously modify the same TargetLink property of several model elements . For Stateflow objects, this includes charts, events, states, and data.

The following illustration shows the new user interface of the Property Manager.



These are the main new user interface elements:

- The **Model Navigator** displays the hierarchy of models.
It lets you perform actions such as navigating through the model hierarchy and opening TargetLink dialogs of model elements (e.g. Stateflow objects). You can select a model element to display the TargetLink properties of this model element and (if applicable) underlying model elements in the **Property View**.
- The **Property View** is the working area in the Property Manager for displaying, editing, and validating the TargetLink properties of model elements and their variables. Each model element variable is displayed in an own row.
You can perform actions such as focusing on specific properties and multi-editing property values.
- The **Message Viewer** displays system messages in a chronological order. It lets you search for messages and filters the messages to be displayed.
- The **Ribbon** organizes, groups, and labels the commands of the Property Manager.

- The Quick Access Toolbar is an easy way to call commands. You can customize it to contain the commands you use most frequently.

Related Documentation

- [Modifying Multiple Properties at Once via the Property Manager \(TargetLink Preparation and Simulation Guide\)](#)
- [Property Manager \(TargetLink Tool and Utility Reference\)](#)
- [Property Manager - Migration Aspects](#) on page 322

General Enhancements and Changes

Improved user interfaces

The following improvements have been made to TargetLink's frontend:

TargetLink menu is always visible TargetLink now provides the TargetLink menu (with access to help, demos, etc.) in models and libraries that do not have TargetLink information (such as pure Simulink models). In addition, the **Make Library TargetLink Compliant** command is available for libraries.

Main dialog tooltips TargetLink now provides better tooltips in the main dialog.

Hyperlink for TargetLink File Export Utility After a successful code generation, a new Export hyperlink in the MATLAB Command window starts the TargetLink File Export Utility.

Improvements to TargetLink block dialogs TargetLink now provides access to Code Generator-relevant Simulink parameters in block dialogs of the following blocks:

- (Bus-) Import
- (Bus-) Outport
- Data Store Read / Write
- Rate Limiter
- TargetLink Stateflow chart dialog

Inactive MIL handler block

TargetLink allows the user to comment out the MIL handler block for simulations in order to simulate faster. When starting a simulation, TargetLink informs the user that a MIL Handler block is commented out. Note that some TargetLink MIL simulation features like Overflow checking are only available with an active MIL Handler block.

Support for A2L 1.6

TargetLink now supports Version 1.6.1 of the ASAM MCD-2 MC standard for A2L file export.

Related documentation [Exchanging A2L Files \(TargetLink Interoperation and Exchange Guide\)](#)

Code Generation Progress dialog TargetLink now displays a Code Generation Progress dialog when you start the code generation, including a Cancel button to abort a long running code generation.

Related documentation [Code Generation Progress Dialog \(TargetLink Tool and Utility Reference\)](#)

New options in style definition file TargetLink's style definition file has the following new options:

Option	Description
<TL:var-definition-comment show-default-scalings="false/true">	Show scaling comments for variables with default scaling.
<TL:decldef-section-comment show="true/false"/>	Do not show information about variable class and bit width in the declaration comments.

TargetLink Demos

New demo models

The following new demo models are available for TargetLink 4.3:

AR_ARRAY_OF_STRUCT_DATA The demo model demonstrates exemplarily how to access array elements from an array of struct via RTE API calls based on a Custom Code (type II) block.

Refer to [AR_ARRAY_OF_STRUCT_DATA \(TargetLink Demo Models\)](#).

AR_MEMORY_MAPPING The AR_MEMORY_MAPPING demo model demonstrates how to perform AUTOSAR memory mapping with TargetLink.

Refer to [AR_MEMORY_MAPPING \(TargetLink Demo Models\)](#).

BUS_CC The BUS_CC demo model shows the use of Simulink buses and TargetLink structured data types with the Custom Code (type II) block.

Refer to [BUS_CC \(TargetLink Demo Models\)](#).

MODULAR_DEVELOPMENT The demo shows how the artifacts generated for incremental code generation units, such the production code or HTML documentation, can be integrated and used in an integration model.

Refer to [MODULAR_DEVELOPMENT \(TargetLink Demo Models\)](#).

Extended demo models

The following demo models have been extended for TargetLink 4.3:

AR_COLLISION_DETECTION The demo model has been extended to demonstrate the use of bus signals at the interface of a Custom Code (type II) block.

Refer to [AR_COLLISION_DETECTION \(TargetLink Demo Models\)](#).

CUSTOM_ENHANCEMENT The subsystem which is to be prepared now contains two Simulink Tapped Delay blocks.

Refer to [CUSTOM_ENHANCEMENT](#) ([TargetLink Demo Models](#)).

New Features of TargetLink 4.2

Where to go from here	Information in this section
	Modeling in Simulink or Stateflow 87
	Code Generation Core Functionality 90
	AUTOSAR 96
	Target Simulation (PIL) 99
	Data Dictionary and Data Management 100
	Code Generator Options 102
	Tool Chain Integration 102
	Other 104
	API Functions and Hook Scripts 107

Modeling in Simulink or Stateflow

Where to go from here	Information in this section
	Newly Supported Simulink Blocks 88
	Improved Data Store Blocks and Bus Support 88
	Improvements to Stateflow Code Generation 89
	Improvements to Model Referencing 90

Newly Supported Simulink Blocks

Supported Simulink blocks (code-relevant)	<p>TargetLink now additionally supports the following Simulink block, which affects TargetLink code generation:</p> <ul style="list-style-type: none">▪ Vector Concatenate This block is similar to the Matrix Concatenate block. These blocks differ only in their Mode setting, Vector versus Multidimensional array.
Supported Simulink blocks (not code-relevant)	<p>TargetLink now supports the following Simulink blocks, which do not affect TargetLink code generation:</p> <ul style="list-style-type: none">▪ Data Type Duplicate▪ Data Type Propagation▪ Floating Scope▪ XY Graph▪ Stop Simulation▪ Timed-Based Linearization▪ Trigger-Based Linearization <p>Related documentation</p> <ul style="list-style-type: none">▪ Code-Irrelevant Simulink Blocks (TargetLink Model Element Reference)

Improved Data Store Blocks and Bus Support

Element selection and bus-capable Data Store	<p>This TargetLink version supports bus signals in Data Store Memory, Data Store Read and Data Store Write blocks. You can map entire buses to predefined structure type definitions and to predefined structure variables. This is especially helpful if the bus consists of many bus elements, and/or the type or variable is used several times in a model or across models.</p>
Obsolete limitations	<p>Some limitations concerning Data Store blocks have become obsolete. For more information, refer to Obsolete Limitations on page 413.</p> <ul style="list-style-type: none">▪ The number of input and output ports (of Data Store Read and Data Store Write blocks) can now be more than one.▪ Partial reading from and writing to data stores is now also supported by TargetLink.

Related documentation

- [Basics on Modeling Buses via Data Store Blocks](#) ([TargetLink Preparation and Simulation Guide](#))
 - [Basics on the Representation of Buses in the Production Code](#) ([TargetLink Preparation and Simulation Guide](#))
-

Bus-related API commands

You can create Simulink.Bus objects from structured DD objects. Vice versa, you can create DD Variable objects from Simulink.Bus objects.

Related documentation

- [Basics on Modeling Buses via Data Store Blocks](#) ([TargetLink Preparation and Simulation Guide](#))
- [tlSimulinkBusObject](#) ([TargetLink API Reference](#))

Improvements to Stateflow Code Generation

Support for Superstep semantics

You can now work with Superstep semantics in state machines. This lets you perform all possible state transitions at once.

Related documentation

- [Basics on Working with Superstep Semantics](#) ([TargetLink Preparation and Simulation Guide](#))
-

Support for functions with multiple output data

TargetLink now supports Stateflow functions (e.g., graphical functions) with multiple output data.

Related documentation

- [Working with Stateflow](#) ([TargetLink Preparation and Simulation Guide](#))
-

Improved support for monitoring state activities

For the monitoring of state activity via Stateflow active state data there are three activity modes: **Self Activity**, **Child Activity** and **Leaf State Activity**.

TargetLink now supports **Self Activity** and **Child Activity** activity modes and generates optimized production code, including C code enums or C preprocessor macros for state encoding. However, **Leaf State Activity** is not supported.

Related documentation

- [Basics on Working with Active State Data](#) ([TargetLink Preparation and Simulation Guide](#))

Improvements to Model Referencing

Enable blocks on model root level	With this TargetLink version, you can control the execution of referenced models not only via Trigger block but also via Enable block.
	Related documentation <ul style="list-style-type: none">▪ Basics on Model Referencing (TargetLink Customization and Optimization Guide)

Code Generation Core Functionality

Where to go from here	Information in this section										
	<table><tr><td>MISRA C Compliance.....</td><td>90</td></tr><tr><td>Improved Code Stability.....</td><td>91</td></tr><tr><td>Improved Code Efficiency.....</td><td>91</td></tr><tr><td>Improved Enumeration Support.....</td><td>95</td></tr><tr><td>More Flexibility When Using TargetLink Custom Code Blocks.....</td><td>95</td></tr></table>	MISRA C Compliance	90	Improved Code Stability	91	Improved Code Efficiency	91	Improved Enumeration Support	95	More Flexibility When Using TargetLink Custom Code Blocks	95
MISRA C Compliance	90										
Improved Code Stability	91										
Improved Code Efficiency	91										
Improved Enumeration Support	95										
More Flexibility When Using TargetLink Custom Code Blocks	95										

MISRA C Compliance

Improvements for MISRA C	Several improvements were made to TargetLink's Fixed-Point Library and to the Code Generator itself to improve the compliance with individual MISRA C rules. The improvements include the following: <ul style="list-style-type: none">▪ Functions have only one return path (MISRA C:2012 - 15.5).▪ Variables are always created in the minimal useful scope (Fixed-Point Library).▪ Less unreachable code (MISRA C:2012 - 2.1, 2.2, 14.3).▪ Improved support for the conversion rules of MISRA C:2012 by adding casts for return values or comparisons.▪ TargetLink now supports the C99 Bool type for the implementation of Bool. Related documentation <ul style="list-style-type: none">▪ None
---------------------------------	--

Improved Code Stability

Changed calculation sequence in atomic subsystems

The production code calculation sequence in atomic subsystems can change in contexts where the block execution order was not completely determined by the control and data flow.

Reason To make the production code's block scheduling more resilient against model changes (like adding an Addfile or Unit Delay block).

Migration issue Depending on your specific situation and use cases, obtaining production code sorting as in previous TargetLink versions might be possible by applying a temporary workaround. Contact dSPACE product support for TargetLink.

Improved Code Efficiency

Omitting init section for Assignment blocks

The code for Assignment blocks changed in the following context:

- The block is used to model reduced rewrites to NVRAM in AUTOSAR NvData communication.
- The block does not have more than one successor block.
- The block is not placed within an iterated subsystem.

In this case, TargetLink no longer generates the `output initialization`, as shown in the following code examples:

TargetLink ≤ 4.1

```
p_Err = Rte_IRead_Run_PR_NV_Err();
...
SCtrl12_Logical_Operator = SCtrl12_Relational_Operator &&
SCtrl12_Relational_Operator1;

for (Aux_ = 0; Aux_ < 10; Aux_++)
{
    /* Assignment: pidcontroller/Efficient_Write_To_NVRAM - output
    initialization */
    SCtrl11_Efficient_Write_To_NVRAM[Aux_] = p_Err[Aux_];
}
SCtrl11_Efficient_Write_To_NVRAM[0] = SCtrl12_Logical_Operator;

p_Err = Rte_IRead_Run_PR_NV_Err();
if (p_Err[0] != SCtrl11_Efficient_Write_To_NVRAM[0]) {
    p_Err_a = Rte_IWriteRef_Run_PR_NV_Err();
    p_Err_a[0] = SCtrl11_Efficient_Write_To_NVRAM[0];
}
```

TargetLink 4.2

```

p_Err = Rte_IRead_Run_PR_NV_Err();
...
SCtrl12_Logical_Operator = SCtrl12_Relational_Operator &&
SCtrl12_Relational_Operator1;
SCtrl11_Efficient_Write_To_NVRAM[0] = SCtrl12_Logical_Operator;
    p_Err = Rte_IRead_Run_PR_NV_Err();
if (p_Err[0] != SCtrl11_Efficient_Write_To_NVRAM[0]) {
    p_Err_a = Rte_IWriteRef_Run_PR_NV_Err();
    p_Err_a[0] = SCtrl11_Efficient_Write_To_NVRAM[0];
}

```

Benefit TargetLink can now completely optimize **SCtrl11_Efficient_Write_To_NVRAM**, which results in increased efficiency

Omitting superfluous casts

TargetLink now removes casts from expressions like **(Type) <Boolean operation, variable, macro>** if they are used in the following contexts:

- In **if**-, **while**-, **do ... while**-, **for**-, and **?:** conditions
- As operand of logical operations

This also holds for comparisons with **0** or **1** that occur in the contexts mentioned above.

In addition, TargetLink omits casts in switch conditions that cast constants and do not change the value. Usually, this allows for further optimizations such as replacing the switch statement by the code of one of the case branches or the default branch.

TargetLink ≤ 4.1	TargetLink 4.2
<code>switch((Type) <constant>)</code>	<code>switch(<constant cast to T>)</code>

Definitions of variables with local scope

TargetLink now makes sure that variables of local scope are never defined directly at the start of loop compound statements. They are now always consistently defined at the top of the function. Keep in mind that optimization might reduce function-local variables to minimal block scope in compound statements nested within loops, depending on the option `ReduceScopeOfVariablesOnlyDownToFunctionLevel`.

TargetLink ≤ 4.1	TargetLink 4.2
<pre> void Subsystem(void) { for (Aux_b = 0; Aux_b < Width; Aux_b++) { UInt8 T1AuxVar; /* ... */ } } </pre>	<pre> void Subsystem(void) { UInt8 T1AuxVar; for (Aux_b = 0; Aux_b < Width; Aux_b++) { /* ... */ } } </pre>

All identifiers of function-local variables and macros now have function-wide visibility, i.e., are treated as if the respective variable or macro is defined at the beginning of the function. This primarily affects the identifiers of auxiliary variables in unoptimized code.

TargetLink ≤ 4.1

```

for (Aux_b = 0; Aux_b < BlockWidth; Aux_b++)
{
    /* SLLutLocal: Default storage class for local variables | Width: 8 */
    UInt8 TlAuxVar; /* Index saturation for Direct Look-Up Table block */
    /* Subsystem/TableAsParam_BlockVW */
    BlockVar4[Aux_b] = Sa1_TableAsParam_BlockVW_table[TlAuxVar];
}

...
for (Aux_d = 0; Aux_d < BlockWidth; Aux_d++)
{
    /* Subsystem/TableAsParam_BlockVW */
    BlockVar4[Aux_b] = Sa1_TableAsParam_BlockVW_table[TlAuxVar];
}

...
for (Aux_d = 0; Aux_d < BlockWidth; Aux_d++)
{
    /* SLLutLocal: Default storage class for local variables | Width: 8 */
    UInt8 TlAuxVar; /* Index saturation for Direct Look-Up Table block */
    TlAuxVar = C_U8SATI16_SATb(Sa1_In[Aux_d], 9, 0);
    /* Subsystem/TableAsInput_TableAndBlockVW */
    BlockVar3[Aux_d] = TableVar1[TlAuxVar];
}

```

TargetLink 4.2

```

for (Aux_b = 0; Aux_b < BlockWidth; Aux_b++)
{
    /* SLLutLocal: Default storage class for local variables | Width: 8 */
    UInt8 TlAuxVar_a; /* Index saturation for Direct Look-Up Table block */
    TlAuxVar_a = C_U8SATI16_SATb(Sa1_In[Aux_b], 9, 0);
    /* Subsystem/TableAsParam_BlockVW */
    BlockVar4[Aux_b] = Sa1_TableAsParam_BlockVW_table[TlAuxVar_a];
}

...
for (Aux_d = 0; Aux_d < BlockWidth; Aux_d++)
{
    /* SLLutLocal: Default storage class for local variables | Width: 8 */
    UInt8 TlAuxVar_b; /* Index saturation for Direct Look-Up Table block */
    TlAuxVar_b = C_U8SATI16_SATb(Sa1_In[Aux_d], 9, 0);
    /* Subsystem/TableAsInput_TableAndBlockVW */
    BlockVar3[Aux_d] = TableVar1[TlAuxVar_b];
}

```

Reason To establish consistent naming of auxiliary variables across code patterns, including different auxiliary variable identifiers for different blocks.

Optimization of logical expressions

TargetLink can now optimize logical expressions containing constants other than zero:

TargetLink ≤ 4.1	TargetLink 4.2
Logical And	
<code>1 && boolexpr => boolexpr</code>	<code><non-zero const> && boolexpr => boolexpr</code>
Logical Or	
<code>1 boolexpr => optimized to: 1</code>	<code><non-zero const> boolexpr => optimized to: 1</code>

Floating-point modulo and remainder

The code generated for the Math block can change if one of its inputs or its output has a floating-point type:

TargetLink ≤ 4.1 For a Math block with modulo function whose output is `Float64` and inputs are `Float32` and `Int16`:

```
if (I16DenomIn != 0.F) {
    F64_F32I16_ = F32NumIn - (((Float64)
I16DenomIn) * ((Float64) (Int32) (F32NumIn / ((Float64) I16DenomIn))));
...
}
```

For a Math block with modulo function whose output is `Int16` and saturated and inputs are `Float32`:

```
if (F32DenomInInt != 0.F) {
...
} else {
    I16_F32F32_ = (Int16) F32NumIn;
}
```

TargetLink 4.2 For a Math block with modulo function whose output is `Float64` and inputs are `Float32` and `Int16`:

```
if (I16DenomIn != 0) {
    F64_F32I16_ = ((Float64) F32NumIn) - (((Float64)
I16DenomIn) * ((Float64) (Int32) (((Float64) F32NumIn) / ((Float64) I16DenomIn))));
```

For a Math block with modulo function whose output is `Int16` and saturated and inputs are `Float32`:

```
if (F32DenomIn != 0.F) {
...
} else {
    if (F32NumIn > 32767.F) {
        I16_F32F32_ = 32767;
    }
    else {
        if (F32NumInInt < -32768.F) {
            I16_F32F32_ = -32768;
        }
        else {
            I16_F32F32_ = (Int16) F32NumIn;
        }
    }
}
```

Details The code can change because TargetLink now behaves as follows:

- Explicitly cast all operands to the type the operation is supposed to be calculated in.
- Calculate all intermediate operations using `Float64` if the operation result or one of its operands is `Float64`.
- Saturate the code in the `else` branch of conditional control flows.

Reason Improves precision and suppresses implicit casts to comply with MISRA C.

Improved Enumeration Support

Enumerations in the generated code

TargetLink supports the use of Simulink enumerations in the model (including Stateflow) as well as their implementation in the generated code. Simulink enumerations are implemented either as *C enums*, as *macros*, or as integers. Hence, simulation (MIL and SIL/PIL) is possible.

The following code example is from the ENUM_TYPES demo model:

```
typedef enum eBlinkMode_tag {
    eBlinkMode_NONE = 0,
    eBlinkMode_BLINKER_ON = 1,
    eBlinkMode_BLINKER_OFF = 2
} eBlinkMode;

switch (Ca1_ChartMode) {
    case CHARTMODE_DO_BLINK: {
        if (!(Sa1_ENABLE)) {
            Ca1_BlinkState = eBlinkMode_NONE;
            Ca1_ChartMode = CHARTMODE_OFF;
        }
        else {
            if (Ca1_BlinkState == eBlinkMode_BLINKER_ON) {
                Ca1_BlinkState = eBlinkMode_BLINKER_OFF;
            }
            else {
                Ca1_BlinkState = eBlinkMode_BLINKER_ON;
            }
        }
    }
}
```

Related documentation

- [Applying Simulink Enumeration Data Types in TargetLink \(TargetLink Preparation and Simulation Guide\)](#)
- [ENUM_TYPES \(TargetLink Demo Models\)](#)

More Flexibility When Using TargetLink Custom Code Blocks

Substitution of strings in custom code template files

TargetLink custom code blocks now support the use of placeholders for arbitrary strings in the custom code template file for each block instance. The placeholders can be replaced during code generation by arbitrary, user-provided strings, which makes custom code template files a lot more flexible.

Related documentation

- [Basics on Custom Code Files \(TargetLink Preparation and Simulation Guide\)](#)
- [VARIABLE_INITIALIZATION \(TargetLink Demo Models\)](#)

Removing braces from code sections

Braces that encapsulate a Custom Code section can be removed by using the new `nobraces` keyword at the beginning of this code section.

Related documentation Basics on Custom Code Files ( TargetLink Preparation and Simulation Guide)

AUTOSAR

Where to go from here

Information in this section

Supported AUTOSAR Releases.....	96
Asynchronous Client-Server Communication.....	97
Data Transformer in Client-Server Communication.....	97
Non-Scalar Interrunnable Variables.....	98
Improved Roundtrip with SystemDesk and other AUTOSAR Tools.....	98
Miscellaneous AUTOSAR Features.....	98

Supported AUTOSAR Releases

Supported AUTOSAR Releases

The following AUTOSAR Releases are supported:

AUTOSAR Release	Revision
4.2	4.2.2 ¹⁾
	4.2.1
4.1	4.1.3
	4.1.2
	4.1.1
4.0	4.0.3
	4.0.2
3.2	3.2.3
	3.2.2
	3.2.1

AUTOSAR Release	Revision
3.1	3.1.5
	3.1.4
	3.1.2
	3.1.0
3.0	3.0.7
	3.0.6
	3.0.4
	3.0.2
2.1	2.1.4

¹⁾ New in TargetLink 4.2 (and also supported since TargetLink 4.1 patch 1)

Asynchronous Client-Server Communication

New options for modeling asynchronous client-server communication

You can now model asynchronous client-server communication to generate calls of the `Rte_Call` and `Rte_Result` API functions in your production code.

Related documentation

- [Modeling Client-Server Communication](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))
- [Changes in TargetLink and TargetLink Data Dictionary API Functions](#) on page 328
- [AR_COLLISION_DETECTION](#) ([TargetLink Demo Models](#))

Data Transformer in Client-Server Communication

Transformer error logic

You can now model transformer error logic for client-server communication.

Related documentation

- [Modeling Transformer Error Logic](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))
- [AR_COLLISION_DETECTION](#) ([TargetLink Demo Models](#))

Non-Scalar Interrunnable Variables

Support of non-scalar interrunnable variables

You can now model non-scalar interruptable variables.

Related documentation

- Modeling Interrunnable Communication ( TargetLink Classic AUTOSAR Modeling Guide)
- Migrating Interrunnable Variable Specifications on page 330

Improved Roundtrip with SystemDesk and other AUTOSAR Tools

Export improvements

The roundtrip between TargetLink and SystemDesk was improved. You can now specify package information for AUTOSAR elements that are implicitly generated during export via DD ExportRule objects. The internal behavior of a software component can be exported into a separate file. Moreover, units can also be exported in separate files and packages.

Related documentation

- Basics on Packages ( TargetLink Interoperation and Exchange Guide)
- ExportRule

Miscellaneous AUTOSAR Features

Improved import and export

TargetLink can now import and export CompuMethod elements whose category is set to BITFIELD_TEXTABLE.

Related documentation

- None

Data Store Memory blocks outside of runnable subsystems

You can now use Data Store Memory blocks to model NvData communication outside of  runnable subsystems.

Related documentation

- None

Data store blocks for simulation

You can now place Data Store Read and Data Store Write blocks outside of runnable subsystems to access Data Store Memory blocks that model NvData or interruptable communication. This lets you directly access the communication variable in the simulation frame for simulation purposes.

Target Simulation (PIL)

Where to go from here

Information in this section

High-Speed Baud Rates for PIL Simulations.....	99
Changes in the Target Simulation Modules.....	99

High-Speed Baud Rates for PIL Simulations

Virtual COM ports

TargetLink supports faster baud rates to transfer data to/from TargetLink evaluation boards (EVB) for PIL simulations. Virtual COM ports support communication baud rates that are 2-, 4-, and 8-times faster than 115,200, but the actual communication baud rate depends on the hardware. Note that the download baud rate is not affected, its maximum limit remains 115,200 (depending on the EVB even less).

Related documentation [Topic Settings \(TargetLink Tool and Utility Reference\)](#)

Changes in the Target Simulation Modules

New and discontinued compiler versions

The following table shows the compiler versions that are now supported by TargetLink 4.2, refer to the New and No changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.

Microcontroller Family	Compiler	New	No Changes	Discontinued
ARM CortexM3	Keil	5.2	—	5.1
C16x	TASKING	—	8.6	—
MPC57xxVLE	Diab	—	5.9	—
	GreenHill	—	2014	—
MPC560xVLE	Diab	—	5.9	—
	GreenHill	—	2012, 2014	—
RH850	GreenHill	2015	—	2014
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3	—

Microcontroller Family	Compiler	New	No Changes	Discontinued
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	6.0	3.2	5.0
TriCore2xx	TASKING	6.0	—	5.0
	GCC	—	4.6	—
V850	GreenHill	2015	—	2014
XC22xx	TASKING	—	3.0	—

For more information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ([Evaluation Board Reference](#)).

Note

For further PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Data Dictionary and Data Management

Where to go from here

Information in this section

- | | |
|---|-----|
| Improvements to the Data Dictionary..... | 100 |
| Predefined Code Generator Option Sets in Data Dictionaries..... | 101 |

Improvements to the Data Dictionary

DD Comparison Pane (GUI) via Windows Command Prompt

You can now graphically compare two DD files via Windows Command prompt. You can start the Data Dictionary Manager and its DD Comparison pane stand-alone without a MATLAB environment, e.g., from inside a version control system.

Related documentation

- [Basics on Comparing and Merging DD Objects and Workspaces Using the DD Two-Way Comparison](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

Custom command calls	You can call your own commands from within the Data Dictionary passing an arbitrary number of parameters. Custom commands can be external programs, your own M scripts or other preparatory scripts or functions you want to call.
-----------------------------	--

Related documentation

- [How to Specify Custom Command Calls](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))
 - [DD_CUSTOM_COMMAND](#) ([TargetLink Demo Models](#))
-

Embedded help	Embedded help was added to several objects and properties in the Data Dictionary Manager.
----------------------	---

Improved handling of non-integer types	Via the Plain Variable dialog, you can now specify a local scaling for non-integer types in the Data Dictionary.
---	--

Additionally, the Adjust to Typedef context-menu command now also works for non-integer types.

Related documentation

- [Adjust to Typedef](#) ([TargetLink Data Dictionary Manager Reference](#))

Predefined Code Generator Option Sets in Data Dictionaries

Adjusting code generation via DD optionsets	TargetLink Data Dictionaries (based on predefined system templates, e.g. dsdd_master_advanced.dd) now provide different option sets to easily adjust the production code generation for specific objectives, e.g., high MISRA C compliance or speed versus code-size.
--	---

Related documentation

- [Basics on Configuring the Code Generator for Production Code Generation](#) ([TargetLink Customization and Optimization Guide](#))

Code Generator Options

New Code Generator Options

Overview of new Code Generator options

The following new Code Generator option is available with TargetLink 4.2.

- [ForceBooleanOperandsInBooleanOperations](#)

Inserts additional *unequal to zero* ($x \neq 0$) comparisons into the generated code to force a Boolean context in logical operations and control flow expressions for non-Boolean types to achieve MISRA C compliance even in such modeling situations.

Related documentation

- [ForceBooleanOperandsInBooleanOperations](#) ([TargetLink Model Element Reference](#)).

For reference information on all Code Generator options, refer to [Alphabetical List of Code Generator Options](#) ([TargetLink Model Element Reference](#)).

Migration aspects of Code Generator options

Migration aspects include:

- Removed Code Generator options
- Changed Code Generator options
- Recommended compatibility settings
- Basics on changed defaults

For details, refer to [Migration Aspects Regarding Code Generator Options](#) on page 333.

Tool Chain Integration

Where to go from here

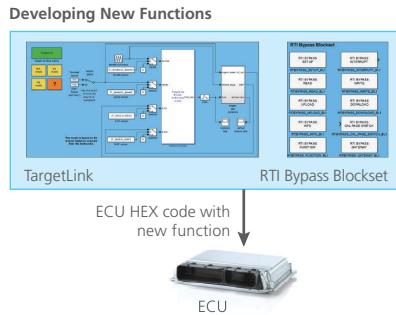
Information in this section

Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing.....	103
Build Binary Files for Functional Mock-up Units.....	103

Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing

Modeling with RTI Bypass blocks and generating code for on-target bypassing

In TargetLink, you can model with the RTI Bypass Blockset. The RTI Bypass Blockset can be used to configure ECU interfaces and implement new ECU functions for bypassing or for tests. You can then use TargetLink's Code Generator for code generation in order to add a new or replace an existing control function directly on the target hardware (on-target bypassing). This is done by using the free resources on the target ECU. On-target bypassing is also often called *On-Target Prototyping*, because you can develop and replace ECU functions without the use of additional (external) hardware.



Related documentation

- [Basics on Modeling with RTI Bypass Blocks and Generating Code for On-Target Bypassing](#) ([TargetLink Interoperation and Exchange Guide](#))
- Also refer to the RTI Bypass Blockset documentation: [General Information on the RTI Bypass Blockset](#) ([RTI Bypass Blockset Reference](#))

Related topics

Basics

[General Information on the RTI Bypass Blockset](#) ([RTI Bypass Blockset Reference](#))

Build Binary Files for Functional Mock-up Units

Improvements to Functional Mock-up Units

The Functional Mock-up Units (FMUs) generated and exported from TargetLink subsystems can now contain pre-built Windows DLL files for 32 and 64-bit. This lets you use these FMUs with FMI-compliant tools that cannot build FMUs on their own.

Related documentation

- [Basics on Exporting FMUs from TargetLink \(TargetLink Interoperation and Exchange Guide\)](#)
- [TargetLink FMU Manager \(TargetLink Tool and Utility Reference\)](#)

Other

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

General Enhancements and Changes.....	104
TargetLink Demos.....	106

General Enhancements and Changes

Improved documentation generation

HTML support TargetLink now supports the adding of HTML files in the generated documentation.

Incremental documentation generation TargetLink now supports incremental documentation generation which means that you can reuse parts of already generated documentation during documentation generation, for example, for an integration model.

Support for Scalable Vector Graphics TargetLink now supports SVG in the generated documentation. SVG is the default format. PNG is also possible. You can set the file format in the `ImageFormatType` property of the `t1doc` command. However, EPS is no longer supported. Also refer to [Discontinued TargetLink Features](#) on page 411.

Improved usability for embedded commands You can now select predefined configurations of embedded commands in the Autodoc Customization block dialog instead of typing them. TargetLink inserts the selected command into its edit field using the correct syntax.

Categorizing and filtering blocks You can categorize and filter Autodoc Customization blocks so that you can easily control which blocks are considered or left out by TargetLink during the documentation generation.

Hook script for further customization You can now use a new hook script to manage Autodoc Customization blocks. For example, you can set the order, in which Autodoc Customization blocks are generated. The hook script always overwrites competing settings specified at block level.

Related documentation

- [Basics on Documentation Generation \(TargetLink Interoperation and Exchange Guide\)](#)

Faster dialog opening

TargetLink now opens the Main dialog and other block dialogs twice as fast as with TargetLink 4.1.

Related documentation

- None

Defining Simulink data types for ddv IC structures

TargetLink now lets you map Simulink data types from the components of structured DD Variable objects used to generate IC structures (ddv).

Related documentation

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

New Coded Type for C99 _Bool supported in TargetConfig.xml

You can now use the C99 _Bool data type, i.e., TargetLink's `Bool` typedef maps to `_Bool`.

Introducing this coded type makes the production code more manageable with some MISRA C checkers. Currently, TargetLink provides its own `Bool` data typedef that is MISRA-compliant. However, some MISRA checkers do not accept this data type, which causes false positive violations. The C99 _Bool data type is always recognized as an effective `boolean` data type during MISRA compliance checking. This might reduce the number of false positive violations of MISRA rules.

Tip

To adjust `TargetConfig.xml` settings, you can create a new target or clone an existing one, especially for PIL simulation.

New Generic C99 code generation target In addition, you can use the new Generic C99 code generation target in the TargetLink Main Dialog. This uses the C99 _Bool data type as default.

Note

The use of `_Bool` also depends on the C compiler. For more information, refer to your target compiler's manual.

For SIL compilation, the supported MEX/SIL compilers support the `_Bool` data type with their default compiler options.

Related documentation

- [Example of Controlling the Generation of the File Containing TargetLink Base Data Types \(TargetLink Customization and Optimization Guide\)](#)

- [How to Clone Target/Compiler Combinations to Outside the TargetLink Installation](#) ([TargetLink Customization and Optimization Guide](#))
-

Support of Windows-1252 character set

TargetLink now also supports the Windows-1252 (CP1252, Western European) character set.

Related documentation

- [Basics on the Code Generation Report](#) ([TargetLink Preparation and Simulation Guide](#))
 - [How to Specify the Used Character Set](#) ([TargetLink Interoperation and Exchange Guide](#))
-

Signal injection and tunneling for struct variables

You can now access the whole structure variable and not only its components when injecting or tunneling signals.

The associated Simulink.Signal object must be of the Bus datatype whose hierarchy tree corresponds to the hierarchy tree of the structure variable. The names of the nodes are not taken into account.

Related documentation

- [Basics on Injecting or Tunneling Signals During Simulation](#) ([TargetLink Preparation and Simulation Guide](#))
- [Modeling Buses via TargetLink Blocks](#) ([TargetLink Preparation and Simulation Guide](#))

TargetLink Demos

New demos

The following new demos come with TargetLink

AR_COLLISION_DETECTION This new demo shows the following features:

- Modeling asynchronous client-server communication to generate calls of the `Rte_Call` and `Rte_Result` API functions in your production code.
- Modeling transformer error logic for client-server communication.

Refer to [AR_COLLISION_DETECTION](#) ([TargetLink Demo Models](#)).

AUTODOC_GENERATION This new demo shows the following features:

- How to control the general structure and content of the document by using a dedicated M script to call the documentation generation
- How to work with Autodoc Customization blocks to control which functions should be documented
- How to insert user-provided content (texts, images, HTML commands, ...) into the documentation generation process

Refer to [AUTODOC_GENERATION](#) ([TargetLink Demo Models](#)).

DD_CUSTOM_COMMAND This new demo shows the following features:

- You can call your own commands from within the Data Dictionary. You can also specify additional arguments. Custom commands can be external programs, your own M scripts or other preparatory scripts or functions you want to call. The dd_customcommand demo contains four different custom command calls in the Pool area of the Data Dictionary Manager.

Refer to [DD_CUSTOM_COMMAND](#) ([TargetLink Demo Models](#)).

EMBEDDING_EXTERNAL_CODE This new demo shows the following features:

- The demo shows different ways of embedding external code, e.g. via a Function block or a Custom Code block.

Refer to [EMBEDDING_EXTERNAL_CODE](#) ([TargetLink Demo Models](#)).

ENUM_TYPES This new demo shows the following features:

- Using Enumerations in model and generated production code, for example with Stateflow active state data.

Refer to [ENUM_TYPES](#) ([TargetLink Demo Models](#)).

FUNCTION_VARIANTS This new demo shows the following features:

- You can choose different binding times for function variants, e.g. while modeling or before generating production code.

Refer to [FUNCTION_VARIANTS](#) ([TargetLink Demo Models](#)).

VARIABLE_INITIALIZATION This new demo shows the following features:

- Different ways to initialize variables, e.g. via a Main Restart function.

Refer to [VARIABLE_INITIALIZATION](#) ([TargetLink Demo Models](#)).

Extended demos

The following demos contain new feature demonstrations:

BUS_STRUCT This changed demo now also shows the following features:

- How to handle buses with Data Stores.

Refer to [BUS_STRUCT](#) ([TargetLink Demo Models](#)).

API Functions and Hook Scripts

Where to go from here

Information in this section

New API Functions	108
New Hook Scripts	108

New API Functions

Overview of new API functions	API Function	Purpose
	tlEnumDataType (TargetLink API Reference)	Imports a Simulink enumeration data type to the Data Dictionary.
	tlExecuteDDCustomCommand	Executes a Data Dictionary CustomCommand or CustomCommandGroup object.
	tlSimulinkBusObject	Processes Simulink.Bus objects in TargetLink.
	tlStartCallbackWithTimer	Starts a callback within a timer.

Related documentation:

- [tlEnumDataType \(TargetLink API Reference\)](#)
- [tlExecuteDDCustomCommand \(TargetLink API Reference\)](#)
- [tlSimulinkBusObject \(TargetLink API Reference\)](#)
- [tlStartCallbackWithTimer \(TargetLink API Reference\)](#)

Related topics

References

[tlStartCallbackWithTimer \(TargetLink API Reference\)](#)

New Hook Scripts

Overview of new hook scripts

TargetLink provides the following new hook scripts:

Hook Script	Purpose
tl_post_add_operationresultprovidersubsystem_hook	Lets you customize AUTOSAR frame model generation.
tl_post_autodoc_filter_hook	Lets you enter your own commands, e.g., to filter the list of the found Autodoc Customization blocks.

Related documentation

- [tl_post_add_operationresultprovidersubsystem_hook \(TargetLink File Reference\)](#)
- [tl_post_autodoc_filter_hook \(TargetLink File Reference\)](#)

New Features of TargetLink 4.1

Where to go from here

Information in this section

Modeling in Simulink or Stateflow.....	109
Code Generation Core Functionality.....	111
Component-Based Development.....	116
AUTOSAR.....	117
Target Simulation (PIL).....	120
Data Dictionary and Data Management.....	122
Code Generator Options.....	125
Tool Chain Integration.....	126
Other.....	127
API Functions and Hook Functions.....	131

Modeling in Simulink or Stateflow

Where to go from here

Information in this section

Newly Supported Simulink Blocks.....	109
Improvements to Custom Look-up Tables.....	110
Support of Simulink's Simplified Mode and IC Structures.....	110
Support of Structures in Stateflow Action Language.....	111

Newly Supported Simulink Blocks

Supported Simulink blocks

TargetLink now supports the following Simulink blocks:

- Signal Conversion block:

TargetLink supports the Simulink Signal Conversion block, which can be used to rescale bus signals, for example.

- Bus Assignment block:
A supported Simulink block that simplifies modeling with Simulink buses.

Related documentation

- [Signal Conversion Block](#) ([TargetLink Model Element Reference](#))
- [Code-Relevant Simulink Blocks](#) ([TargetLink Model Element Reference](#))

Improvements to Custom Look-up Tables

Variables inherit an input signal's dimension

Variables specified via the `t1script` command can inherit their dimension from an input signal. This allows, for example, the implementation of a last index state variable for the Local search table search method.

Related documentation

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

Support of vectors and matrices

In addition to scalar signals, the custom look-up script mechanism now supports vector and matrix input signals.

Related documentation

- [Obsolete Limitations](#) on page 416

New demo model

The new demo model TABLE1D_USR_LOCAL shows how custom look-up tables process vector and matrix input signals and how you can implement a local search.

Related documentation

- [Lookup Tables, User-Written Lookup Functions](#) ([TargetLink Demo Models](#))
- [TABLE1D_USR_LOCAL](#) ([TargetLink Demo Models](#))

Support of Simulink's Simplified Mode and IC Structures

Simulink's simplified initialization mode and IC structures

In order to determine initial values that are not specified in the model, TargetLink supports Simulink's simplified initialization mode, which you can select by setting Simulink's Underspecified initialization detected parameter to **Simplified**.

Additionally, in this mode you can initialize bus-capable blocks by using Simulink's initial condition (IC) structures. These structs are now also supported by TargetLink 4.1.

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

Support of Structures in Stateflow Action Language

Bus signals at Stateflow charts

TargetLink now supports Simulink data objects using the `Simulink.Bus` data type for Stateflow variables, and Simulink buses at the inputs and outputs of Stateflow charts and at Stateflow-internal data. To access those signals, TargetLink supports structures in the Stateflow action language. The associated data objects must reference either a structured `Typedef` or a structured `DD` Variable object.

Related documentation

- [Basics on the Representation of Buses in the Production Code](#) ([TargetLink Preparation and Simulation Guide](#))
- [Basics on the Compatibility of Buses and Predefined Structs](#) ([TargetLink Customization and Optimization Guide](#))

Code Generation Core Functionality

Where to go from here

Information in this section

MISRA-C Compliance	111
Improved Code Efficiency	112

MISRA-C Compliance

Improvements to TargetLink's Fixed-Point Library

Several improvements were made to TargetLink's Fixed-Point Library so it complies with MISRA-C. The improvements include the following:

- The value of an expression of integer type shall not be implicitly converted to a different underlying type if a) it is not a conversion to a wider integer type of the same signedness, b) the expression is complex, c) the expression is not constant and is a function argument, d) the expression is not constant and is a return expression.
- The unary minus operator shall not be applied to an operand whose underlying type is unsigned.

- Before preprocessing, a null statement shall only occur on a line by itself. It may be followed by a comment, provided that the first character following the null statement is a white-space character.
- Removed superfluous division by zero protection: The code protection handling division by zero operations is generated by the Code Generator when needed and is not additionally contained in TargetLink's Fixed-Point Library.

Code Generator improvements

Code Generator improvements to comply with MISRA-C:

- TargetLink does not generate bit-wise XOR operations in look-up table code anymore.
- TargetLink no longer converts subtractions with unsigned integer results into additions using the compute-through-overflow (CTO) calculation method. Instead, the subtraction is generated into the production code, improving readability and leading to MISRA compliance.

≤ TargetLink 4.0	TargetLink 4.1
U8 = U8 + 255;	U8 = U8 - 1;

- Dereferences in logical operations are now written in parentheses.

≤ TargetLink 4.0	TargetLink 4.1
Sa1_OutPort1 = (Int16) (*In2 *In1);	Sa1_OutPort1 = (Int16) ((*In2) (*In1));

- There are code changes regarding *Assignments of Stateflow State IDs* and *Assignments to bitfields*. For details, refer to [Assignments to bitfields](#) on page 529.

Further improvements

Further improvements to comply with MISRA-C:

- By default, TargetLink now generates the **void** data type instead of the **Void** typedef.
- Also refer to [No Void typedef](#) on page 522.

Improved Code Efficiency

Dimension downgrade

TargetLink can now replace accesses to the same index range of a matrix variable by a scalar variable.

Additionally, elements of vector and matrix variables can now be replaced if they are based on the same non-loop-variable expression:

TargetLink < 4.1	TargetLink 4.1
vector2[e-d] = c[e-d]*7; g(&vector2[e-d]);	Aux_S16_a = c[e-d]*7; g(&Aux_S16_a);

This also works for vectors and matrices as well as index access ranges with relative offsets.

Related documentation

- [Basics on Optimizing Variables](#) ([TargetLink Customization and Optimization Guide](#))
- [Basics on Eliminating Temporary Variables](#) ([TargetLink Customization and Optimization Guide](#))

Copy propagation

TargetLink can now remove variables in more contexts.

≤ TargetLink 4.0	TargetLink 4.1
<pre>if (cond) { b = a + 1; } a = b;</pre>	<pre>if (cond) { a = a + 1; }</pre>
<pre>if (cond1) { if (cond2) { s = In1; } else { s = x; } } else { s = In2; } x = s;</pre>	<pre>if (cond1) { if (cond2) { x = In1; } } else { x = In2; }</pre>

Related documentation

- [ERASABLE](#) ([TargetLink Customization and Optimization Guide](#))

Moving code into conditionally executed branches

TargetLink can now move more code into conditionally executed branches. This is especially visible for vector or matrix code:

≤ TargetLink 4.0	TargetLink 4.1
<pre> if (Cond1[0]) { S1[0] = Input1[0]; } else { S1[0] = Input2; } ... if (Cond1[3]) { S1[3] = Input1[3]; } else { S1[3] = Input2; } /* Switch: CascSw/S2 CascSw/S2: Omitted comparison with constant. */ if (Cond2[0]) { Output[0] = (Int16) (-S1[0]); } else { Output[0] = 1; } ... if (Cond2[3]) { Output[3] = (Int16) (-S1[3]); } else { Output[3] = 4; } </pre>	<pre> /* Switch: CascSw/S2 [0] CascSw/S2: Omitted comparison with constant. */ if (Cond2[0]) { if (Cond1[0]) { Output[0] = (Int16) (-Input1[0]); } else { Output[0] = (Int16) (-Input2); } } else { /* # combined # TargetLink output: CascSw/Output */ Output[0] = 1; } ... /* Switch: CascSw/S2 [3] CascSw/S2: Omitted comparison with constant. */ if (Cond2[3]) { if (Cond1[3]) { Output[3] = (Int16) (-Input1[3]); } else { Output[3] = (Int16) (-Input2); } } else { Output[3] = 4; } </pre>

Related documentation

- [MOVABLE](#) ([TargetLink Customization and Optimization Guide](#))

Better optimization of Assignment blocks in iterated subsystems

The production code generated for Assignment blocks that reside in iterated subsystems and whose Omit dispensable initializations checkbox is cleared was improved:

≤ TargetLink 4.0	TargetLink 4.1
<pre> for (Sa3_For_Iterator_it = 0; Sa3_For_Iterator_it <= 9; Sa3_For_Iterator_it++) { if (Sa3_For_Iterator_it == 0) { for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { /* Assignment: ... - output initialization */ Ass_For[Aux_S32] = X_UD_For[Aux_S32]; } } /* Assignment: ... - output calculation * combined # Product: ... */ Ass_For[Sa3_For_Iterator_it] = Op(Sa3_For_Iterator_it); for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { /* Unit delay: ... */ X_UD_For[Aux_S32] = Ass_For[Aux_S32]; } } for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { /* TargetLink outport: ... */ OutFor[Aux_S32] = Ass_For[Aux_S32]; } </pre>	<pre> for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { /* Assignment: ... - output initialization */ Ass_For[Aux_S32] = X_UD_For[Aux_S32]; } for (Sa3_For_Iterator_it = 0; Sa3_For_Iterator_it <= 9; Sa3_For_Iterator_it++) { /* Assignment: ... - output calculation * combined # Product: Direct_Init/For/Product */ Ass_For[Sa3_For_Iterator_it] = Op(Sa3_For_Iterator_it); for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { /* Unit delay: ...For */ X_UD_For[Aux_S32] = Ass_For[Aux_S32]; } } </pre>

Which can be simplified as follows:

```

for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) {
    /* TargetLink outport: ... */
    OutFor[Aux_S32] = Ass_For[Aux_S32];
}

```

Optimization of Rte_IRead() pointer return variables

TargetLink no longer generates unnecessary return variables for Rte_IRead() and Rte_IWWriteRef():

≤ TargetLink 4.0	TargetLink 4.1
<pre> if (cond1) { p_DE = Rte_IRead_Run_DE(); ... /* Use p_DE */ } ... p_DE_a = Rte_IRead_Run_DE(); ... /* Use p_DE_a */ </pre>	<pre> p_DE = Rte_IRead_Run_DE(); if (cond1) { ... /* Use p_DE */ } /* Use p_DE */ </pre>

More algebraic simplifications for auxiliary variables used for code patterns

≤ TargetLink 4.0	TargetLink 4.1
<pre>aux = 42; aux += 0; aux -= 0; aux *= 1; aux /= 1;</pre>	<pre>aux = 42;</pre>

Component-Based Development

Improvements to Function Reuse

Function reuse for subsystems configured for incremental code generation and referenced models

With this TargetLink version, function reuse is possible not only for simple atomic subsystems but also for subsystems configured for incremental code generation and referenced models (with multiple instances).

Related documentation

- [Basics on Function Reuse](#) ([TargetLink Customization and Optimization Guide](#))
- [MULTIPLE_INSTANCES_REFMODEL](#) ([TargetLink Demo Models](#))

Variable propagation for function reuse

TargetLink now lets you reuse variables of predecessor and successor blocks of the reusable system definition without generating additional interface variables.

Related documentation

- [Basics on Reusing Variables of Preceding and Subsequent Blocks](#) ([TargetLink Customization and Optimization Guide](#))
- [FUNCTION_REUSE](#) ([TargetLink Demo Models](#))

AUTOSAR

Where to go from here

Information in this section

Supported AUTOSAR Releases.....	117
Support of NvData Communication.....	118
Data Transformation.....	118
Activation Reasons of Runnables.....	118
Port-Defined Argument Values.....	119
Miscellaneous AUTOSAR Features.....	119

Supported AUTOSAR Releases

Supported AUTOSAR Releases

The following AUTOSAR Releases are supported:

AUTOSAR Release	Revision
4.2	4.2.1 ¹⁾
4.1	4.1.3 4.1.2 4.1.1
4.0	4.0.3 4.0.2
3.2	3.2.3 3.2.2 3.2.1
3.1	3.1.5 3.1.4 3.1.2 3.1.0
3.0	3.0.7 3.0.6 3.0.4 3.0.2
2.1	2.1.4

¹⁾ New in TargetLink 4.1.

Support of NvData Communication

NvData communication

TargetLink supports implicit NvData communication as described by AUTOSAR. This includes the support of provide-require ports.

You can model access to the NVRAM via port blocks, data store memory blocks, and block parameters.

Via special modeling styles, TargetLink lets you reduce accesses to the NVRAM.

Related documentation

- [Modeling NvData Communication](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#))
- [AR_NVDATA_TRANSFORMER](#) ( [TargetLink Demo Models](#))

Data Transformation

Modeling and simulating error logic

TargetLink lets you use data transformation as described by AUTOSAR to model and simulate related error logic, e.g., for end-to-end protection for safety-critical applications or for Automotive Ethernet and SOME/IP.

Related documentation

- [Basics on Data Transformation](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#))
- [How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#))
- [tITransformerError](#) ( [TargetLink API Reference](#))
- [AR_NVDATA_TRANSFORMER](#) ( [TargetLink Demo Models](#))

Activation Reasons of Runnables

Activation reasons

TargetLink lets you use activation reasons for your runnables as described by AUTOSAR:

TargetLink lets you specify DD ActivationReason objects in a Runnable object's subtree.

To model activation reasons, you use TargetLink InPort blocks.

Related documentation

- [Basics on Activation Reasons](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#))

- [How To Model a Runnable's Activation Reasons \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [AR_POSCONTROL \(TargetLink Demo Models\)](#)

Port-Defined Argument Values

Port-defined argument values TargetLink supports port-defined argument values as described by AUTOSAR.

In the model, you reference a DD PortDefinedArgument object at the TargetLink InPort block that represents the port-defined argument value.

In production code, TargetLink generates port-defined argument values as formal arguments within the runnable function's signature.

Related documentation

- [Basics on Port-Defined Argument Values \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [How to Model Port-Defined Argument Values \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Miscellaneous AUTOSAR Features

Support of Rte_IWriteRef for NvData communication TargetLink supports the Rte_IWriteRef API function for NvData communication.

Transformation ComSpecs TargetLink can import and export communication specifications for data transformations and TRANSFORMER_HARD_ERROR_EVENT.

ImplementationPolicy of parameter prototypes TargetLink can import and export implementation policies for parameter data prototypes.
The data is stored in the DD ImplementationPolicy property of parameter prototypes.

StepSize property TargetLink can import and export a step size specified for measurement and calibration tools.
The data is stored in the StepSize property of primitive application data types or data prototypes.

ConstrLevel property	TargetLink can import and export constraint levels specified for measurement and calibration tools The data is stored in the ConstrLevel property of primitive application data types or data prototypes. For implementation data types, it is stored in the Constraints subtree.
Scaling for LINEAR category	TargetLink can import and export default values of scalings whose category is set to LINEAR. The data is stored in the DefaultValue property of DD Scaling objects.
Data status of DataReceiverComSpec objects	TargetLink supports the HandleDataStatus property at DD DataReceiverComSpec objects.

Target Simulation (PIL)

Where to go from here	Information in this section
	Changes in the Target Simulation Modules..... 120 Folder for TSM Extensions..... 122

Changes in the Target Simulation Modules

New and discontinued compiler versions	The following table shows the compiler versions that are now supported by TargetLink 4.1, refer to the New and No Changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.
--	--

Target	Compiler	New	No Changes	Discontinued
ARM CortexM3	Keil	5.1	—	—
C16x	TASKING	—	8.6	8.7
HCS12	Cosmic	—	—	4.8
	Metrowerk	—	—	5.1
M32R	Gaio	—	—	11, 9
	Renesas	—	—	5.1

Target	Compiler	New	No Changes	Discontinued
MC56F83	Metrowerk	—	—	8.3
MPC55xx	Diab	—	—	5.9
	GreenHill	—	—	2013
	GNU	—	—	4.1
	Metrowerk	—	—	2.8
MPC55xxVLE	Diab	—	—	5.9
	GreenHill	—	—	2013
	Metrowerk	—	—	2.8
MPC57xxVLE	Diab	5.9	—	—
	GreenHill	2014	—	—
MPC560xVLE	Diab	—	5.9	—
	GreenHill	2014	2012	2013
RH850	GreenHill	2014	—	—
S12X ¹⁾	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3	9.4
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	5.0	3.2	4.3
	GNU	—	—	3.4
TriCore2xx	TASKING	5.0	—	—
	GNU	4.6	—	—
V850	GreenHill	2014	—	2013
	NEC	—	—	3.40
XC22xx	TASKING	—	3.0	—

¹⁾ Freescale S12XEVB/S12XEVB_USB is replaced by the new Freescale EVB9S12XEP100.

For more details on the evaluation boards supported by TargetLink, refer to
 [Evaluation Board Reference](#).

Note

For further PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Discontinued boards

No longer supported, no longer distributed The following boards are no longer supported by TargetLink and no longer distributed by dSPACE:

- Freescale 56F83xx
- Freescale HCS12

- Freescale PowerPC MPC5500
- Freescale PowerPC MPC5500VLE
- NEC V850ES
- Renesas M32R

Note

If you want to use the unsupported evaluation boards with TargetLink 4.1, contact dSPACE Support (www.dspace.com/go/supportrequest).

Still supported, no longer distributed The following boards are still supported by TargetLink but no longer distributed by dSPACE:

- Freescale S12XEVB/S12XEVB_USB is replaced by the new Freescale EVB9S12XEP100

Folder for TSM Extensions

Specifying the folder via the API

You can now specify the folder for TSM extensions not only via the Preferences Editor but also via the API, for example:

```
TlTsmManager.exe -SetTsmExtensionFolder -Folder:C:\exp
```

Related documentation

- [How to Clone Target/Compiler Combinations to Outside the TargetLink Installation](#) ( [TargetLink Customization and Optimization Guide](#)) (the Preconditions in particular)

Data Dictionary and Data Management

Where to go from here

Information in this section

Improvements to the Data Dictionary.....	123
New DD MATLAB API Functions.....	124
Referencing DD CodegenOptions Objects at TargetLink Main Dialog Block.....	124

Improvements to the Data Dictionary

Predefined filter rule sets for different DD views

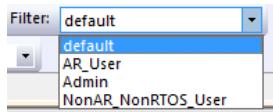
TargetLink comes with new predefined filter rule sets that can hide specific objects and properties in the DD tree. This makes it easier to view the relevant data. The filter rule sets are designed for typical user groups.

The following predefined filter rule sets are available:

- Admin - A filter rule set for administrators.
- AR_User - A filter rule set for AUTOSAR users.
- NonAR_NonRTOS_User - A filter rule set for users that use neither AUTOSAR nor RTOS.

For further information on the filter rule sets, refer to [DD_FILTER](#) ([TargetLink Demo Models](#)).

You can change the filter rule set in the Filter list of the Data Dictionary Manager's toolbar.



Related documentation

- [Basics on Filter Rule Sets for the Data Model](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))
- [How to Create Filter Rule Sets via DD Files](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

Filter views for the Object Comparison Navigator

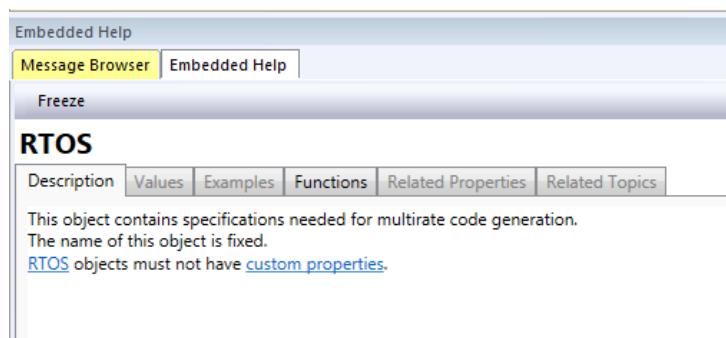
You can use the Filter list in the Object Comparison Navigator to hide specific objects and properties of no interest. This makes it easier to compare a high number of DD objects. You can either select one of the predefined filter rule sets or build your own.

Related documentation

- [Basics on Filter Rule Sets for the Data Model](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))
- [Comparing and Merging Data Dictionaries](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

Message alerts

If the Message Browser pane is opened but hidden because of an active Embedded Help pane, the tab of the Message Browser indicates that there are new messages by changing color.



- A new Info message - blue tab
- A new Warning message - yellow tab
- A new Error message - red tab

Related documentation

- [Overview of the User Interface](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

New DD MATLAB API Functions

New DD API functions

TargetLink provides several new DD MATLAB API functions that are listed below. For more details, refer to the [TargetLink Data Dictionary Reference](#).

GetAutosarVersion

```
[version, errorCode] = dsdd('GetAutosarVersion',[,<DD_identifier>]);
```

retrieves the Autosar version as specified with the /Pool/Autosar/Config.AutosarVersion property of the specified DD.

GetRenameBaseType

```
[version, errorCode] = dsdd('GetAutosarVersion',[,<DD_identifier>]);
```

retrieves the Typedef object which specifies the rename rule for a base data type.

Referencing DD CodegenOptions Objects at TargetLink Main Dialog Block

Referencing DD CodegenOptions options

For centralized handling and easier specification of consistent options, TargetLink 4.1 lets you directly reference DD CodegenOptions objects at the TargetLink Main Dialog block.

Related documentation [Basics on Configuring the Code Generator for Production Code Generation](#) ([TargetLink Customization and Optimization Guide](#))

Code Generator Options

New Code Generator Options

Overview of new Code Generator options

The following new Code Generator options are available with TargetLink 4.1.

- [AvoidNestedVariablePropagationPointerAccess](#)
Generates additional pointers in reuse structures if these reuse structures access variables for which variable propagation is specified. For details, refer to [AvoidNestedVariablePropagationPointerAccess](#) ([TargetLink Model Element Reference](#)).
- [ReportFailedFunctionReuseVariablePropagation](#)
Enables an optional report indicating problems that occurred in function reused systems during variable propagation. For details, refer to [ReportFailedFunctionReuseVariablePropagation](#) ([TargetLink Model Element Reference](#)).
- [ReplaceUnrolledVectorsAndMatricesByScalar](#)
Controls the replacement of unrolled vector and matrix accesses by scalars. For details, refer to [ReplaceUnrolledVectorsAndMatricesByScalars](#) ([TargetLink Model Element Reference](#)).

For reference information on all Code Generator options, refer to [Alphabetical List of Code Generator Options](#) ([TargetLink Model Element Reference](#)).

Migration aspects of Code Generator options

Migration aspects include:

- Removed Code Generator options
- Changed Code Generator options
- Recommended compatibility settings
- Basics on changed defaults

For details, refer to [Migration Aspects Regarding Code Generator Options](#) on page 339.

Tool Chain Integration

Where to go from here

Information in this section

Exporting Functional Mock-up Units.....	126
Requirement Information in the Data Dictionary.....	126

Exporting Functional Mock-up Units

Functional Mock-up Units

TargetLink lets you generate Functional Mock-up Units (FMUs) for TargetLink subsystems. These FMUs are based on the FMI 2.0 standard in order to execute TargetLink code in FMI-compliant simulation environments.

These include VEOS, SCALEXIO and a large range of third-party FMI-compliant tools (<https://www.fmi-standard.org/tools>).

Related documentation

- Definition of the FMI Standard and FMUs ([TargetLink Interoperation and Exchange Guide](#))
- Basics on Exporting FMUs from TargetLink ([TargetLink Interoperation and Exchange Guide](#))

Requirement Information in the Data Dictionary

Storing requirement information as DD objects

TargetLink lets you store requirement information in the Data Dictionary as RequirementInfo objects. These objects act as a proxy to your requirements management system.

In the model, you can reference these objects at TargetLink blocks and from Stateflow objects. This instructs TargetLink to add requirement information as comments to the generated production code and the generated documentation.

You can programmatically handle the block data for requirement information via the `t1RequirementInfo()` API function, which is used instead of `t1_set()` or `t1_get()` for this data.

Related documentation

- Basics on Requirement Information in the Generated Code ([TargetLink Interoperation and Exchange Guide](#))

- [Basics on Using DD-Based Requirement Information \(TargetLink Interoperation and Exchange Guide\)](#)
- [tlRequirementInfo \(TargetLink API Reference\)](#)

Other

Where to go from here

Information in this section

General Enhancements and Changes.....	127
TargetLink Demos.....	129

General Enhancements and Changes

TargetLink context menu

The context menu of TargetLink blocks provides two new options:

- [Create Reference \(Incr. Subsystem to Ref. Model\)](#)
- [Disable Reference \(Ref. Model to Incr. Subsystem\)](#)

Before, these options were available only in the dialog of the TargetLink Main block. You can use them to replace a subsystem configured for incremental code generation with a referenced model, and vice versa.

Related documentation

- [Common TargetLink Context Menu Options \(TargetLink Model Element Reference\)](#)

Improved simulation performance

TargetLink's simulation performance has been improved. The following types of models are now simulated with better performance in MIL simulation mode:

- Models that contain many scaling-invariant subsystems
- Models that use many workspace variables

Note

To maximize simulation performance for TargetLink models, it is best to use one of the following methods to start the simulation:

- Via the `tl_sim` API function instead of Simlunk's own `sim()` function
- Via the TargetLink Main Dialog block
- Via a TargetLink plot dialog

Related documentation

- None
-

Improved synchronization of function system signatures

The following improvements apply to the specification and synchronization of function system signatures:

- In addition to the DD Function object, now you can link a DD Signature object in a TargetLink Function block. DD Signature objects contain interface specifications (port data).
- You can perform consistency checks between the interface (port data) specified in the DD Signature object and the modeled interface of the Function system (Check ports).
- You can update/synchronize the interfaces in the model with the specifications of the DD Signature object (DD to Model).
- The SyncSystemSignature object in the /Pool/ModelDesign/Config/ object tree no longer contains string properties. Instead, it contains typed data types like Boolean or Enum. This makes it easier to configure them in the Data Dictionary Manager.
- If you do not have any DD Signature object specifications but want to transfer your modeled interface data to the Data Dictionary, you can also synchronize the port data to the Data Dictionary (Model to DD).
- An HTML report is generated whenever you check or synchronize ports. The report contains detailed information about differences in the specification of the DD Signature object and the interface of the Function system.

Related documentation

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

Setting MEX and SIL compilers independently

You can now set a SIL compiler for the production code DLL independently of the MEX compiler for the simulation S-function. This lets you use the free MSVC Express Edition for compiling the production code and SIL debugging.

Related documentation

- [How to Set or Change MEX and SIL Compilers](#) ([TargetLink Preparation and Simulation Guide](#))
 - [How to Debug in SIL Simulation Mode](#) ([TargetLink Preparation and Simulation Guide](#))
 - [tlProductionCodeSILCompiler](#) ([TargetLink API Reference](#))
-

Related topics**References**

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

TargetLink Demos

New demos

The following new demos come with TargetLink 4.1.

Ar_nvdata_transformer This new demo shows two features:

- Read and Write access to nonvolatile RAM via NVData Interfaces
- Use of AUTOSAR transformers: e.g., to model end-to-end communication protection for safety-critical applications

DD_filter The demo shows how to create XML filter rule sets based on DD files with the required specification. The demo contains the

`tl_example_CreateDDFilterBasedOnDDFile` script and three generic DD files with the specification for typical filter use cases. The script generates an XML filter rule set.

Note

This demo does not contain any model.

Related Documentation

- [DD_FILTER](#) (TargetLink Demo Models)
- [Basics on Filter Rule Sets for the Data Model](#) (TargetLink Data Dictionary Basic Concepts Guide)

DD_ML_API This demo shows two examples of how to use the DD MATLAB API:

- The `simple.m` script creates a DD variable object and a user-defined generic DD object via dsdd commands.
- The `findCalVariables.m` script demonstrates how to find all variable objects with STATIC_CAL class in a DD file.

Related Documentation

- [DD_ML_API](#) (TargetLink Demo Models)

DD_ML_ImportExport This demo contains various M scripts that show how to import objects to the Data Dictionary from XLS and XML files.

Related Documentation

- [DD_ML_IMPORTEXPORT](#) (TargetLink Demo Models)

Function_reuse This demo shows the function reuse feature that is applied to subsystems with instance-specific initial parameter values by defining mask parameters.

Related Documentation

- [FUNCTION_REUSE](#) (TargetLink Demo Models)
- [Basics on Function Reuse](#) (TargetLink Customization and Optimization Guide)

Multiple_instances_refmodel This demo shows the function reuse feature that is applied to referenced models with instance-specific initial parameter values by defining model arguments.

Related Documentation

- [MULTIPLE_INSTANCES_REFMODEL](#) (TargetLink Demo Models)
- [Basics on Function Reuse](#) (TargetLink Customization and Optimization Guide)

Table1d_usr_local This demo shows how to replace TargetLink look-up table code that uses a local search algorithm with nonscalar inputs by custom look-up functions.

Related Documentation

- [TABLE1D_USR_LOCAL](#) (TargetLink Demo Models)
- [Basics on Using Custom Look-Up Functions](#) (TargetLink Preparation and Simulation Guide)

Variable_vector_width This demo model shows how to work with vectorized variables that have width variants. Using preprocessor macros for vector widths, the same model and generated production code can be used for all widths. The code of complete subsystems can become width-varying at code compile time.

Related Documentation

- [VARIABLE_VECTOR_WIDTH](#) (TargetLink Demo Models)
- [Introduction to Variable Vector Widths](#) (TargetLink Customization and Optimization Guide)

Improved demos

The following demos contain new feature demonstrations:

Ar_poscontrol This demo simulates activation reasons with a Stateflow chart triggering various RTE events. Signals are automatically adjusted to match the values specified in the DD.

Related Documentation

- [AR_POSCONTROL](#) (TargetLink Demo Models)
- [Basics on Activation Reasons](#) (TargetLink Classic AUTOSAR Modeling Guide)

Poscontrol The demo now shows the connection of TargetLink function blocks with DD Function and DD Signature objects in the Data Dictionary to ensure interface consistency between functions and the modeled subsystems.

Related Documentation

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

API Functions and Hook Functions

Where to go from here

Information in this section

New API Functions.....	131
New Hook Functions.....	132

New API Functions

Exporting FMUs

The `t1_generate_fmu` API function lets you export functional mock-up units (FMUs) to use in FMI-compliant tools.

Related documentation

- Definition of the FMI Standard and FMUs ([TargetLink Interoperation and Exchange Guide](#))
- Basics on Exporting FMUs from TargetLink ([TargetLink Interoperation and Exchange Guide](#))
- How to Generate an FMU to use in an FMI-Compliant Tool ([TargetLink Interoperation and Exchange Guide](#))
- TargetLink FMU Manager ([TargetLink Tool and Utility Reference](#))

Switching SIL compilers

The `t1ProductionCodeSILCompiler` API function lets you set or change SIL compilers.

Related documentation

- How to Set or Change MEX and SIL Compilers ([TargetLink Preparation and Simulation Guide](#))

Setting requirement information at blocks

The `t1RequirementInfo` API function lets you manage DD-based requirement information at TargetLink blocks and Stateflow objects. `t1RequirementInfo` is used instead of `t1_set()` or `t1_get()` for this data.

Related documentation

- Basics on Using DD-Based Requirement Information ([TargetLink Interoperation and Exchange Guide](#))

Preparing the simulation of transformation error logic

The `t1TransformerError` API function helps you prepare AUTOSAR models for simulation that contain transformation error logic.

Related documentation

- [Basics on Data Transformation](#) (TargetLink Classic AUTOSAR Modeling Guide)
- [How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication](#) (TargetLink Classic AUTOSAR Modeling Guide)

Related topics**References**

- [tl_generate_fmu](#) (TargetLink API Reference)
- [tlProductionCodeSILCompiler](#) (TargetLink API Reference)
- [tlRequirementInfo](#) (TargetLink API Reference)
- [tlTransformerError](#) (TargetLink API Reference)

New Hook Functions

Generating and synchronizing system signatures

TargetLink provides the following new hook functions to customize subsystem generation from the DD:

- [**tl_pre_add_ddsignatureport_hook**](#) (TargetLink File Reference) This hook function is called before a new DD SignaturePort object is added to the specified DD Signature object.
- [**tl_post_add_ddsignatureport_hook**](#) (TargetLink File Reference) This hook function is called after a new DD SignaturePort object is added to the specified DD Signature object.
- [**tl_pre_sync_systemsignatureport_hook**](#) (TargetLink File Reference) This hook function is called before an existing Port block is synchronized with the corresponding DD Signature object.
- [**tl_post_sync_systemsignatureport_hook**](#) (TargetLink File Reference) This hook function is called after an existing Port block is synchronized with the corresponding DD Signature object.

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

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

Initializing buses via DD Variable objects

TargetLink provides the following customization file to customize the initialization of buses via Simulink IC structures:

- [**tlGetBusStructMapping**](#) This customization file gets the bus struct mappings (DD Variable objects to Simulink.Bus objects).

This customization file lets you map DD-based struct variables to Simulink.Bus objects.

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

New Features of TargetLink 4.0

Where to go from here

Information in this section

Modeling in Simulink or Stateflow.....	133
Code Generation Core Functionality.....	139
Data Dictionary and Data Management.....	141
AUTOSAR.....	146
Test Support.....	149
Code Generator Options.....	151
Tool Chain Integration.....	152
Other.....	153
API Commands.....	155

Modeling in Simulink or Stateflow

Where to go from here

Information in this section

Support of Matrix Signals.....	134
Newly Supported Simulink Blocks.....	134
Improved Bus Support.....	134
Dynamic Look-Up Tables.....	135
Improvements to TargetLink's Simulation Frame.....	135
Improvements to Scaling-Invariant Systems.....	136
Additionally Supported Block Properties.....	136
Central Specification of Function Subsystem Signatures.....	137

Support of Matrix Signals

2-D matrix signals

TargetLink supports code generation for 2-D signals and parameters (see also: [matrix signal](#)) and lets you perform simulations in MIL/SIL/PIL simulation mode.

Note

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

Related documentation

- [Introduction to Matrix Signals](#) ([TargetLink Preparation and Simulation Guide](#))
- [Examples of Working With Matrix Signals](#) ([TargetLink Preparation and Simulation Guide](#))
- [Block-Specific Limitations](#) ([TargetLink Limitation Reference](#))

Newly Supported Simulink Blocks

Overview of newly supported Simulink blocks

TargetLink now supports the following Simulink blocks:

- Matrix Concatenate
- Permute Dimensions
- Reshape

Related documentation

- [Supported Simulink Blocks](#) ([TargetLink Model Element Reference](#))
- [Working With Matrix Signals](#) ([TargetLink Preparation and Simulation Guide](#))

Improved Bus Support

Mapping Entire Buses to Predefined Structs

This TargetLink version lets you create a predefined struct type in the TargetLink Data Dictionary (DD), and assign a complete bus to this DD struct type in TargetLink's BusImport and BusOutport blocks. This is especially helpful if the bus consists of many bus elements, and/or the type or variable is used several times in a model or across models.

In addition, you can reference DD struct types or struct variables in bus-capable blocks, for example Switch, Multiport Switch, Merge, and Unit Delay blocks.

Related documentation

- [Basics on the Representation of Buses in the Production Code \(TargetLink Preparation and Simulation Guide\)](#)
- [Mapping Entire Buses to Explicit Structs in the Code \(TargetLink Customization and Optimization Guide\)](#)

Dynamic Look-Up Tables

Changing table data during simulation

For the three TargetLink look-up table blocks listed below, you can specify the table data not only in the block dialog but you can also feed the table data (1-D or 2-D) into the block through the block's optional *table data input port*. The latter allows you to change the table data during simulation and run time of the generated code.

- Direct Look-Up Table (n-D)
- Prelookup
- Interpolation Using Prelookup

Related documentation

- [Basics on Look-up Tables \(TargetLink Preparation and Simulation Guide\)](#)
- [How to Prepare Dynamic Look-Up Table Specification \(TargetLink Preparation and Simulation Guide\)](#)

Improvements to TargetLink's Simulation Frame

Disabling the MIL subsystem via the Commented block parameter

When switching to SIL or PIL simulation mode, TargetLink now disables the MIL subsystem by setting the Commented block parameter to **on**.

This is beneficial in the following ways:

- Improved speed of model initialization in SIL or PIL simulation mode
- Improved speed when switching to SIL or PIL simulation mode

Compatibility consideration

- Simulink does not perform consistency checks for commented blocks.
- By default, Simulink's `find_system` API function does not include commented blocks in its search.
- Simulink does not allow using OpenFcn callbacks that directly or indirectly change the Commented block property. For details, refer to [Various Migration Aspects](#) on page 355.

Solution Two possible solutions exist:

- Adapt your user scripts so that the simulation mode is switched to MIL first.

- Adapt your user scripts to using TargetLink API functions:
 - `tl_get_blocks`
 - `tl_get_sfobjects`
 - `tl_find`

The new Activate MIL entry in the TargetLink menu in Simulink Editor lets you easily switch into MIL simulation mode.

Related topics

References

- [tl_find](#) (TargetLink API Reference)
[tl_get_blocks](#) (TargetLink API Reference)
[tl_get_sfobjects](#) (TargetLink API Reference)

Improvements to Scaling-Invariant Systems

Improved scaling inheritance

For scaling-invariant subsystems, you can optionally specify the import(s) that influence an outport's scaling. You can specify the port mapping in the scaling propagation function. The Code Generator takes this mapping into account during code generation if the (new) `UtilizeExplicitDependenciesForScalingInvariantSystems` Code Generator option is enabled. The feature is beneficial for a better scaling propagation in case of loops including scaling-invariant systems. Dependencies of output scalings with regard to individual inputs are now taken into account, which helps to resolve loops during scaling-propagation.

Related documentation

- [Details on the Scaling Propagation Function](#) (TargetLink Customization and Optimization Guide)

Access to System Handle

Scaling propagation functions now also provide access to the handle of the instance of the scaling-invariant system. This is particularly helpful if the scaling depends not only on inputs but also on instance-specific block data, e.g. specified in mask variables.

Additionally Supported Block Properties

Additional options for the Assignment block

The TargetLink Assignment block now supports the Starting index (dialog) and Starting index (port) options.

Related documentation

- Assignment Block

Additional options for the Selector block

TargetLink now supports the Starting index (dialog) and Starting index (port) options of the Simulink Selector block.

Improved plotting for the Sink block

The TargetLink Sink block now supports the specification of individual plot channels rather than a general switch for all plot channels at once.

Related documentation

- [Sink Block](#) ( [TargetLink Model Element Reference](#))

Date property for Stateflow data

TargetLink now provides a Date property for Stateflow data just as for TargetLink blocks. The property helps you to identify the latest changes, e.g., when inspecting Stateflow objects by using the TargetLink Property Manager.

Related topics**References**

[Assignment Block](#) ( [TargetLink Model Element Reference](#))

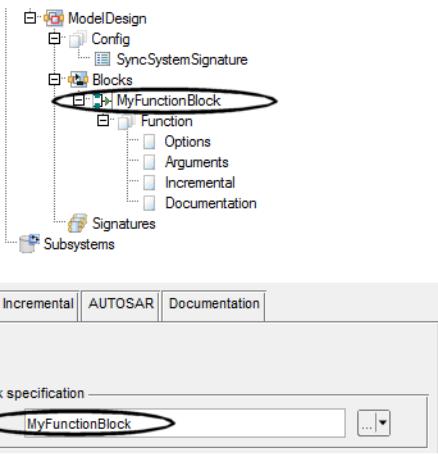
Central Specification of Function Subsystem Signatures

Overview

You can now centrally specify the signature of a function subsystem in the Data Dictionary. Additionally, you can generate function subsystems from these central specifications:

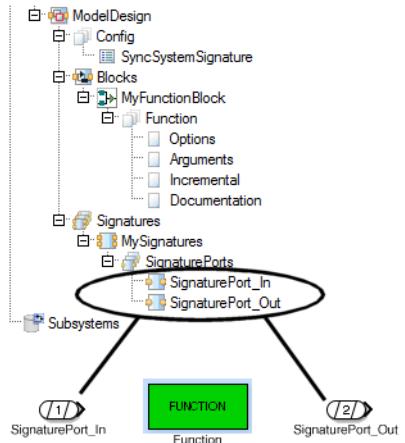
DD Function Block object specifying Function Block data

You can now specify Function Block data from within the TargetLink Data Dictionary by using a Function Block object. You can then reference this object in a TargetLink Function Block as shown below.



DD Signature object specifying a subsystem's signature

You can also specify a subsystem's signature by using a Signature object. To do so, use the Create Signature context command and specify the ports of the subsystem.



Finally, you can create the subsystem and its signature in the model from within the Data Dictionary Manager by using the Synchronize System Signature context command. You can also use the `t1SyncSystemSignature` API command.

For further information, refer to

- [Basics on Centrally Specifying Function System Signatures](#) ([TargetLink Customization and Optimization Guide](#))
- [How to Specify Function Block Data](#) ([TargetLink Customization and Optimization Guide](#))
- [How to Specify System Signature Data](#) ([TargetLink Customization and Optimization Guide](#))

Code Generation Core Functionality

Where to go from here

Information in this section

MISRA-C Compliance.....	139
Improved Code Efficiency.....	140

MISRA-C Compliance

Improvements to comply with MISRA-C

Several improvements were made to TargetLink's Fixed-Point Library to comply with MISRA-C. The improvements include:

- The redundant initializations for the accumulate register were removed from FIR filter macros.
- At many locations in the code, the numeric constants for the data type limits were replaced by global macros (e.g., INT32MIN).
- Constants functioning as call parameters were given suffixes or were cast to the expected type if necessary.
- Numeric constants functioning as initial values were given suffixes if necessary.

Further improvements to comply with MISRA-C:

- For macro access functions, the following placeholders for macro arguments are now enclosed by brackets:
 - _var
 - _value
- For Logical Operator blocks, TargetLink now generates the logical expression ! = instead of the arithmetic expression ^ to consistently separate logical and arithmetic expressions.
- For relational and logical operations, TargetLink now allows you to avoid assigning the result directly to a non-Boolean variable. This behavior can be controlled via the new AssignmentOfConditions Code Generator option.
By default, Relational Operator and Logical Operator blocks now lead directly to Output = <Condition>; assignments in production code (independent of optimizations), while

```
if (<Condition>) {
    Output = 1; /* or 0 */
} else {
    Output = 0; /* or 1, respectively */
}
```

is only optimized to an assignment statement if Output is of Boolean type (and Optimization is enabled).

- For RDI macro definitions, TargetLink now places brackets around the initial values that receive a cast.

Improved Code Efficiency

Loops for nonscalar signals

TargetLink's loop code generation was improved. The improvements include:

- More block code pattern that lead to element-wise vector assignments or computations in certain configurations now perform these assignments and computations in loops.
This applies especially to **MinMax**, **Product**, **Sum**, and **Custom Code** blocks.
- Generating **for** loops for matrix code, including Stateflow matrix code
- Better loop merging

Elimination of unused definitions

TargetLink can now remove unused previous definitions from conditional control flows. This applies primarily to Stateflow outputs driving a **Merge** block.

Optimization of vectors and vector slices

TargetLink's optimization of vectors and vector slices whose dimensions are smaller than the **LoopUnrollThreshold** has changed: they are now initially optimized as if their dimensions were equal to or larger than the **LoopUnrollThreshold**. This allows for vector-specific optimizations in addition to element-wise optimizations.

Implicitly created auxiliary variables

TargetLink can now create implicit auxiliary variables of the vector kind, if this leads to more advantageous block code patterns for subsequent blocks.

Constant indices folding

TargetLink now supports constant folding in indices generated from **Assignment** and **Selector** blocks. This improves initial code and does not depend on code optimization.

- TargetLink 3.5:
`Sa2_Assignment[1 - 1] = ...`
- TargetLink 4.0:
`Sa2_Assignment[0] = ...`

Substitution of vector variables

TargetLink now more frequently substitutes vector variables which cannot be eliminated with scalar variables if possible by the code's semantics. This reduces RAM/stack consumption of the generated code:

≤ TargetLink 3.5	TargetLink 4.0
<pre>Int16 vec[...]; loop() { if (...) { vec[i] = ... } else { vec[i] = ... } ... = vec[i] ... = vec[i] }</pre>	<pre>Int16 scalar; loop() { if (...) { scalar = ... } else { scalar = ... } ... = scalar ... = scalar }</pre>
<pre>Int16 vec[...]; loop() { fnc(&vec[i]); ... = vec[i]; }</pre>	<pre>Int16 scalar; loop() { fnc(&scalar); ... = scalar; }</pre>

Data Dictionary and Data Management

Where to go from here

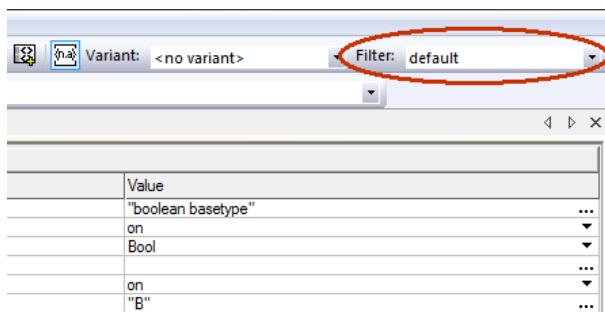
Information in this section

- | | |
|--|-----|
| Improvements to the Data Dictionary..... | 141 |
| New DD MATLAB API Commands..... | 143 |

Improvements to the Data Dictionary

Hiding specific objects and properties using filter rule sets

You can now hide specific DD objects and properties in the DD Manager using XML-based filter rule sets. A filter rule specifies whether an object or property defined in the Data Model is visible or invisible. Filter rule sets allow you to customize views for different team members. You can select a filter rule set in the Filter list of the TargetLink Data Dictionary Manager or via the MATLAB API.

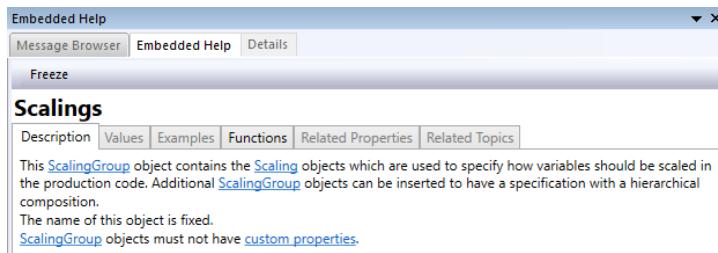


For further information on how to generate filter rule sets, refer to [Basics on Filter Rule Sets for the Data Model](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)) and [How to Create Filter Rule Sets via DD Files](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Examples of filter rule sets can be downloaded from the TargetLink Product Support Center: <http://www.dspace.com/tlpsc>.

Embedded help for objects and properties

The TargetLink Data Dictionary Manager now provides the Embedded Help pane for DD objects and properties. It provides detailed descriptions on selected objects or properties. By default, the pane is activated. From the Help menu, click Show Embedded Help to deactivate or reactivate it.



For further information, refer to [How to Get Help on DD Objects and Properties](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)) and [Show Embedded Help](#) ([TargetLink Data Dictionary Manager Reference](#)).

Improved merge function for objects and properties

A further option was added to the merge functions of the Data Dictionary Manager. The new Merge and Replace context command now supports replacing objects and properties. You can

- **Merge <left/right> without overwrite:** The DD objects are merged without overwriting child objects or property values.
- **Merge <left/right>, and overwrite properties:** The DD objects are merged without overwriting DD child objects. Property values are overwritten.
- **Merge <left/right>, and overwrite objects:** The DD objects are merged. Child objects and their property values are overwritten. Values and child objects existing in both DD objects are overwritten.
- **New option: Replace <left/right>:** The DD objects are replaced. The source objects are copied to the target object, replacing the target object.

This context command is also available for the DD Comparison Pane.

For further information, refer to [How to Merge or Replace DD Objects](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)) and [How to Merge or Replace a DD Object Using the DD Two-Way Comparison](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Extended Recent Files list

The Recent Files list now also shows DD project files opened from within a model or via MATLAB API. Previously, only DD project files opened in the Data Dictionary Manager were shown.

Complete message texts

The messages in the Data Dictionary Manager's Message Browser and in the Custom Output View are now displayed completely. Previously, only the tooltip of the message contained the complete text.

More robust XML import

The XML import of the Data Dictionary is now more robust to invalid XML files. This is helpful to make sure that XML files exported from previous versions of the Data Dictionary can be imported into the Data Dictionary of TargetLink 4.0.

Related topics**Basics**

[Basics on Filter Rule Sets for the Data Model](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

HowTos

[How to Create Filter Rule Sets via DD Files](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))
[How to Merge or Replace a DD Object Using the DD Two-Way Comparison](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))
[How to Merge or Replace DD Objects](#) ([TargetLink Data Dictionary Basic Concepts Guide](#))

New DD MATLAB API Commands

Overview of new DD MATLAB API functions

TargetLink provides several new DD MATLAB API functions that are listed below. For detailed information, refer to the [TargetLink Data Dictionary Reference](#).

CountItems

```
[numOfObjects,numOfProperties] = dsdd('CountItems',<objectIdentifier>);
```

Counts the number of objects and properties in a subtree.

CreateFilterRuleSet

```
bSuccess = dsdd('CreateFilterRuleSet',<filterRuleSet>);
```

Creates a filter rule set.

DeleteFilterRuleSet

```
bSuccess = dsdd('DeleteFilterRuleSet',<filterRuleSet>);
```

Deletes a filter rule set.

DumpDataModelPaths

```
bSuccess = dsdd('DumpDataModelPaths',<file>);
```

Writes all data model paths to a file.

GetCurrentFilterRuleSet

```
filterRuleSet = dsdd('GetCurrentFilterRuleSet');
```

Gets a current filter rule set.

GetDataModelPath

```
dataModelPath = dsdd('GetDataModelPath',<objectIdentifier>[,<propertyName>]);
```

Gets the data model path of the object or property.

GetDataModelPaths

```
dataModelPaths = dsdd('GetDataModelPath',<objectKind>[]);
```

Gets all data model paths for an object kind.

GetDataModelTag

```
dataModelTag = dsdd('GetDataModelTag',<DataModelPath>);
```

Gets the data model tag.

GetDefaultFilterRule

```
[bVisible,bValidPath] = dsdd('GetDefaultFilterRule',<DataModelPath>);
```

Gets the default filter rule of a data model item.

GetFilterRule

```
[bVisible,bValidPath] = dsdd('GetFilterRule',<DataModelPath>);
```

Gets the filter rule of a data model item.

GetFilterRuleChecksum

```
checkSum = dsdd('GetFilterRuleChecksum');
```

Gets the data model filter rule checksum.

GetFilterRuleSets

```
filterRuleSets = dsdd('GetFilterRuleSets');
```

Gets the list of filter rule sets.

GetNumOfFilterRuleSets

```
numOfFilterRuleSets = dsdd('GetNumOfFilterRuleSets');
```

Gets the number of filter rule sets.

GetPropertyTable

```
propertyTable = dsdd('GetPropertyTable',<objectIdentifier>);
```

Gets a table with property names and property values.

GetUnsetPropertyNames

```
propertyNames = dsdd('GetUnsetPropertyNames',<objectIdentifier>);
```

Returns the names of properties which are unset.

IsCustomProperty

```
bIsCustomProperty = dsdd('IsCustomProperty',<objectIdentifier>,<propertyName>);
```

Checks if a property is a custom property.

IsVisible

```
[bVisible] = dsdd('IsVisible',<objectIdentifier>[,<propertyName>]);
```

Checks if object or property is visible according to the current filter rule set.

ReadFilterRuleSet

```
bSuccess = dsdd('ReadFilterRuleSet',attributeName1,attributeValue1,...);
```

Reads a filter rule set XML file.

ReloadFilterRuleSets

```
bSuccess = dsdd('ReloadFilterRuleSets');
```

Re-reads filter rule set XML files.

RemoveVariants

```
errorCode = dsdd('RemoveVariants',<objectIdentifier>[,<propertyName>]);
```

Removes variants with ID != 0.

Replace

```
[hDDObject,errorCode] = dsdd('Replace',[,attributeName1,attributeValue1,...]);
```

Replaces an object by the copy of another object.

ResetFilterRuleSet

```
bSuccess = dsdd('ResetFilterRuleSet',<filterRuleSet>);
```

Resets a filter rule set.

SetCurrentFilterRuleSet

```
bSuccess = dsdd('SetCurrentFilterRuleSet',<filterRuleSet>);
```

Sets the current filter rule set.

SetFilterRule

```
bValidPath = dsdd('SetFilterRule',<DataModelPath>,<bVisible>);
```

Sets a filter rule of a data model item.

SetFilterRuleByDataModelTag

```
nRulesSet = dsdd('SetFilterRuleByDataModelTag',<DataModelTag>,<bVisible>);
```

Sets a filter rule for all items with a specified data model tag.

WriteFilterRuleSet

```
bSuccess = dsdd('WriteFilterRuleSet',attributeName1,attributeValue1,...]);
```

Writes a filter rule set to an XML file.

AUTOSAR

Where to go from here

Information in this section

[Supported AUTOSAR Releases](#)..... 146

[New AUTOSAR Features](#)..... 147

Supported AUTOSAR Releases

Supported AUTOSAR Releases

The following AUTOSAR Releases are supported:

AUTOSAR Release	Revision
4.1	4.1.3 ¹⁾
	4.1.2 ¹⁾
	4.1.1
4.0	4.0.3
	4.0.2
	3.2.3 ¹⁾
3.2	3.2.2
	3.2.1
	3.2.1

AUTOSAR Release	Revision
3.1	3.1.5
	3.1.4
	3.1.2
	3.1.0
3.0	3.0.7
	3.0.6
	3.0.4
2.1	3.0.2
	2.1.4

¹⁾ New in TargetLink 4.0

Specifying the AUTOSAR release in the TargetLink Data Dictionary

Dictionary TargetLink allows you to generate AUTOSAR-compliant code for both AUTOSAR Releases 2.x/3.x and 4.x.

You can specify which AUTOSAR Release to use in the `DD /Pool/Autosar/Config` object of the TargetLink Data Dictionary.

For information on generating AUTOSAR-compliant code, refer to [Generating Classic AUTOSAR-Compliant Code](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

Creating AUTOSAR-compliant DD workspaces with system templates

templates You can now create AUTOSAR-compliant DD workspaces using new system templates for both AUTOSAR Releases 3.x and 4.x. In the TargetLink Data Dictionary Manager, click File - New -Create New DD Workspace and select

- `dsdd_master_autosar3.dd` [System] for AUTOSAR 2.x and 3.x
- `dsdd_master_autosar4.dd` [System] for AUTOSAR 4.x

For further information, refer to [How to Create DD Workspaces](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

New AUTOSAR Features

Matrices at AUTOSAR interfaces TargetLink supports 2-D matrices for all AUTOSAR-related signals with the exception of inter-runnable variables.

CompuMethods of category IDENTICAL TargetLink's AUTOSAR import now supports CompuMethods of category **IDENTICAL**.

Port initialization TargetLink's AUTOSAR import now supports initialization values which are specified locally at AUTOSAR ports.

Global constants	TargetLink's AUTOSAR import now supports global constants with references to other global constants.				
PerInstanceCalPrm	The creation of PerInstanceCalPrm objects in the TargetLink Data Dictionary Manager was simplified.				
AUTOSAR memory mapping	You can now use the \$(Component) name macro within the values of the DeclarationStatements property of VariableClass objects to simplify the use of AUTOSAR memory mapping.				
Model-linked code view	TargetLink now supports model-linked code view in AUTOSAR code generation mode to provide improved traceability between models and generated code. For details, refer to Tracing Objects between Model and Code (Model-Linked Code View) (TargetLink Preparation and Simulation Guide).				
Support of IncludedDataTypeSets	TargetLink now supports IncludedDataTypeSets as described by the AUTOSAR 4.x standard.				
Import/export of NetworkRepresentations	TargetLink now imports/exports NetworkRepresentations elements.				
Support of SwImplPolicyEnum	TargetLink now provides the following property of DataElement objects:				
	<table border="1"> <thead> <tr> <th>Property</th><th>Value</th></tr> </thead> <tbody> <tr> <td>ImplementationPolicy</td><td> <ul style="list-style-type: none"> ▪ standard - For non-queued sender receiver communication ▪ queued - For queued sender receiver communication ▪ measurementPoint - The data element is used for measurement purposes only Only used for AUTOSAR file import/export </td></tr> </tbody> </table>	Property	Value	ImplementationPolicy	<ul style="list-style-type: none"> ▪ standard - For non-queued sender receiver communication ▪ queued - For queued sender receiver communication ▪ measurementPoint - The data element is used for measurement purposes only Only used for AUTOSAR file import/export
Property	Value				
ImplementationPolicy	<ul style="list-style-type: none"> ▪ standard - For non-queued sender receiver communication ▪ queued - For queued sender receiver communication ▪ measurementPoint - The data element is used for measurement purposes only Only used for AUTOSAR file import/export 				

Note

The ImplementationPolicy property replaces the IsQueued property. Refer to [Replaced IsQueued Property](#) on page 346 for compatibility considerations.

Test Support

Where to go from here

Information in this section

- [Improved Online Parameter Modification.....](#) 149
- [Changes in the Target Simulation Modules.....](#) 149

Improved Online Parameter Modification

Automatic update to changed MIL values

Before starting a simulation, you can now have TargetLink update parameter values in a SIL/PIL simulation application automatically. In detail, variable values are then automatically overwritten by their MIL value counterparts if the values' differences have changed and exceed a user-defined tolerance level. In addition, the online parameter update for variables in Stateflow is also supported.

Related documentation

- [Basics on Modifying Parameter Values for Simulation \(TargetLink Preparation and Simulation Guide\)](#)
- [How to Provide Automatic Parameter Updates via a Hook Script \(TargetLink Preparation and Simulation Guide\)](#)

Changes in the Target Simulation Modules

New and discontinued compiler versions

The following table shows the compiler versions that are now supported by TargetLink 4.0, refer to the New and No changes columns. Compiler versions that are no longer supported are listed in the Discontinued column.

Target	Compiler	New	No changes	Discontinued
C16x	TASKING	—	8.6, 8.7	—
HCS12	Cosmic	—	4.8	4.7
	Metrowerk	—	5.1	3.1
M32R	Gaio	—	11, 9	—
	Renesas	—	5.1	—
MC56F83	Metrowerk	—	8.3	—

Target	Compiler	New	No changes	Discontinued
MPC55xx	Diab	—	5.9	5.7
	GreenHill	2013	—	2012
	GNU	—	4.1	—
	Metrowerk	—	2.8	—
MPC55xxVLE	Diab	—	5.9	—
	GreenHill	2013	—	2012
	Metrowerk	—	2.8	—
MPC560xVLE	Diab	—	5.9	—
	GreenHill	2013	2012	5.2
MPC5xx	Diab	—	—	5.7
	GreenHill	—	—	5.1
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3, 9.4	—
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	4.3	3.2	4.2
TriCore1796	GNU	—	3.4	—
V850	GreenHill	2013	—	2012
	NEC	—	3.4	—
XC22xx	TASKING	—	3.0	—

For detailed information on the evaluation boards supported by TargetLink, refer to  [Evaluation Board Reference](#).

Note

The MPC5xx target is no longer supported by TargetLink but still distributed by dSPACE.

For further PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website at the [TargetLink Product Support Center](#).

Code Generator Options

New Code Generator Options

Overview of new Code Generator options

The following new Code Generator options are available with TargetLink 4.0.

Description	Explanation	Default
AssignmentOfConditions		
Controls the code pattern for Relational Operator and Logical Operator blocks and related optimizations.	<p>For Relational Operator and Logical Operator blocks, TargetLink lets you decide whether you want code of the form</p> <pre>if (<Condition>) { Output = 1; } else { Output = 0; }</pre> <p>or of the form</p> <pre>Output = <Condition>;</pre> <p>In addition, you can activate a control flow optimization that transforms the former code pattern into the latter, e.g., for code stemming from a Stateflow chart or a Switch block with one 0 and one 1 data input.</p> <p>The option values have the following effect:</p> <ul style="list-style-type: none"> ▪ 0 - None: Always generate the control flow pattern for Relational Operator and Logical Operator blocks and do not perform control flow optimization. ▪ 1 - RelationAndLogicBlocks: Generate the assignment pattern for Relational Operator and Logical Operator blocks where possible but do not perform control flow optimization. ▪ 2 - AllBooleanOutputs: Generate code as for the 'RelationAndLogicBlocks' setting and perform the control flow optimization if the output variable is of Boolean type. ▪ 3 - AllOutputs: Generate code as for setting 'RelationAndLogicBlocks' and perform the control flow optimization if the output variable is of any numerical type. <p>The last setting might unexpectedly lead to code that violates MISRA rules and performs arithmetic or bitwise operations with operands that are logic or relation operations.</p>	2
InsertComputeThroughOverflowComments		
Introduce additional code comments to mark casts introduced for Compute Through Overflow operations.	<p>To produce more efficient code, TargetLink allows controlled overflows in the calculation of additions/subtractions (Compute Through Overflow).</p> <p>When this option is enabled, comments are introduced in the generated code at casts that introduce these controlled overflows.</p> <ul style="list-style-type: none"> ▪ 0 - off ▪ 1 - on 	off

Description	Explanation	Default
Allow64BitMultiplicationsForArbitraryScaled16BitOperands		
Controls the code pattern of product operation with 16-bit operands.	<p>A product operation with 16-bit operands and an arbitrary scaling usually returns 64-bit code in order to generate code with the highest possible precision for the rescaling operations.</p> <p>If 64-bit code is to be avoided, this option allows to suppress this behavior for scale factors ≤ 1.</p> <p>This might cause a great loss of accuracy.</p> <ul style="list-style-type: none"> ▪ 0 - off ▪ 1 - on 	on
UtilizeExplicitDependenciesForScalingInvariantSystems		
Consideres optionally specified information on which import(s) a scaling invariant outport depends on.	<p>When this option is set to on, the optionally in the scaling propagation function specified information, specifying the import(s) a scaling invariant outport depends on, is taken into account for the code generation. Otherwise, a worst case assumption (i.e., each outport is dependent on every import of the scaling invariant system) is made for a scaling invariant system. Specifying the dependencies is necessary for the modeling of feedback loops with scaling invariant systems.</p> <ul style="list-style-type: none"> ▪ 0 - off ▪ 1 - on 	off

For reference information on all Code Generator options, refer to [Code Generator Options](#) ([TargetLink Model Element Reference](#)).

Migration aspects of Code Generator options	<p>Migration aspects include:</p> <ul style="list-style-type: none"> ▪ Obsolete Code Generator options ▪ Changed Code Generator options ▪ Recommended compatibility options for new Code Generator options to ensure the best possible downward compatibility in the generated production code <p>Refer to Migration Aspects Regarding Code Generator Options on page 350 for more information.</p>
--	--

Tool Chain Integration

Improved Windows Compliance

Installation files in user-defined locations	<p>This TargetLink version can search for the following customization files at arbitrary locations, i.e., outside your TargetLink installation:</p> <ul style="list-style-type: none"> ▪ DD template files ▪ DD Data Model filter rules
---	---

- DD menu extensions
- A2L style sheets

Related documentation

- [How to Define TargetLink's Search Path for DD-Related Customization Files](#)
([TargetLink Customization and Optimization Guide](#))

Import/export of TargetLink Preferences

You can now import and export the TargetLink preferences settings also via command line.

Related documentation

- [Basics on Using the Preferences Editor](#) ([TargetLink Customization and Optimization Guide](#))
- [How to Import and Export TargetLink Preferences via Command Line](#)
([TargetLink Customization and Optimization Guide](#))

Cloning TSMs

You can clone already installed TSMs and store them to the TSM Extension Packages folder (which can be specified in the Preferences Editor). They are then also listed in the Target Simulation Module > Preselection dialog of the Preferences Editor. Additionally, you can remove TSMs that do not belong to the TargetLink installation, for example, TSMs that were created by cloning an installed TSM or added as additional TSM packages.

Related documentation

- [How to Clone Target/Compiler Combinations to Outside the TargetLink Installation](#) ([TargetLink Customization and Optimization Guide](#))

Other

General Enhancements and Changes

Improved documentation generation

The following new features are available for TargetLink's documentation generation:

Support for Japanese characters in PDF generation TargetLink now also supports Japanese characters in the PDF document generation process. By default, TargetLink automatically detects the installed language of Microsoft® Windows® and then sets the appropriate character set and font. However, you can manually set the **CharacterSet** property in the **Config/General** object of the DD object tree. In addition, you have to set the font to be used in the generated PDF documentation in the script that controls the PDF documentation generation (default: `t1doc_pdf`).

New table of contents and optional cover page PDF document generation now provides a table of contents. The depth of the table of contents can be set by the user (see `t1doc_pdf` script for details).

Optionally, you can also add a cover page to the generated PDF. To do so, edit the `t1doc_pdf` M script and add a `CoverPage` entry as described in the script.

Disabling the generation of the function hierarchy You can now disable the generation of a hierarchical list of functions contained in the generated code. By default, the generation is activated. To disable it, edit the M script that controls the generation of the documentation (for example `t1doc_default`, `t1doc_pdf`, `t1doc_rtf`) and set the `FunctionsHierarchy` entry to 'off'.

Order of user-inserted chapters in the function list In former TargetLink versions, user-inserted chapters that were created via the Autodoc Customization block were not integrated in the navigation pane of the generated documentation. They were listed at the end of the contents. Now user-inserted chapters are integrated by default. This only applies to HTML-generated documentation.

For further information on the document generation process, refer to [Basics on Customizing the Generated Documentation](#) ( [TargetLink Interoperation and Exchange Guide](#)).

Improved consistency check for indices

TargetLink now performs consistency checks for indices during code generation.

For Assignment blocks and Simulink Selector blocks, TargetLink now performs a consistency check for indices.

If the block's index mode does not match the signal at the block's index port, an error message occurs.

Creating customization files via GUI

TargetLink provides a tool (GUI) to derive customization files from templates. In addition, you can use this tool to create DD menu extensions and A2L style sheets.

Related documentation

- [How to Create Customization Files via the Create Customization Files Dialog](#) ( [TargetLink Customization and Optimization Guide](#))
-

Obsolete limitations

With TargetLink 4.0, several limitations were removed. For details, refer to [Obsolete Limitations](#) on page 420.

API Commands

New API Functions

t1ExtractSubsystem

The **t1_extract_subsystem**, API function (TargetLink ≤ 3.5) was replaced.

TargetLink provides the **t1ExtractSubsystem** API function to generate a new independent TargetLink subsystem from a TargetLink subsystem part.

Related documentation

- [t1ExtractSubsystem](#) ([TargetLink API Reference](#))

t1SimInterface

The **t1_sim_interface** API function (TargetLink ≤ 3.5) was replaced.

TargetLink provides the **t1SimInterface** API function as M interface for the TargetLink simulation engine.

Related documentation

- [Implementing Online Parameter Modification](#) ([TargetLink Preparation and Simulation Guide](#))
- [t1SimInterface](#) ([TargetLink API Reference](#))

t1SimParameterUpdate

TargetLink provides the **t1SimParameterUpdate** API function to allow automatic online parameter modification.

Related documentation

- [Implementing Online Parameter Modification](#) ([TargetLink Preparation and Simulation Guide](#))
- [t1SimParameterUpdate](#) ([TargetLink API Reference](#))

t1MoveDDObject

TargetLink provides the **t1MoveDDObject** API function to move or rename Data Dictionary objects and adapt references to this object.

Related documentation

- [t1MoveDDObject](#) ([TargetLink API Reference](#))
- [t1FindDDReferences](#) ([TargetLink API Reference](#))

t1SyncSystemSignature

TargetLink provides the **t1SyncSystemSignature** API function to generate a new Simulink system from a Data Dictionary Signature or Block object.

Related documentation

- [t1SyncSystemSignature](#) ([TargetLink API Reference](#))

tlOperationMode	The tl_switch_blockset API function (TargetLink ≤ 3.5) was replaced. TargetLink provides the tlOperationMode API function to get or set TargetLink's operation mode (<i>full-featured</i> versus <i>stand-alone</i>). Related documentation <ul style="list-style-type: none">▪ tlOperationMode (TargetLink API Reference)
tlUpgrade	The tl_upgrade API function (TargetLink ≤ 3.5) was replaced. TargetLink provides the tlUpgrade API function to upgrade block diagrams containing TargetLink blocks to the current version. Related documentation <ul style="list-style-type: none">▪ tlUpgrade (TargetLink API Reference)
TlTsmManager	TargetLink provides the TlTsmManager.exe command to clone existing TargetLink Simulation Modules. Related documentation <ul style="list-style-type: none">▪ How to Clone Target/Compiler Combinations to Outside the TargetLink Installation (TargetLink Customization and Optimization Guide)

New Features of TargetLink 3.5

Where to go from here

Information in this section

Modeling in Simulink or Stateflow.....	157
Code Generation Core Functionality.....	159
Data Dictionary and Datamanagement.....	160
AUTOSAR.....	164
Testing Support.....	169
Code Generator Options.....	172
Toolchain Integration.....	175
Documentation.....	177
Other.....	178
API Commands.....	181
Hook Functions.....	182

Modeling in Simulink or Stateflow

Where to go from here

Information in this section

Newly Supported Simulink Blocks.....	157
Enumeration Support.....	157
Improved System Preparation.....	158
Improved Stateflow Support.....	159

Newly Supported Simulink Blocks

Variant Subsystem block

TargetLink supports the Simulink Variant Subsystem block as follows:

- TargetLink always uses the active subsystem variant. Inactive subsystem variants are ignored.
- System preparation is possible for active and, with slight restrictions, also for inactive subsystem variants. For details, refer to [Basics on Preparing Simulink Systems for TargetLink Code Generation](#) ([TargetLink Preparation and Simulation Guide](#)).

Model Variant block

TargetLink supports the Simulink Model Variant block as follows, which reflects the usual model referencing behavior:

- TargetLink processes only the active model variant.
- The inactive model variants are not processed.

Enumeration Support

Support of Simulink enumeration data types

TargetLink now supports Simulink enumeration data types:

- TargetLink's system preparation replaces a Simulink Enumerated Constant block by a standard TargetLink **Constant Block** ([TargetLink Model Element Reference](#)). When TargetLink data is cleared from a system, system preparation replaces the TargetLink Constant block by a Simulink Enumerated Constant block.
- For details, refer to [Basics on Making Simulink Enumerations TargetLink-Compliant](#) ([TargetLink Preparation and Simulation Guide](#)).

- By default, TargetLink maps Simulink enumeration data types used in Simulink blocks and Stateflow charts onto the new `EnumImplementation` TargetLink data type used for code generation during system preparation. You can change this data type mapping by using specific mapping functions.
- TargetLink's production code does not contain enumeration values (e.g., Red, Yellow, and Green) but only the enumeration's integer values (e.g., 0, 1, and 2). Macros are also not supported. The same applies to logging and plotting because an enumeration's integer values are used. The base data type of the `EnumImplementation` data type is set to `Int32` in the Data Dictionary templates, which you can change to other integer data types.

Improved System Preparation

Enhancements to Simulink ports

Root system You can decide whether the Simulink port blocks of the system that is going to be prepared are enhanced or not. These ports were always enhanced in older TargetLink versions. For details, refer to [System Preparation Dialog](#) ( [TargetLink Tool and Utility Reference](#)).

Underlying systems You can decide whether the Simulink ports of nested subsystems are enhanced or not. TargetLink lets you specify the nested subsystems that are to be considered. Specification of the subsystems is possible via dialog and API. For details, refer to [Subsystems whose ports should be enhanced](#) in the [System Preparation Dialog](#) ( [TargetLink Tool and Utility Reference](#)).

TargetLink simulation frame

You can add and remove TargetLink simulation frames via API commands. These operations do not affect a subsystem's ID. For details on these API commands, refer to `tl_addsimframe` and `tl_removesimframe`.

Block replacement

TargetLink now allows you to override standard rules for the enhancement of Simulink blocks. For example, you can enhance a Simulink Gain block to a TargetLink Custom Code block instead of a TargetLink Gain block. Filter conditions as part of the libmap definitions allow you to exactly control in which situations the changed behavior is applied. For details on libmaps, refer to [How to Configure Block Replacement](#) ( [TargetLink Preparation and Simulation Guide](#)).

Utility for block replacement

TargetLink provides a utility for easily enhancing unsupported Simulink blocks by TargetLink Custom Code (type II) blocks. This is very helpful for Simulink blocks that are not natively supported by TargetLink.

Related topics**References**

[tl_addsimframe](#) ([TargetLink API Reference](#))
[tl_removesimframe](#) ([TargetLink API Reference](#))

Improved Stateflow Support

Improved Stateflow support

TargetLink now supports the Execute (enter) Chart At Initialization chart property for the code generation. As a result, you can specify whether a chart's state configuration is initialized at model initialization time, which is new with TargetLink 3.5, or later, that is, at the occurrence of the first chart input event, which is the default setting.

Enumeration data types

TargetLink supports Simulink enumeration data types used in Stateflow, refer to [Enumeration Support](#) on page 157.

Variant modeling with preprocessor #if

The code for executing and exiting substates has been optimized by replacing the IF queries of the states' activity variables by preprocessor IF statements. This optimization is applied to substates ($N \geq 2$) of a parent state with exclusive decomposition under specific conditions. For details, refer to [Basics on Substituting IF Queries with Preprocessor IF Statements](#) ([TargetLink Preparation and Simulation Guide](#)).

Code Generation Core Functionality

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

MISRA-C:2004 compliance	160
Improved Code Efficiency	160

MISRA-C:2004 compliance

Improvements to comply with MISRA-C:2004

Several improvements were made to TargetLink's Fixed-Point Library to comply with MISRA-C:2004. Improvements include:

- Bracketing of macro parameters
- Bracketing of bodies of loops and control flows
- Naming of module guards
- Using same parameter names in declarations and definitions

For the generated code TargetLink now brackets the arguments of the conditional operator (? :), if the argument is an operation.

Improved Code Efficiency

Elimination of zero-initialized state variables

A state variable that is copied to another variable with static storage duration is replaced by the other variable if the initial values are equal. The implicit zero initialization value of uninitialized static storage duration variables now is taken into account and leads to elimination of more unnecessary state variables.

Related documentation

- UtilizeZeroInitializationOfStaticStorageDurationVariables (refer to [Code Generator Options](#) on page 172) Code Generator option
- [Code Changes between TargetLink 3.4 and TargetLink 3.5](#) on page 547

Data Dictionary and Datamanagement

Where to go from here

Information in this section

Enhanced Usability of the Data Dictionary Manager.....	161
Further Improvements to the Data Dictionary.....	163
New DD MATLAB API Commands.....	164

Enhanced Usability of the Data Dictionary Manager

Automatic adaptation of references while moving or renaming DD objects

The Reference Handling Options let you specify the way references in DD objects will be handled when changing DD objects. By default, you can let the Data Dictionary Manager automatically adapt references for you. You can also disable this function via the Never Adapt References option. For information on configuring the Reference Handling Options, refer to [Reference Handling Options](#) ([TargetLink Data Dictionary Manager Reference](#)).

Accessing DD objects via DD handles

The Data Dictionary Manager now supports quick access to DD objects by entering Data Dictionary object handles (DD handles). Each DD object is identified by a unique handle number. Instead of using the object path, you can simply enter or paste the handle number into the DD object list. You can use this feature if you develop and debug M-scripts and need quick access to an object in the Data Dictionary Manager. However, DD handles are not persistent and expire if you delete their objects.

You can get the handle number in different ways, for example:

- using the `GetDDAttributes` DD MATLAB API command.
- using the Show Object Details context command in the DD Navigator pane

For further information, refer to [Basics on Accessing DD Objects via Paths and Handles](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Easier swapping of DD workspaces

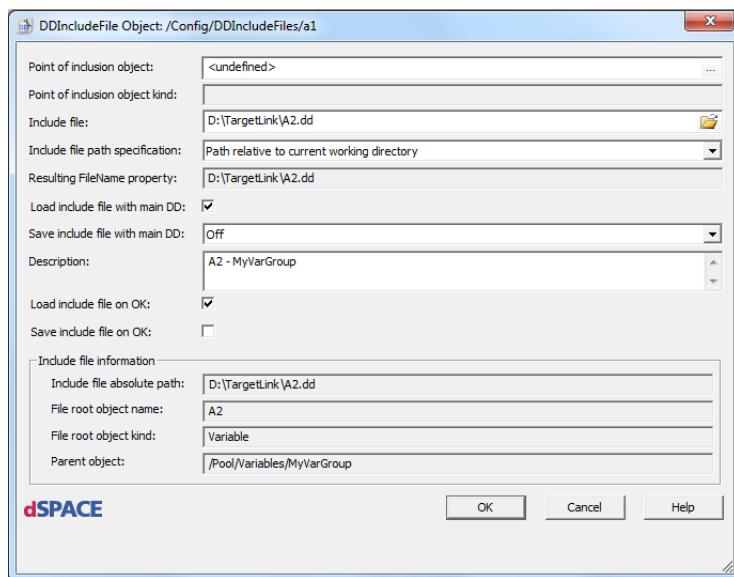
The new Make to Primary Workspace toolbar button allows you to easily swap DD workspaces. Select a DD workspace (except for DDO itself) and make it the primary workspace (DD0) with one click.

For further information on swapping DD workspaces, refer to [How to Swap a Secondary DD Workspace with the Primary DD Workspace](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

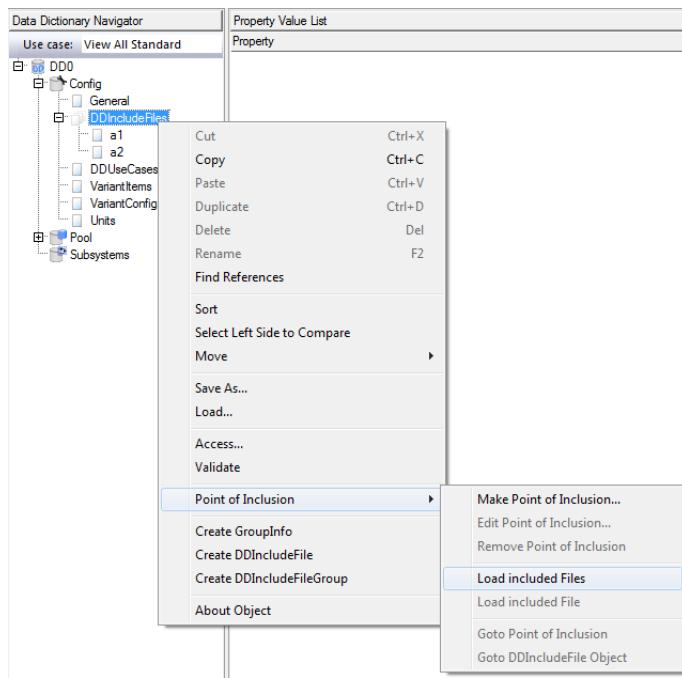
Improved handling of DDIncludeFile objects

To easily transfer DD files, you can use the Point of Inclusion functions of the Data Dictionary Manager. Another quick method to manage points of inclusion is to handle their corresponding DDIncludeFile objects in the `Config/DDIncludeFiles` object tree:

- You can now access DDIncludeFile objects by double-clicking them in the `Config/DDIncludeFiles` subtree.
- A new dialog lets you edit the DDIncludeFile objects and load the corresponding included file, even if the point of inclusion object is yet to be defined.



- In the context menu of the DDIncludeFile object, you can jump to the corresponding point of inclusion object (if already defined) or load its included file. In the parent DD tree, you can even load all included files.

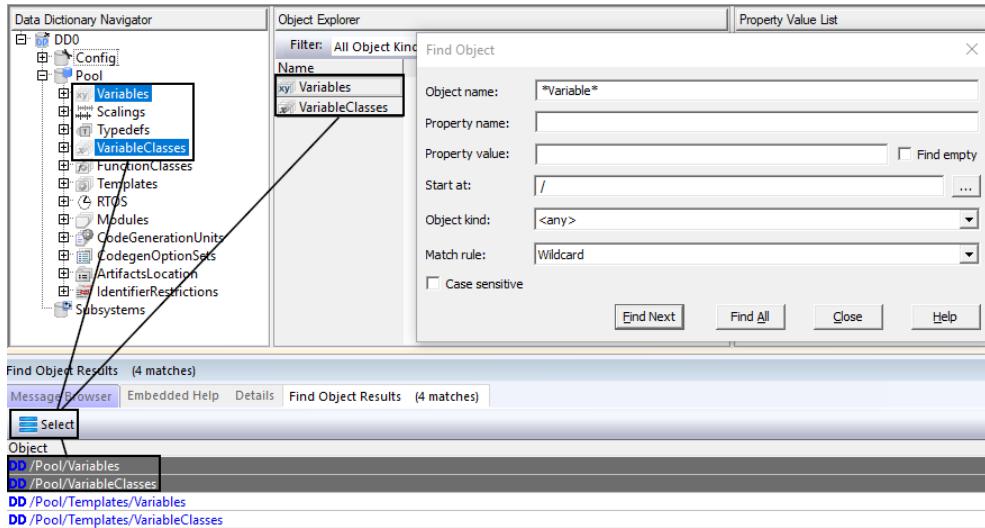


For further information on DDIncludeFile objects and Points of Inclusion, refer to [Point of Inclusion](#) ([TargetLink Data Dictionary Manager Reference](#)).

Improved selection of multiple objects in panes

A new Select button makes it easy to multiselect objects in the following panes: Message Browser, Find Object Results pane, Find Reference pane and Custom Output pane. For example, if you have multiple search results in the

In the Find Object Results pane, you can now multiselect them via left click and then click the Select button. All the selected search results are then automatically displayed/selected in the Data Dictionary Navigator and in the Object Explorer as shown in the illustration below.



For further information, refer to [Overview of the User Interface \(TargetLink Data Dictionary Basic Concepts Guide\)](#).

Related topics

Basics

[Overview of the User Interface \(TargetLink Data Dictionary Basic Concepts Guide\)](#)

HowTos

[How to Swap a Secondary DD Workspace with the Primary DD Workspace \(TargetLink Data Dictionary Basic Concepts Guide\)](#)

References

[Point of Inclusion \(TargetLink Data Dictionary Manager Reference\)](#)
[Reference Handling Options \(TargetLink Data Dictionary Manager Reference\)](#)

Further Improvements to the Data Dictionary

Properties written to embedded objects

It is now possible to write properties directly to embedded objects via DD API commands.

Refer to the following example:

```
dsdd('Set','/Pool/Variables/GroupInfo','CustomProp',1001);
```

New DD MATLAB API Commands

Overview of new DD MATLAB API commands

The following new DD MATLAB API commands are available:

- **'HandleRefsToStringRefs'**
- **'SetUuids'**

'HandleRefsToStringRefs' dsdd('HandleRefsToStringRefs',<objectIdentifier>[,<attributeName>,<attributeValue>])

Converts all handle reference properties of all objects in a subtree into string reference properties. Empty reference properties are left untouched. This also applies to reference properties with more than one variant of which at least one is empty.

'SetUuids' dsdd('SetUuids',<objectIdentifier>)

Sets the Uuid property for all objects in the subtree which have this property according to the Data Model, but do not have it set.

Related documentation

- HandleRefsToStringRefs
- SetUuids

AUTOSAR

Where to go from here

Information in this section

Supported AUTOSAR Releases.....	165
Multiple Instantiation of Software Components.....	165
Per Instance Calibratable Parameters.....	166
Support of Code Generation Units Containing Only Operation Calls.....	166
Easier Specification of AUTOSAR Objects in the Data Dictionary Manager.....	167
Improved Restart Function Behavior.....	168
Improved Container Management.....	168

Supported AUTOSAR Releases

Supported AUTOSAR Releases

TargetLink now supports AUTOSAR Release 4.1, Revision 4.1.1. The following AUTOSAR Releases are supported:

AUTOSAR Release	Revision
4.1	4.1.1 ¹⁾
4.0	4.0.3
	4.0.2
3.2	3.2.2
	3.2.1
3.1	3.1.5
	3.1.4
	3.1.2
	3.1.0
3.0	3.0.7
	3.0.6
	3.0.4
	3.0.2
2.1	2.1.4

¹⁾ New in TargetLink 3.5

Specifying the AUTOSAR release in the TargetLink Data Dictionary

Dictionary TargetLink allows you to generate AUTOSAR-compliant code for both AUTOSAR Releases 3.x/4.x.

You can specify which AUTOSAR Release to use in the `DD /Pool/AUTOSAR/Config` object the TargetLink Data Dictionary.

For information on generating AUTOSAR-compliant code, refer to [Generating Classic AUTOSAR-Compliant Code](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

Multiple Instantiation of Software Components

Multiple Instantiation of SWCs

AUTOSAR supports multiple instantiation for code reuse. This lets you reduce memory overhead and testing efforts.

TargetLink helps you prepare SWC types for multiple instantiation. It performs all the necessary steps during code generation:

- Analyzes your model and DD-based specification and collects all the variables with static duration that belong to the SWC type prepared for multiple instantiation (states).

- Creates a PIM for each SWC type prepared for multiple instantiation. It automatically determines the correct data type of the PIM based on the collected states and moves them to the autogenerated PIM.
- Generates code such that all calls to instance-specific entities carry the instance handle as their first parameter. It ensures that the instance handle is propagated to subfunctions accordingly.
- Reports variables that could not be (automatically) implemented as PIM in its Code Generation Report.

You can simulate a single SWC instance in all the three simulation modes. Multiple instances can be simulated in the context of virtual validation.

Related documentation

- [Preparing SWCs for Multiple Instantiation](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))
- [Analyzing the Code Generation Report](#) ([TargetLink Preparation and Simulation Guide](#))
- [AR_POSCONTROL](#) ([TargetLink Demo Models](#))

Per Instance Calibratable Parameters

Instance-specific calibration parameters

TargetLink can create instance-specific calibration parameters (PerInstanceCalPrms) during code generation. The import- and export- of instance-specific calibration parameters from/to AUTOSAR files is also supported.

You have to explicitly specify how to implement calibration parameters via the Class property of the DD Variable objects used to specify the calibration parameters:

Parameter Type	Variable Class
Shared	AUTOSAR/SHARED_CALPRM
Instance specific	AUTOSAR/PER_INSTANCE_CALPRM

Reference the DD VariableGroup object containing your parameters at the CalibratablesRef/PerInstanceCalibratablesRef property of the RelatedVariables object belonging to the DD SoftwareComponent object. For instructions, refer to [Basics on Preparing SWCs for Multiple Instantiation](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

Support of Code Generation Units Containing Only Operation Calls

CGUs containing only operation calls

TargetLink lets you model operation calls as top-level subsystems into single code generation units (CGUs).

Related documentation For details, refer to [Partitioning Model and Code for Classic AUTOSAR](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

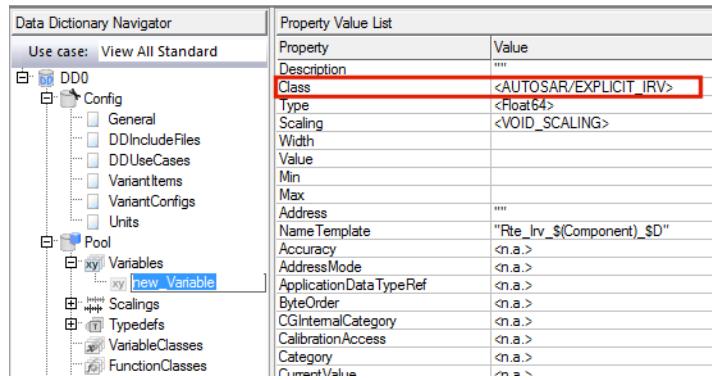
Easier Specification of AUTOSAR Objects in the Data Dictionary Manager

Specifying AUTOSAR-compliant Variable objects

The new AUTOSAR context command in the DD Navigator pane allows you to specify **Variable** objects with AUTOSAR-compliant settings.

If you use the command on a **VariableGroup** object, a new AUTOSAR-compliant **Variable** object is created in the selected group.

If you use the command on an existing **Variable** object, the object is converted and given AUTOSAR-compliant settings.



The following AUTOSAR-compliant settings are supported for **Variable** objects:

- Explicit IRV
- Implicit IRV
- PIM
- SharedCalPRM
- Composite SharedCalPrm

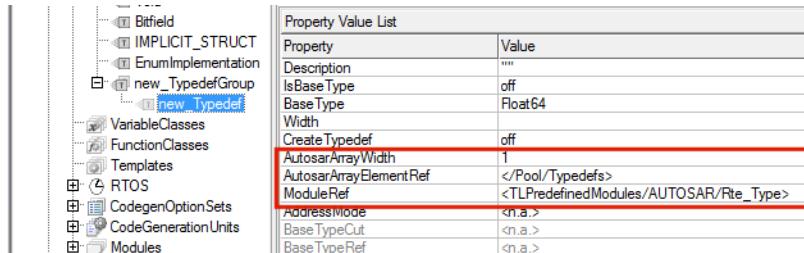
For further information, refer to [AUTOSAR](#) ([TargetLink Data Dictionary Manager Reference](#)).

Specifying AUTOSAR-compliant Typedef objects

The AUTOSAR context command in the DD Navigator pane allows you to specify **Typedef** objects with AUTOSAR-compliant settings.

If you use the command on a **TypedefGroup** object, a new AUTOSAR-compliant **Typedef** object is created in the selected group.

If you use the command on an existing `Typedef` object, the object is converted and given AUTOSAR-compliant settings.



The following AUTOSAR-compliant setting is supported for `Typedef` objects:

- AUTOSAR Array Type

For further information, refer to AUTOSAR ([TargetLink Data Dictionary Manager Reference](#)).

Improved Restart Function Behavior

Improved restart function behavior for incrementally generated runnable code

TargetLink's restart function behavior for incrementally generated runnable code has been improved:

For each runnable a restart function is created and called in TargetLink's restart runnable. This lets you initialize each runnable's state variables during restart in incremental code generation or several TargetLink subsystems.

The new DD `NoRestartCode` property specified at DD `Runnable` objects lets you suppress the restart function generation.

Error handling If TargetLink detects necessary restart code suppressed via the `NoRestartCode` property, it displays an error message.

If you set the `SuppressNoRunnableRestartCodeError` Code Generator option to on, TargetLink displays a warning instead.

Related documentation

- [TargetLink AUTOSAR Guidelines](#) ([TargetLink Modeling Guidelines](#))
- [How to Model Runnables](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))
- [Runnable](#)

Improved Container Management

Improvements of container management

Container management was improved.

- Improved assignment of elements to container files in SystemDesk 4.5:

With this version of the Container Manager, assignment of AUTOSAR elements to container file assignments has been improved for SystemDesk 4.5. The file assignments are required for exchanging software components between SystemDesk and TargetLink.

- Creating Simulation Systems for Virtual Validation:
You can now use SystemDesk's ECU configuration framework for configuring module configurations of code-based V-ECUs.

Testing Support

Where to go from here

Information in this section

Signal Injection/Tunneling.....	169
Third-Party Tool Support for Code Coverage Measurements.....	170
Enhanced Overflow Detection.....	170
Changes in the Target Simulation Modules.....	170

Signal Injection/Tunneling

Injecting/tunneling signals without a connection line

For testing purposes, TargetLink allows you to inject/tunnel signals without a connection line in all the three simulation modes. This lets you:

- Inject stimulus signals into TargetLink subsystems
- Tunnel signals from TargetLink subsystems to the outside

Both of the approaches listed above are modeled with [Simulink data stores](#) and Data Store Read/Write blocks. A new demo model is provided for illustration.

Related documentation

- [Modifying Signal Variables During Simulation \(Signal Injection/Tunneling\)](#)
([TargetLink Preparation and Simulation Guide](#))
- [SIGNAL_INJECTION](#) ([TargetLink Demo Models](#))

Third-Party Tool Support for Code Coverage Measurements

Support for third-party code coverage measurement tools

TargetLink now provides additional support for third-party code coverage measurement tools, such as the CTC Testwell Code Coverage (<http://www.testwell.fi/ctcdesc.html>).

Related documentation

- [How to Measure Code Coverage via Third-Party Tools \(Testwell CTC\)](#) ([TargetLink Preparation and Simulation Guide](#))
- [t1CodeCoverage](#) ([TargetLink API Reference](#))

Enhanced Overflow Detection

Supported data types

TargetLink now supports overflow detection for single and integer Simulink/Stateflow data types. This is independent of TargetLink's data type settings.

For details, refer to [Overflow Detection](#) ([TargetLink Preparation and Simulation Guide](#)).

Message types and saturation

TargetLink displays a message for each overflow it encounters during simulation. The message type depends on the overflow's circumstances. Note the following distinction:

Saturation for Code Generation Selected	Message Type
✓	Note
—	Warning

For details, refer to [Details on the Visualization of Output Overflow](#) ([TargetLink Preparation and Simulation Guide](#)).

Changes in the Target Simulation Modules

New compiler versions

The following table shows the compiler versions that are now supported by TargetLink 3.5. There is 1 new version and 1 discontinued version.

Target	Compiler	New	No changes	Discontinued
C16x	TASKING	—	8.6, 8.7	—
HCS12	Cosmic	—	4.7, 4.8	—
	Metrowerk	—	5.1, 3.1	—

Target	Compiler	New	No changes	Discontinued
M32R	Gaio	—	11, 9	—
	Renesas	—	5.1	—
MC56F83	Metrowerk	—	8.3	—
MPC55xx	Diab	—	5.9, 5.7	—
	GreenHill	—	2012	—
	GNU	—	4.1	—
	Metrowerk	—	2.8	—
MPC55xxVLE	Diab	—	5.9	—
	GreenHill	—	2012	—
	Metrowerk	—	2.8	—
MPC560xVLE	Diab	—	5.9	—
	GreenHill	—	2012, 5.2	—
MPC5xx	Diab	—	5.7	—
	GreenHill	—	5.1	—
S12X	Cosmic	—	4.8	—
	Metrowerk	—	5.1	—
SH2	Renesas	—	9.3, 9.4	—
SH2A-FPU	Renesas	—	9.4	—
TriCore17xx	TASKING	4.2	3.2	4.0
TriCore1796	GNU	—	3.4	—
V850	GreenHill	—	2012	—
	NEC	—	3.4	—
XC22xx	TASKING	—	3.0	—

For detailed information on the evaluation boards supported by TargetLink, refer to  [Evaluation Board Reference](#).

Note

For further PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website.

Code Generator Options

Code Generator Options

New Code Generator Options

The following new Code Generator Options are available with TargetLink 3.5.

Description	Explanation	Behavior <= TL3.4.	Default
EnableReportGeneration			
Enables the generation of the Code Generation Report.	If enabled, the Code Generator creates an XML-based report. The report contains information like the current value of the Code Generator options, the messages generated during code generation, and other information controlled by other options.	off	off
ImplementFloatEqualityViaMacros			
Uses approximation for float equality.	If an equality operation == or != is necessary, macro calls (C_EQFLT, C_EQDBL, C_NEFLT, C_NEBDL) are generated instead of a plain operation. These code patterns allow compliance with the MISRA standard. But the generated code may be less efficient, because the optimization will not eliminate such equality operations if macros are used.	off	off
ReportMismatchingPortBlockSignalSpecifications			
Enables an optional report indicating mismatching port block signal specifications.	If this option is enabled (and the option "EnableReportGeneration" is enabled) an optional report is generated, which indicates mismatches in the data type, scaling and min/max values between a port and its predecessor blocks. This may indicate an unwanted loss of accuracy.	off	on
ReportVariablesExcludedFromPIM			
Enables an optional report indicating which variables could not be implemented as PIM and why.	If this option is enabled (and the option "EnableReportGeneration" is enabled) an optional report is generated, which gives detailed information on which variables could not be implemented as PIM and why.	off	on

Description	Explanation	Behavior <= TL3.4.	Default
SuppressNoRunnableRestartCodeError			
Emit a warning instead of an error if there is restart code for a NoRestartCode runnable.	The generation of runnable-specific restart functions can be omitted by setting a runnable's NoRestartCode property to 'on'. If restart code is necessary for such a runnable, TargetLink will issue an error message. Setting this option changes this error to a warning, which makes it possible to analyze the generated code and to find the reason for the unwanted restart code.	on	off
UtilizeZeroInitializationOfStaticStorageDurationVariables			
Assume that "no initial value" means "initial value all zeros" for variables with static storage duration and utilize this for the elimination of intermediate variables in situations where initial values have to be equal.	The C language demands that variables with static storage duration without an explicit initializer are initialized as if there was an initializer with 0 entry (or an initializer list with all 0 entries). Due to special compiler or linker configurations for the target platform, this may not hold for the target code. This option allows TargetLink to assume explicitly specified zero initialization. This affects in particular the elimination of output variables of conditionally executed subsystems. Often the default initial value 0 applies to these variables. Example: An enabled subsystem's output drives a TargetLink Outport block of the TargetLink subsystem. The output variable of the TargetLink OutPort is a global variable (i.e. has static storage duration), but has no explicit initial value derived from Simulink semantics. The output variable of the enabled	off	on

Description	Explanation	Behavior <= TL3.4.	Default
	<p>subsystem has the initial value 0. This leads to the following situation:</p> <pre data-bbox="774 403 1081 635">Int16 Output; void Subsystem(void) { static Int16 EnabledOut = 0; if (condition) { ... EnabledOut = ...; } Output = EnabledOut; }</pre> <p>EnabledOut can be eliminated only if Output has the same initial value -- otherwise the simulation behavior would change until the first time "condition" is true.</p> <p>If the option is activated, TargetLink performs this optimization for the above code. In addition, TargetLink makes the zero initial value explicit:</p> <pre data-bbox="774 925 997 1100">Int16 Output = 0; void Subsystem(void) { if (condition) { ... Output = ...; } }</pre> <p>If the option is deactivated, then TargetLink assumes a special-purpose initialization for the variable Output.</p> <p>Note that you can make the initial value explicit and achieve optimization independently of the option's setting if Output is a Data Dictionary variable with a 0 value property.</p>		

For reference information on all Code Generator options, refer to [Code Generator Options](#) ( [TargetLink Model Element Reference](#)).

Toolchain Integration

Where to go from here

Information in this section

Handling Customization Files.....	175
Enhanced Handling and Restoring of Demo Models.....	175
TargetLink Simulation Module Extension Packages.....	176

Handling Customization Files

Customization file templates

TargetLink's customization files, such as hook functions and the files related to data type synchronization, are now shipped as template files (SAM). You can use the `t1CustomizationFiles('Create',...)` API function to derive functional customization files (M).

Customization file folder

You can place the derived customization files in a customization file folder outside the TargetLink installation folder. This lets you use customization files without administrator privileges and facilitates work in large workgroups.

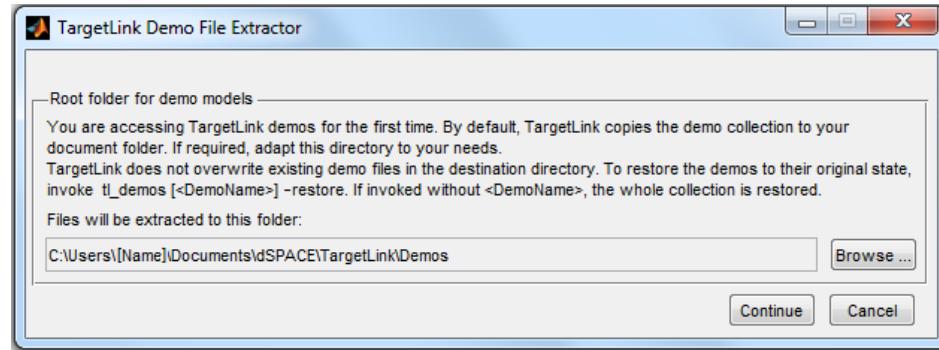
Related documents

- [Defining Search Paths for Customization Files, Demo Models, TSM Extension Packages, and Instruction Set Simulators](#) ([TargetLink Customization and Optimization Guide](#))
- [How to Define TargetLink's Search Path for DD-Related Customization Files](#) ([TargetLink Customization and Optimization Guide](#))
- [How to Define TargetLink's Search Path for DD-Related Customization Files](#) ([TargetLink Customization and Optimization Guide](#))
- [t1CustomizationFiles](#) ([TargetLink API Reference](#))

Enhanced Handling and Restoring of Demo Models

Selecting a root folder for the demo models

In TargetLink 3.4, the TargetLink demo model collection was saved in the TargetLink installation folder. This meant it was not possible to make any changes in the model without administrator privileges. When you first start a demo model in TargetLink 3.5, you are prompted to select a root folder. By default, the **Documents** folder is selected (`<DocumentsFolder>\dSPACE\TargetLink\<Version>\Demos`).



Restoring the demo models to their original state

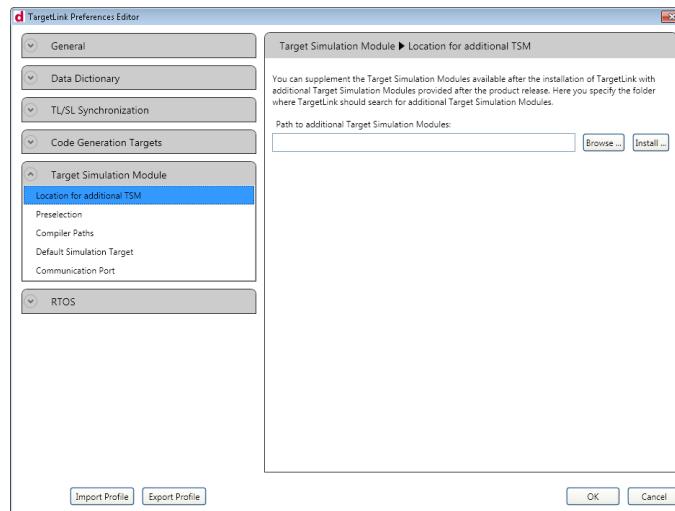
You can restore a demo model in the selected root folder with the `t1_demos [<DemoName>] -restore` command. If you use the command without `<DemoName>`, all demo models are restored to their original state.

TargetLink Simulation Module Extension Packages

Location for additional TSM

TargetLink lets you install Simulation Module Extension Packages outside the installation directory.

You can install the extension packages via the Preferences Editor:



Related documentation For details, refer to [Customizing the TargetLink Environment](#) ([TargetLink Customization and Optimization Guide](#)).

Documentation

Improved Documentation

New user guides

With TargetLink 3.5, the structure of the user documentation has been improved. To facilitate access to the documentation, the topics are now grouped by their use case.

As a result, new use-case-specific guides are being introduced:

- The all-new  [TargetLink Orientation and Overview Guide](#)

This guide contains basic information for all people who work with TargetLink, from beginners up to expert users. It provides an introduction to TargetLink and points you to the user documentation that applies to your use case. It also lists TargetLink limitations and provides a glossary of terms used throughout the TargetLink documentation.

-  [TargetLink Preparation and Simulation Guide](#)

This guide shows you how to use TargetLink out-of-the-box for preparing Simulink models for code generation and for simulating and testing the generated production code.

-  [TargetLink Customization and Optimization Guide](#)

This guide shows you how to adapt TargetLink settings so that the generated production code meets your specific requirements.

-  [TargetLink Interoperation and Exchange Guide](#)

This guide shows you how to use TargetLink as part of a tool chain, for example, exchanging data with other tools or using TargetLink via command line.

Replaced guides

With the introduction of the new guides listed above, some of the previous guides in TargetLink have been replaced.

Previous Guides	New Guides
TargetLink Production Code Generation Guide	TargetLink Preparation and Simulation Guide
TargetLink Advanced Practices Guide	TargetLink Customization and Optimization Guide TargetLink Interoperation and Exchange Guide

Other

Where to go from here

Information in this section

New Code Generation Report.....	178
General Enhancements and Changes.....	179

New Code Generation Report

Report to locate critical or inefficient code parts

You can create a report during code generation. The report contains the following information:

- Messages that have been generated (also shown in the Message Browser)
- The code generator options used

Additional information helps you to locate critical or inefficient code parts during the development process:

- A list of mismatching port block signal specifications
- A list of variables not implemented as per instance memory (PIM)

The Code Generation Report is available in two file formats:

- An HTML file (`CGReport.html`) for viewing which you can open in the MATLAB Command Window after code generation
- An XML file (`CGReport.xml`) which you can use to further process the report contents in other tools/toolchains

The Code Generation Report contains useful links to the model or the Data Dictionary. The screenshot below is an example of a list containing links to mismatching port block signal specifications.

Mismatching Port Block Signal Specifications

Available subsystems

- [poscontrol/controller/Subsystem/controller/Linearization](#)

Linearization [poscontrol/controller/Subsystem/controller/Linearization](#)

min/max mismatch	Port block: Simulink: poscontrol/controller/Subsystem/controller/Linearization/POSITION Signal indices: 1
	Predecessor block: Simulink: poscontrol/controller/Subsystem/controller/POS Port Nr.: 1 Signal indices: 1

For basic information on the Code Generation Report and its lists, refer to [Basics on the Code Generation Report](#) ([TargetLink Preparation and Simulation Guide](#)).

For information on how to create a Code Generation report, refer to [How to Customize the Code Generation Report](#) ([TargetLink Preparation and Simulation Guide](#)).

Related topics

Basics

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

HowTos

[How to Customize the Code Generation Report](#) ([TargetLink Preparation and Simulation Guide](#))

General Enhancements and Changes

Latched InPorts

TargetLink now fully supports the Simulink *Latch input for feedback signals of function-call subsystem outputs* property of port blocks.

Initial values of TargetLink outports

You can specify initial values for the TargetLink OutPort Block and Bus Outport Block as in their Simulink counterparts. This is helpful (and only possible) if these blocks reside in conditionally executed subsystems.

Restriction Bus signals must use the same scalar initial value.

Debugging scaling-invariant functions

By using the predefined `Failure` and `FailureMsg` variables in your M script, you can identify blocks that do not meet the preconditions for implementing scaling-invariant functions.

For an example, refer to [Details on the Scaling Propagation Function](#) ([TargetLink Customization and Optimization Guide](#)).

Adaptation of Simulink configuration parameters

TargetLink now notifies you of inconsistent bus signals. A certain Simulink configuration setting is expected. For details, refer to [Basics on Preparing Simulink Systems for TargetLink Code Generation](#) ([TargetLink Preparation and Simulation Guide](#)).

In a future version of TargetLink this configuration setting will become mandatory.

Floating-point parts encapsulated in Fixed-Point library

All the floating-point components of TargetLink's Fixed-Point Library are now encapsulated by the `TL_NO_FLOATS` or `TL_NO_FLOAT64` preprocessor directive.

This lets you control usage/removal of floating point code from the library during compilation:

Directive	Purpose
<code>TL_NO_FLOATS</code>	Suppress all the floating-point related macros and functions of the Fixed-Point Library (32-Bit and 64-Bit)
<code>TL_NO_FLOAT64</code>	Only suppress 64-bit floating-point related macros and functions of the Fixed-Point Library

Related documentation

- [Basics on Conditional Code Compilation via Preprocessor Directives](#) ([TargetLink Customization and Optimization Guide](#))
- [Macros and Functions Provided by the Fixed-Point Library](#) ([TargetLink File Reference](#))
- [Controlling the Use of Fixed-Point Library Macros and Functions](#) ([TargetLink File Reference](#))

Obsolete limitations

With TargetLink 3.5, several limitations have been removed. For details, refer to [Obsolete Limitations](#) on page 423.

Related topics**References**

Bus Outport Block ([TargetLink Model Element Reference](#))
OutPort Block ([TargetLink Model Element Reference](#))

API Commands

New API Commands

tlCustomizationFiles

TargetLink provides the **tlCustomizationFiles('Create',...)** API command that assists you in handling TargetLink's customization files and hook function templates.

Related documentation:

- [tlCustomizationFiles \(\[TargetLink API Reference\]\(#\)\)](#)
- [How to Create Customization Files via the Create Customization Files Dialog \(\[TargetLink Customization and Optimization Guide\]\(#\)\)](#)
- [How to Define TargetLink's Search Path for DD-Related Customization Files \(\[TargetLink Customization and Optimization Guide\]\(#\)\)](#)

tlCodeCoverage

TargetLink provides the **tlCodeCoverage** API command for support of the third-party code coverage tools.

Related documentation: For details, refer to [How to Measure Code Coverage via Third-Party Tools \(Testwell CTC\) \(\[TargetLink Preparation and Simulation Guide\]\(#\)\)](#).

tlFindDDReferences

TargetLink provides the **tlFindDDReferences** API command, which returns TargetLink blocks, Stateflow or Data Dictionary objects that reference a certain Data Dictionary object.

Related documentation: For details refer to [tlFindDDReferences \(\[TargetLink API Reference\]\(#\)\)](#).

Hook Functions

New Hook Functions

Comparing Code Generator options

There is now a new `tl_pre_compare_creator_options_hook.sam` hook function template. It allows you to customize the comparison of Code Generator options that is performed during the build process.

For details, refer to [tl_pre_compare_creator_options_hook](#) ( TargetLink File Reference).

New Features of TargetLink 3.4

Where to go from here

Information in this section

New Production Code Generation Features	183
New AUTOSAR-Related Features	197
New TargetLink Data Dictionary Features	202

New Production Code Generation Features

Where to go from here

Information in this section

Component-Based Development Using Abstract Interfaces.....	183
Improvement for Custom Code.....	184
Optional Deactivation of Compute-Through-Overflow.....	184
Generating Virtual ECUs for Virtual ECU Testing.....	185
Specifications of the Target Simulation Modules.....	186
Code Generator Options.....	189
New API Commands.....	194
New Hook Functions.....	194
Access Function Changes.....	195
Improvement of Code Efficiency.....	195
General Enhancements and Changes.....	196

Component-Based Development Using Abstract Interfaces

Separating functionality to software modules

Support of component-based development allows you to separate the overall functionality into smaller software modules, called [modular units](#) which can be developed and tested independently. A separated modular unit can be tested with full test coverage more easily. Testing with full coverage is often not possible if the functionality is integrated in a larger context and therefore not all the test vectors can easily be applied directly at the modular unit. Thus the testability of modular units is generally higher. Later you only have to test the interaction between the modular units in the integration model for covering the overall functionality. Modular units also help you solve problems with long code generation times and high memory consumption during code generation for very large models, and can easily be reused without modification across different projects and in different contexts ([integration model](#)). This is an advantage if you want to change certain aspects of modular unit implementation: e.g., when you make a parameter of a modular unit a constant variable in one ECU project and a calibratable variable in another project. TargetLink provides [abstract interfaces](#) to support such changes without modifying the modular unit itself. You can map abstract interfaces to concrete implementations independently and after production code has been generated and tested for the modular unit (and should therefore not be changed).

For further information, refer to

- [Basics on Component-Based Development Using Abstract Interfaces \(RDI Objects\)](#) ([TargetLink Customization and Optimization Guide](#))
- Demo model: [REPLACEABLE_DATA_ITEMS](#) ([TargetLink Demo Models](#))
- [Incremental code generation for AUTOSAR SWCs](#) on page 198

Improvement for Custom Code

Enhancing unsupported Simulink blocks to TargetLink

Any unsupported Simulink block type (including subsystems) can be enhanced to a TargetLink Custom Code (type II) block. Type II means that no S-function is generated from the custom code template, but the original MIL simulation behavior of the enhanced Simulink block is kept.

For details, refer to [Details on Masking Simulink Blocks with TargetLink Custom Code \(Type II\) Blocks](#) ([TargetLink Preparation and Simulation Guide](#)).

Width-invariant custom code template (type II)

A width-invariant custom code template can be (re)used in multiple Custom Code (type II) block instances of varying width contexts. The width context relates to the block's input and output signals, and its states. Width invariance is achieved by using width macros in the custom code template. The width macros are resolved to the actual instance-specific widths at code generation time.

For details, refer to [Basics on Using Width Macros in Custom Code \(Type II\) Block](#) ([TargetLink Preparation and Simulation Guide](#)).

Demo model

TargetLink provides a demo model, refer to [CUSTOM_ENHANCEMENT](#) ([TargetLink Demo Models](#)).

Optional Deactivation of Compute-Through-Overflow

Controlling the use of CTO patterns

By default, TargetLink uses compute-through-overflow (CTO) code patterns to improve code efficiency.

Optionally, you can configure TargetLink to

- *Never implement CTO patterns*
- *Always apply CTO code patterns whenever an overflow can occur*
- *Optimized use CTO patterns to improve the code efficiency*

This allows you to adapt TargetLink to your company's coding style.

New Fixed-Point Library header file The new header file `sumprot.h` was added to the Fixed-Point Library, used by TargetLink to avoid certain 64-bit calculations when configured to never implement CTO patterns.

For further information, refer to

- [Code Generator Options](#) on page 189
(The new Code Generator options.)
- Resulting production code changes in comparison to the code generated by previous TargetLink versions (refer to [Code Changes between TargetLink 3.3 and TargetLink 3.4](#) on page 551)
- [Compute-Through-Overflows Property of Two's Complement Arithmetic](#) ([TargetLink Preparation and Simulation Guide](#))
- [Macros and Functions Provided by the Fixed-Point Library](#) ([TargetLink File Reference](#))

Generating Virtual ECUs for Virtual ECU Testing

V-ECU Generation

TargetLink lets you generate a V-ECU for a TargetLink subsystem and use it in virtual ECU testing (VET). V-ECUs emulate real ECUs in offline simulation scenarios.

You can download the V-ECUs to VEOS, dSPACE's offline simulator. This allows seamless integration into the dSPACE tool chain and lets you automate tests with AutomationDesk or run experiments in ControlDesk. This allows early testing and validation throughout the ECU development process.

V-ECU Manager TargetLink's V-ECU Manager assists you in configuring and performing the V-ECU build process.

API commands New API commands allow you to integrate TargetLink's V-ECU generation capability in your workflow.

For further information, refer to

- [TargetLink V-ECU Manager](#) ([TargetLink Tool and Utility Reference](#))
- [New API Commands](#) on page 194

Specifications of the Target Simulation Modules

Supported evaluation boards, microcontrollers, and compilers

The following table shows the combinations of evaluation boards, microcontrollers, and compilers supported by TargetLink 3.4 (TargetLink abbreviations).

Evaluation Board	Microcontroller Type	Compiler ¹⁾
Freescale HCS12EVB ²⁾	Freescale MC9S12DP256	Cosmic 4.7
Freescale HCS12EVB ²⁾	Freescale MC9S12DP256	Cosmic 4.8
Freescale HCS12EVB ²⁾	Freescale MC9S12DP256	Metrowerks CodeWarrior 3.1
Freescale HCS12EVB ²⁾	Freescale MC9S12DP256	Metrowerks CodeWarrior 5.1
MCT HCS12 T-Board (DP256) ²⁾	Freescale MC9S12DP256	Cosmic 4.7
MCT HCS12 T-Board (DP256) ²⁾	Freescale MC9S12DP256	Cosmic 4.8
MCT HCS12 T-Board (DP256) ²⁾	Freescale MC9S12DP256	Metrowerks CodeWarrior 3.1
MCT HCS12 T-Board (DP256) ²⁾	Freescale MC9S12DP256	Metrowerks CodeWarrior 5.1
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	Cosmic 4.7
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	Cosmic 4.8
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	Metrowerks CodeWarrior 3.1
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	Metrowerks CodeWarrior 5.1
Freescale 56F8367 Evaluation Module	Freescale MC56F8367	Metrowerks CodeWarrior 8.3
Axiom CMD-0565	Freescale MPC565	Wind River Diab 5.7
Axiom CME-0555	Freescale MPC555	Green Hills 5.1
Axiom CME-0555	Freescale MPC555	Wind River Diab 5.7
Axiom MPC5554DEMO	Freescale MPC5554	GNU 4.1
Axiom MPC5554DEMO	Freescale MPC5554	No longer supported Green Hills 5.2
Axiom MPC5554DEMO	Freescale MPC5554	New Green Hills 2012.1
Axiom MPC5554DEMO	Freescale MPC5554	Metrowerks CodeWarrior 2.8
Axiom MPC5554DEMO	Freescale MPC5554	Wind River Diab 5.7
Axiom MPC5554DEMO	Freescale MPC5554	Wind River Diab 5.9
Freescale MPC5561EVB	Freescale MPC5561	No longer supported Green Hills 5.2
Freescale MPC5561EVB	Freescale MPC5561	New Green Hills 2012.1
Freescale MPC5561EVB	Freescale MPC5561	Metrowerks CodeWarrior 2.8
Freescale MPC5561EVB	Freescale MPC5561	Wind River Diab 5.9
Freescale MPC5561EVB USB	Freescale MPC5561	Metrowerks CodeWarrior 2.8
Freescale MPC5561EVB USB	Freescale MPC5561	Wind River Diab 5.9
Freescale MPC5561EVB USB	Freescale MPC5561	No longer supported Green Hills 5.2
Freescale MPC5561EVB USB	Freescale MPC5561	New Green Hills 2012.1
Freescale MPC5604BEVB	Freescale MPC5604B	Green Hills 5.2

Evaluation Board	Microcontroller Type	Compiler ¹⁾
Freescale MPC5604BEVB	Freescale MPC5604B	New Green Hills 2012.1
Freescale MPC5604BEVB	Freescale MPC5604B	Wind River Diab 5.9
MCT S12X T-Board ²⁾	Freescale MC9S12XDP512	Cosmic 4.8
MCT S12X T-Board ²⁾	Freescale MC9S12XDP512	Metrowerks CodeWarrior 5.1
MCT S12X T-Board USB ²⁾	Freescale MC9S12XDP512	Cosmic 4.8
MCT S12X T-Board USB ²⁾	Freescale MC9S12XDP512	Metrowerks CodeWarrior 5.1
I+ME Promotion Package 166	Infineon c167	TASKING C166/ST10 Toolchain 8.6
I+ME Promotion Package 166	Infineon c167	TASKING C166/ST10 Toolchain 8.7
Infineon TriBoard TriCore 1766	Infineon TC1766	TASKING TriCore VX-Toolset 3.2
Infineon TriBoard TriCore 1766	Infineon TC1766	No longer supported TASKING TriCore VX-Toolset 3.5
Infineon TriBoard TriCore 1766	Infineon TC1766	New TASKING TriCore VX-Toolset 4.0
Infineon TriBoard TriCore 1766 20 MHz	Infineon TC1766	TASKING TriCore VX-Toolset 3.2
Infineon TriBoard TriCore 1766 20 MHz	Infineon TC1766	No longer supported TASKING TriCore VX-Toolset 3.5
Infineon TriBoard TriCore 1766 20 MHz	Infineon TC1766	New TASKING TriCore VX-Toolset 4.0
Infineon TriBoard TriCore 1767	Infineon TC1767	TASKING TriCore VX-Toolset 3.2
Infineon TriBoard TriCore 1767	Infineon TC1767	No longer supported TASKING TriCore VX-Toolset 3.5
Infineon TriBoard TriCore 1767	Infineon TC1767	New TASKING TriCore VX-Toolset 4.0
Infineon TriBoard TriCore 1796	Infineon TC1796	HighTec GNU 3.4
Infineon TriBoard TriCore 1796	Infineon TC1796	TASKING TriCore VX-Toolset 3.2
Infineon TriBoard TriCore 1796	Infineon TC1796	No longer supported TASKING TriCore VX-Toolset 3.5
Infineon TriBoard TriCore 1796	Infineon TC1796	New TASKING TriCore VX-Toolset 4.0
Infineon EasyKit XC2287	Infineon XC2287	TASKING VX-toolset for C166 3.0
Renesas M3A-2154	Renesas M32192	Gaio 11
Renesas M3A-2154	Renesas M32192	Gaio 9
Renesas M3A-2154	Renesas M32192	Renesas 5.1
Renesas EVB7058	Renesas SH-2E/SH7058	Renesas 9.3

Evaluation Board	Microcontroller Type	Compiler ¹⁾
Renesas EVB7058	Renesas SH-2E/SH7058	Renesas 9.4
Renesas SH72513 System Development Kit	Renesas SH-2A-FPU/SH72513	Renesas 9.4
New Renesas AB_050_Fx4	New Renesas V850E2/FL4- μPD70F4012	New Green Hills 2012.1
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	New Green Hills 2012.1
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	No longer supported Green Hills 5.3
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	NEC 3.40

¹⁾ Compiler Suite Version Supported

²⁾ The board is no longer distributed by dSPACE but is still supported for downward compatibility

For detailed information on the evaluation boards supported by TargetLink, refer to  [Evaluation Board Reference](#).

Note

For further PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website.

Code Generator Options

New Code Generator options The following new Code Generator options are available with TargetLink 3.4:

Description	Explanation	Default
AllowStructAssignments		
Copy structs or substructs of identical type via a single assignment instead of assignments of all struct (substruct) components.	<p>The C language allows direct assignments of struct variables. This is equivalent to copying all struct members but can instead be implemented by a compiler as copying the storage area (e.g. via memcpy()). This Code Generator option gives TargetLink permission to replace component-wise assignments of structs or substructs by the assignment of whole structs (substructs). If the situation is not sufficiently clear, e.g., due to code optimization, TargetLink will generate the component-wise assignments instead.</p> <p>Example: Consider</p> <pre>typedef struct S_tag { T3 c; T4 d; } S; typedef struct T_tag { T1 a; T2 b; S c; }; T S1; T S2; ... S2.a = S1.a; S2.b = S1.b; S2.c.d = S1.c.d; S2.c.e = S1.c.e;</pre> <p>Here, the copy may be written instead as</p> <pre>S2.a = S1.a; S2.b = S1.b; S2.c = S1.c; or S2 = S1;</pre>	on

Description	Explanation	Default
AssumeOperationCallsHaveNoUnknownDataFlow		
<p>For optimization purposes, treat every AUTOSAR operation as if the Data Dictionary Operation object has the NoDataFlowWithOtherOperations property set (or if the underlying function has a function class with the SIDE_EFFECT_FREE Optimization property set).</p>	<p>In principle, the implementation of an interaction between AUTOSAR operation calls is unknown. For other functions, TargetLink allows a function's behavior to be specified via the function class Optimization property:</p> <ul style="list-style-type: none"> ▪ "There is no unknown data flow via global variables, apart from the interface defined in the model" (SIDE_EFFECT_FREE): Accesses to global variables and function calls accessing global variables can be moved past a call of the respective function. ▪ "There are no internal states, and the number of function calls along an execution path can be changed" (MOVABLE): The function call can be moved into a conditionally executed control flow branch or to the second operand of a logical AND or OR operation. <p>There is usually no way to specify a function class for operation calls, unless they are also server runnables. Instead, TargetLink offers the NoDataFlowWithOtherOperations and NoStatesOrSideEffects properties at the Data Dictionary Operation object. This option allows you to ensure that all operation calls can be treated as if the function class has the SIDE_EFFECT_FREE Optimization property set or the operation has the NoDataFlowWithOtherOperations property set, i.e., it overrides the NoDataFlowWithOtherOperations property of all operations. If the option is switched off, then the value of the NoDataFlowWithOtherOperations property of each operation is evaluated.</p>	on
ExploitComputeThroughOverflow		
<p>Specifies whether and when compute-through-overflow code is implemented.</p>	<p>TargetLink uses compute-through-overflow (CTO) code patterns to implement efficient additions and subtractions and especially to avoid 64-bit operation. Since CTO can introduce overflows by casting the operands of an addition/subtraction to an unsigned data type, code checking tools might emit warnings or errors for this kind of code.</p> <p>The possible values:</p> <ul style="list-style-type: none"> ▪ 1 = Never: No compute-through-overflow code pattern will be implemented. ▪ 2 = Optimized: Apply compute-through-overflow patterns to improve the code efficiency. ▪ 3 = Always: Apply compute-through-overflow style code patterns whenever an overflow can occur. 	2 = Optimized

Description	Explanation	Default
UtilizeValueEqualitySignalLineSplit		
Replace block output variables of blocks preceding signal line splits by a variable with identical value, e.g., the state variable of a subsequent Unit Delay block.	<p>If there is a signal line split after block B's output and the block code for one of the blocks with inputs driven by B's output leads to a plain value copy via a direct assignment to a variable of the same data type, then this option allows TargetLink to eliminate B's block output variable and replace it by the variable on the left-hand side of the assignment. This situation usually occurs if the output of B drives a TargetLink Outport block and is also used inside the system or if the output of B drives a Unit Delay block in addition to subsequent calculations. TargetLink optimizes the code pattern per se, as it may be generated in other constellations as well.</p> <p>Example:</p> <p>Without this optimization, you can find</p> <pre>Float64 Sa1_Switch; static Float64 X_Sa1_Unit_Delay1 = 0.; if (Sa1_InPort1 >= 0.) { Sa1_Switch = Sa1_InPort; } else { Sa1_Switch = (Sa1_InPort2 + X_Sa1_Unit_Delay1) * 0.5; } Sa1_OutPort1 = (Float64) sin(Sa1_Switch); Sa1_OutPort = Sa1_Switch; X_Sa1_Unit_Delay1 = Sa1_Switch;</pre> <p>The optimization can replace the switch output by the Unit Delay state or the output of Outport, i.e.:</p> <pre>static Float64 X_Sa1_Unit_Delay1 = 0.; if (Sa1_InPort1 >= 0.) { X_Sa1_Unit_Delay1 = Sa1_InPort; } else { X_Sa1_Unit_Delay1 = (Sa1_InPort2 + X_Sa1_Unit_Delay1) * 0.5; } Sa1_OutPort1 = (Float64) sin(X_Sa1_Unit_Delay1); Sa1_OutPort = X_Sa1_Unit_Delay1;</pre>	on

Description	Explanation	Default
UtilizeValueEqualityStructFunctionArguments		
<p>If a function has a parameter of pointer-to-struct type that is either an input or an output parameter, e.g., specified via TargetLink Bus InPort or OutPort blocks, and if the argument that is passed is the address operator applied to a struct variable, then for (if the option is activated) TargetLink is allowed to eliminate this struct variable if a complete copy from (or to) another struct takes place before (or after) the function call.</p>	<p>Note that this is effectively an elimination of all struct components that have to be assigned beforehand, so TargetLink considers the non-struct-type struct components of the struct variable and all the (recursively) contained substruct components, the “leaf components”, and can only perform the optimization if these components are eligible for the elimination of intermediate variables. Specifically, this means that the struct components</p>	

Description	Explanation	Default
	<p>have to have a default variable class or a user variable class with the ERASABLE Optimization property set.</p> <p>Examples:</p> <p>Without this optimization, you can find</p> <pre data-bbox="533 424 1076 667"><code>struct tag_T S1; struct tag_T S2; ... S2.a = S1.a; S2.sub.b = S1.sub.b; S2.sub.c = S1.sub.c; S2.d = S1.d; /* Pointer-to-struct parameter only used for input */ r = f(&S2);</code></pre> <p>or, if struct assignments are requested via the AllowStructAssignments Code Generator option,</p> <pre data-bbox="533 772 1076 931"><code>struct tag_T S1; struct tag_T S2; ... S2 = S1; /* Pointer-to-struct parameter only used for input */ r = f(&S2);</code></pre> <p>The optimization can replace S2, i.e.:</p> <pre data-bbox="533 1005 1076 1110"><code>struct tag_T S1; ... /* Pointer-to-struct parameter only used for input */ r = f(&S1);</code></pre> <p>Conversely, the TargetLink optimization</p> <pre data-bbox="533 1195 1081 1343"><code>struct tag_T S3; struct tag_T S4; ... /* Pointer-to-struct parameter only used for output */ g(in1, in2, &S3); S4 = S3;</code></pre> <p>can replace S3 if it is certain that all leaf components are assigned a new value in g(), i.e.:</p> <pre data-bbox="533 1448 1081 1554"><code>struct tag_T S4; ... /* Pointer-to-struct parameter only used for output */ g(in1, in2, &S4);</code></pre> <p>TargetLink needs to gather information about the accesses to the struct components inside the respective functions. This also counts against the memory limit set by the SideEffectFreeAnalysisThreshold option.</p>	

For reference information on all Code Generator options, refer to [Code Generator Options](#) ( [TargetLink Model Element Reference](#)).

Migration aspects of Code Generator options	<p>Migration aspects include</p> <ul style="list-style-type: none">▪ Obsolete Code Generator options▪ Changed Code Generator options▪ Recommended compatibility options for new Code Generator options to ensure the best possible downward compatibility of the generated production code <p>Refer to Migration Aspects Regarding Code Generator Options on page 366 for more information.</p>
--	---

New API Commands

V-ECU generation	<p>New API commands allow you to integrate TargetLink's virtual ECU (V-ECU) generation capability in your workflow:</p> <ul style="list-style-type: none">▪ tl_generate_vecu_implementation (To generate V-ECU implementations that comprises all files necessary for virtual ECU testing.)▪ tl_compile_vecu (To compile V-ECU implementations.)▪ tl_build_vecu (To generate and compile V-ECU implementations in one step.) <p>For further information, refer to</p> <ul style="list-style-type: none">▪ Generating Virtual ECUs for Virtual ECU Testing on page 185.▪ API Functions (TargetLink API Reference).
-------------------------	---

Related topics	References
	tl_generate_vecu_implementation (TargetLink API Reference)

New Hook Functions

AUTOSAR	<p>Frame model generation/update TargetLink provides new hook functions for the AUTOSAR use case. You can use these hook functions to customize the frame model generation/update process.</p> <ul style="list-style-type: none">▪ tl_pre_add_comspecblock_hook.m▪ tl_post_add_comspecblock_hook.m
----------------	--

For further information, refer to

- [Improved frame model generation](#) on page 199
- [Hook Scripts](#) ( [TargetLink File Reference](#))

Access Function Changes

Improvements

TargetLink's access function mechanism was significantly improved. The following improvements were made:

Specifying access functions Optionally, TargetLink allows you to relate access functions to different variable kinds (scalar, vector, structure) by specifying the `VariableKindSpec` property of DD `AccessFunction` objects.

This allows you to

- Specify variable access functions for struct variables
- Specify separate or common access functions for different variable kinds within one access function template and thus also via one DD `VariableClass` object

Propagation into structures You can now configure TargetLink to propagate access function templates (AFTs) into structures by specifying the `PropagateStructToComponents` property of `DD /Pool/Templates/AccessFunctions/<AccessFunctionName>/Settings` objects.

This allows you to

- Specify an AFT applying to the leaf struct components (scalar or vector components) at the root struct variable
(`PropagateStructToComponents` property set to `on`).
- Access the root struct variable itself by access function (AF)
(`PropagateStructToComponents` property set to `off`).

Controlling use of auxiliary variables When specifying access function templates, you can now control whether TargetLink uses auxiliary variables to reduce the number of access function calls.

The specification is made at the DD `AccessFunction` objects's `CreateLocalValueCopy` property.

Improvement of Code Efficiency

Stronger code optimization by default

The TargetLink default settings of the Code Generator options that relate to code optimization have changed slightly. Production code generation leads to code that is even more optimized and readable, which is possible without burdening memory consumption and execution time in most cases.

Assignment of structs	TargetLink allows the assignment of structs as a whole: <pre>struct_B = struct_A;</pre> For details, refer to Basics on Optimizing Struct Variables (TargetLink Customization and Optimization Guide).
Elimination of struct function arguments	If a function has a parameter of pointer-to-struct type and certain conditions are met, then TargetLink can eliminate this struct variable if a complete copy from/to another struct takes place before/after the function call. For details, refer to Basics on Optimizing Struct Variables (TargetLink Customization and Optimization Guide).
Optimization of block outputs followed by signal line splits	TargetLink can eliminate superfluous block output variables if a block output is followed by signal line splits. For details, refer to Basics on Eliminating Temporary Variables (TargetLink Customization and Optimization Guide).
Scope reduction of structures	TargetLink can now reduce the scope of structures down to local. Because this implies setting the lifetime of the structure to auto, this optimization greatly reduces RAM consumption. For further information, refer to <ul style="list-style-type: none">▪ Scope of structured variables, in Obsolete Limitations on page 424▪ Basics on Optimizing Variables (TargetLink Customization and Optimization Guide)▪ Basics on the Scope Reduction of Variables (TargetLink Customization and Optimization Guide)

General Enhancements and Changes

Reduced memory consumption for logging	Basically, the memory consumption for logged simulation data has been optimized for all simulation modes (MIL, SIL and PIL simulation modes). If necessary, you can further reduce the memory consumption in MIL simulation mode. For details, refer to Performance of logging and memory consumption in MIL simulation mode (TargetLink Preparation and Simulation Guide).
Variable bit shifts in Stateflow	You can specify variable bit shifts in Stateflow. For example, the operation $x \ll n$ shifts the value of x left by n bits. Refer to Basics on Specifying Stateflow Models (TargetLink Preparation and Simulation Guide).

Signal specification for the Constant block	You can specify scaling parameters (Type, LSB, Offset) even though the default variable class is specified. Other blocks can then inherit these specified properties from a preceding Constant block. For details on the relevant block setting, refer to AllowSignalSpecification .
New Fixed-Point Library header file	The <code>sumprot.h</code> header file was added to the Fixed-Point Library to avoid certain 64-bit calculations when suppressing compute-through-overflow (CTO). For further information, refer to <ul style="list-style-type: none">▪ Macros and Functions Provided by the Fixed-Point Library ( TargetLink File Reference).
Support of Simulink SLX model format	TargetLink 3.4 supports the Simulink SLX model format.
Obsolete limitations	With TargetLink 3.4, several limitations have been removed. For details refer to Obsolete Limitations on page 424.

New AUTOSAR-Related Features

Features of the TargetLink AUTOSAR Module

Supported AUTOSAR Releases	AUTOSAR Release	Revision
4.0	4.0	4.0.3 ¹⁾
		4.0.2
3.2	3.2	3.2.2 ¹⁾
		3.2.1

AUTOSAR Release	Revision
3.1	3.1.5
	3.1.4
	3.1.2
	3.1.0
3.0	3.0.7
	3.0.6
	3.0.4
	3.0.2
2.1	2.1.4

¹⁾ New in TargetLink 3.4

Specifying AUTOSAR release in the TargetLink Data Dictionary

TargetLink allows you to generate AUTOSAR-compliant code for both AUTOSAR releases 3.x/4.x.

You can specify which AUTOSAR release to use in the `DD /Pool/AUTOSAR/Config` object the TargetLink Data Dictionary.

For information on generating AUTOSAR-compliant code, refer to [Generating Classic AUTOSAR-Compliant Code](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

Working with new AUTOSAR 4.x data type concept AUTOSAR 4.x introduced the distinction between application data types (ADTs) and implementation data types (IDTs).

TargetLink provides two wizards to assist you in the creation of ADTs and IDTs to provide a smooth transition to the AUTOSAR 4.x use case.

You can use the `ApplicationDataType Creation Wizard` to create ADTs for existing typedefs (IDTs) and the `ImplementationDataType Creation Wizard` to create IDTs for existing ADTs.

For information on working with ADTs/IDTs, refer to [Basics on Working with Implementation and Application Data Types \(Classic AUTOSAR\)](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

Incremental code generation for AUTOSAR SWCs

TargetLink provides several modeling features for component-based development. You can now also use these features for AUTOSAR models.

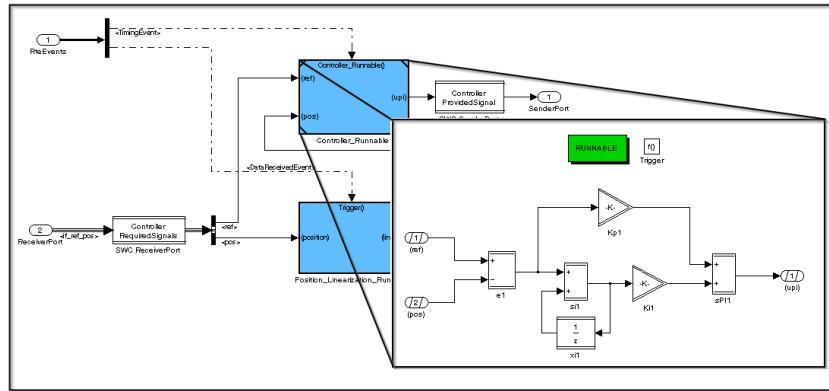
TargetLink allows you to place your AUTOSAR software components and runnables in separate code generation units (CGUs), i.e., referenced models or subsystems configured for incremental code generation.

Possible use cases: TargetLink 3.4 supports the following cases:

Feature: CGU Contains...	Purpose:	CGU Nesting: Can Contain CGUs of...
Type A: Functional part of runnable	To separate a functional part of a runnable in one CGU. Must not contain AUTOSAR blocks. Modeling of <i>calibratable parameters</i> and <i>per instance memories</i> is possible.	Type A
Type B: Complete runnable	To separate a complete runnable in one CGU. Can contain AUTOSAR blocks.	Type A
Type C: SWC containing at least one runnable	To separate a complete SWC in one CGU. Runnables have to be modeled in Simulink subsystems or TargetLink CGUs.	Type A
Type D: Several runnables	To separate several runnables in one CGU. The runnables can belong to different SWCs. The CGU does not need to contain all the runnables of one SWC. Runnables have to be modeled in Simulink subsystems or TargetLink CGUs.	Type A

Example The following screenshot shows a runnable containing a referenced model of type B.

Note that the referenced model contains AUTOSAR blocks. The Runnable block holds the parameters of the referenced model.



For further information, refer to

- [Component-Based Development Using Abstract Interfaces](#) on page 183
- [Partitioning Model and Code for Classic AUTOSAR](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).
- [Decomposing Models for Distributed Development](#) ([TargetLink Customization and Optimization Guide](#)).

Improved frame model generation

TargetLink's frame model generation feature allows you to generate TargetLink models with AUTOSAR blocks for Runnables and their interfaces from AUTOSAR files or AUTOSAR data contained in the Data Dictionary.

This feature has been further improved:

Update capability TargetLink's frame model generation now has an update capability that facilitates your round trips between TargetLink and software architecture tools such as SystemDesk.

You can now update existing frame models according to AUTOSAR files imported to the Data Dictionary. This enables you to quickly adapt your model to changes in the AUTOSAR files when implementing the software component in TargetLink.

TargetLink allows you to generate or update frame models for selected software components or groups of software components.

Update report When updating an existing model, TargetLink generates an HTML update report that gives you quick access to changed model parts via hyperlinks:

SwcModel Update Report			
Software component model update date: 20-Jun-2012			
AUTOSAR Element	DD AUTOSAR Object	TargetLink Block	Annotation
SoftwareComponent	FuelsysController	FuelsysController	
ReceiverPort		PP_RpConnectedSensors	For the block TL_Fuelsys/Subsystem/Subsystem/Subsystem/FuelsysController/PP_RpConnectedSensors corresponding receiver port object was not found. This block and its connection must be deleted.
ReceiverPort	NewReceiverPort	PP_NewReceiverPort	Block added.
SenderPort	PpFuelRate	PP_PpFuelRate	Block unchanged.
Runnable	Run_AirflowCalculation	Run_AirflowCalculation	Block unchanged.
Runnable	Run_AirflowCorrection	Run_AirflowCorrection	Block unchanged.
Runnable	Run_FuelRateCalcNormal	Run_FuelRateCalcNormal	Block unchanged.
Runnable	Run_FuelRateCalcRich	Run_FuelRateCalcRich	Block unchanged.
Runnable	Run_GenFuelRateCalc	Run_GenFuelRateCalc	Block unchanged.

TargetLink's SwcModel Update Report informs you about updated blocks and whether user action is required.

Further improvements TargetLink's frame model generator now supports

- Application errors
- Communication specifications
- Mode access points

For further information, refer to

- [New Hook Functions](#) on page 194
(Customizing the frame model generation/update process via hook functions.)
- [Generating/Updating a Frame Model from Classic AUTOSAR Data](#)
( [TargetLink Classic AUTOSAR Modeling Guide](#)).

Delivery of software components as object code

TargetLink now allows you to deliver software components as object code.

This means you can better protect your intellectual property and to deliver tested components.

You can specify the delivery format at the `DeliveryFormat` property of DD `SoftwareComponent` objects.

Compatibility mode By default, the software components delivered as object code are compliant to AUTOSAR *compatibility mode*, which results in functions with standardized names and data structures.

You can use an RTE generator from any vendor (provided that the RTE generator supports the compatibility mode).

This provides better interoperability with software architecture tools.

For further information, refer to

- [Exchanging Software Component Code \(TargetLink Interoperation and Exchange Guide\)](#).
- [Basics on Generated Code \(TargetLink Classic AUTOSAR Modeling Guide\)](#).

Improved handling of SIL/PIL simulations

The handling of SIL/PIL simulations for operation call subsystems has been improved.

TargetLink now generates simulation code if the **Use this subsystem for SIL/PIL simulation** checkbox is deactivated for *all* operation call subsystems contained in the model.

If the *same* operation call subsystem is *instantiated multiple times*, TargetLink chooses *one* instance.

Improved Container Manager

The Container Manager has been improved.

Exporting V-ECU implementations in SystemDesk 3.2 and TargetLink In SystemDesk and TargetLink, you can generate V-ECUs for use with VEOS.

You can export V-ECU implementations as containers. When you export a V-ECU implementation, a collection of files that belongs to the V-ECU implementation is gathered in a CTLGZ file archive.

Three kinds of files can be part of a V-ECU implementation:

- Configuration files (ARXML)
- Code files (H, C)
- A variable description file (A2L)

Exchanging SWCs between SystemDesk 4.0 and TargetLink You can exchange software components between SystemDesk 4.0 and TargetLink using containers.

You can use a container to transfer AUTOSAR files from SystemDesk 4.0 to TargetLink, either to start implementing a software component with TargetLink or to update an existing software component. After implementing or updating, you can export a container from TargetLink and import the relevant part of its contents in SystemDesk 4.0.

New TargetLink Data Dictionary Features

TargetLink Data Dictionary 3.4

The TargetLink Data Dictionary 3.4 (DD) has the following new features, enhancements and changes:

Where to go from here

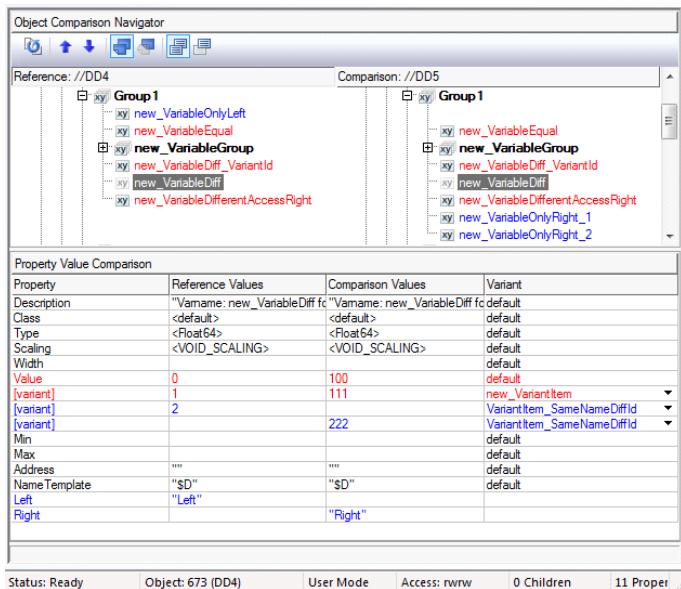
Information in this section

Compare and Merge Features.....	202
Improvements for Data Dictionary File Handling.....	203
Further Improvements of the Data Dictionary Manager.....	205
New DD MATLAB API Commands.....	206

Compare and Merge Features

Comparing and merging DD objects in the DD Comparison pane of the DD Manager

If you need to compare and merge DD objects, the DD Manager automatically opens a Comparison pane when you select two DD objects for comparison. It contains a synchronized tree structure of the compared DD objects including their child objects. You can compare and merge selected DD objects in one DD workspace, DD objects in different DD workspaces, or whole DD workspaces. The differences in the compared DD objects are indicated by color codes. This makes it easier for you to manage even the most complex DD object trees.



Benefits

- You can compare Data Dictionary workspaces and objects efficiently (inter- and intra-workspace comparisons).
- You can merge Data Dictionary subtrees, objects and properties efficiently using the synchronized tree view.
- You can customize the comparison, for example, by ignoring specific object properties.
- You can generate a report on the comparison result (XML or HTML). In a comparison pane, right-click anywhere to generate a report for the current comparison. As an alternative, you can start a comparison in a DOS command prompt: Use the `DsddComp.exe` command.

Related documentation:

- [Comparing and Merging Data Dictionaries](#) ( [TargetLink Data Dictionary Basic Concepts Guide](#))

Improvements for Data Dictionary File Handling

Simplified loading of partial Data Dictionary files by storing their original locations

The Open command can read metadata that is stored together with a partial DD file so that its content can easily be reloaded to its original location. You can also multiselect DD files and open them with this command.

If you open a DD file by using the Open command in the File menu of your TargetLink Data Dictionary Manager, the loading behavior is as follows:

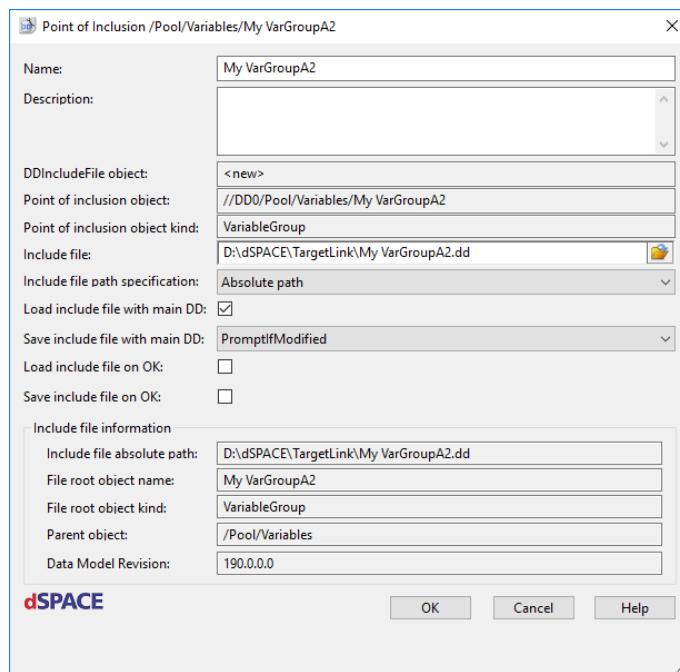
- If the file is a complete Data Dictionary, the active DD workspace is filled with the content of the DD file.
- If the file is a partial Data Dictionary with metadata about its original position, the Data Dictionary Manager automatically reads the original subtree and loads the file contents to that position.
- If the file is a partial Data Dictionary without metadata about its original position, the Data Dictionary Manager loads the content to a valid position according to the data model. If this is not possible, it is copied to a `/tmp` subtree. You can then move the objects to valid positions manually.

Note

Partial DD files saved with TargetLink versions earlier than TL 3.3 contain no metadata.

Simplified inclusion of partial DD files with the Point of Inclusion dialog

The new Point of Inclusion dialog makes it easier to handle partial DD files, which can be included at specific positions in the DD subtree. You can now create inclusion points for partial DD files directly by selecting an object tree and using the Point of Inclusion dialog.



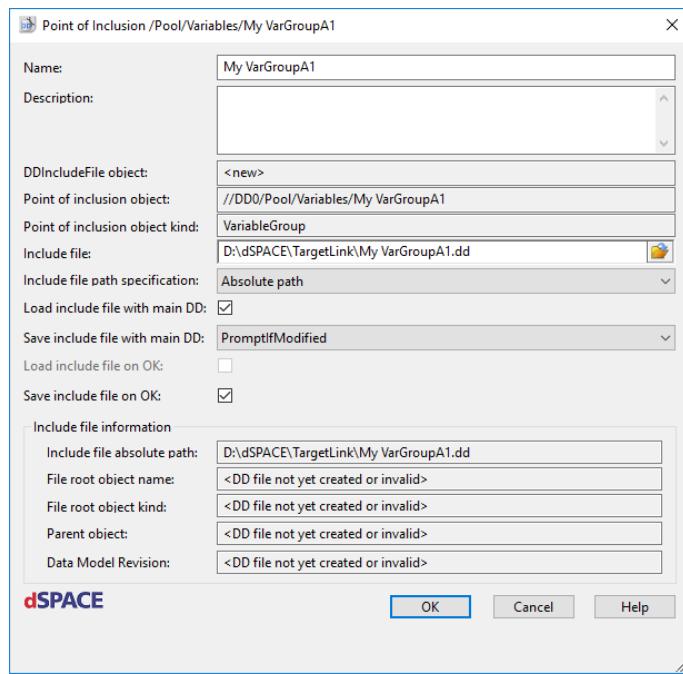
Benefits of the new Point of Inclusion dialog:

- Allows you to work with simplified file specification by using relative paths and paths relative to the main DD.
- Provides you information on the file content, for example, its position in the DD subtree.

For more information on how to include partial DD files, refer to [How to Include Partial Data Dictionary Files](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Simplified partition of a DD project file by separating partial DD files with the Point of Inclusion dialog

If you open a DD project file in your TargetLink Data Dictionary Manager, you can separate specific subtrees and save them as included DD files.



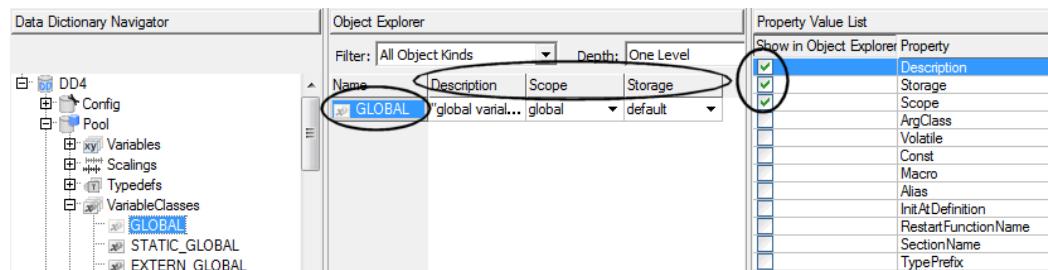
For more information on how to separate subtrees and save them as included DD files, refer to [How to Separate DD Subtrees and Save Them as Included DD Files](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Further Improvements of the Data Dictionary Manager

Object Explorer

Adding properties (columns) from the Property Value List to the Object Explorer has been improved.

In addition to the existing methods for adding properties to the Object Explorer, the TargetLink Data Dictionary Manager now offers an easier way: You can add them by selecting a DD object in the Object Explorer and then selecting the new checkboxes in the Property Value List.



For more information on adding properties to the Object Explorer, refer to [How to Specify Properties \(Columns\) Shown in the Object Explorer](#) ( [TargetLink Data Dictionary Basic Concepts Guide](#)).

Property Value List	Properties that cannot be set are now greyed out in the Property Value List. A tooltip informs you why the property cannot be set.
Loading DD files	You can now load the same DD file into different workspaces.
DD IncludeFileGroup objects	You can now group DD DDIncludeFile objects in DD DDIncludeFileGroup objects. Nesting is possible.

New DD MATLAB API Commands

Overview of new DD MATLAB API commands	<p>The following new DD MATLAB API commands are available:</p> <ul style="list-style-type: none"> ▪ <code>'Compare'</code> ▪ <code>'CreateComparisonReport'</code> ▪ <code>'Duplicate'</code> ▪ <code>'FindPrevious'</code> ▪ <code>'GetCodeFiles'</code> ▪ <code>'GetDDFiles'</code> ▪ <code>'GetFileAttributes'</code> ▪ <code>'GetPathToSLObject'</code> ▪ <code>'IsFileRoot'</code> <p><code>'Compare'</code> <code>dsdd('Compare',<objectIdentifier1>,<objectIdentifier2>[, attributeName1,attributeValue1,...])</code></p> <p>To compare two objects and to build a comparison tree.</p> <p><code>'CreateComparisonReport'</code> <code>dsdd('CreateComparisonReport',<objectIdentifier>[, attributeName1,attributeValue1,...]);</code></p> <p>To create an XML file that contains the results of an object comparison. The specified object must be a comparison tree root object. If the specified XML file exists, it is overwritten.</p> <p><code>'Duplicate'</code> <code>dsdd('Duplicate',<objectIdentifier>)</code></p> <p>To duplicate a DD object. The result must comply with the Data Model.</p> <p><code>'FindPrevious'</code> <code>dsdd('FindPrevious',<objectIdentifier>[, attributeName1,attributeValue1,...])</code></p> <p>Looks for an object which matches specified criteria. This command is like the 'FindNext' command except that the DD tree is traversed in reverse order. Use this command to look for objects iteratively in a loop.</p>
---	--

'GetCodeFiles' dsdd('GetCodeFiles' ,<objectIdentifier>)

To return names of generated files that are associated with an object that resides in the /Subsystems area after code generation.

'GetDDFiles' dsdd('GetDDFiles' [,<DD_identifier>]);

To return information about partial DD files that have been loaded into a workspace. These are files that have been loaded either by file inclusion or explicitly, for example, with the Load or the AutoLoad command.

'GetFileAttributes' dsdd('GetFileAttributes' ,<fileName>)

To retrieve attributes of a DD file.

dsdd('GetPathToSLObject'

[...]) dsdd('GetPathToSLObject' ,<objectIdentifier>[,attributeName1,attributeValue1,...])

To get the path to the Simulink system or SF object associated with a DD object that resides in the /Subsystems area after code generation.

dsdd('IsFileRoot' [...]) dsdd('IsFileRoot' ,<objectIdentifier>)

To check if a DD object is a file root object. Note that DD root objects are always file roots.

Related topics

References

dsddman ( TargetLink API Reference)

New Features of TargetLink 3.3

Where to go from here

Information in this section

New Production Code Generation Features..... 208

New AUTOSAR-Related Features..... 226

New dSPACE Data Dictionary Features..... 232

New Production Code Generation Features

Where to go from here

Information in this section

64-Bit Version of TargetLink.....	208
Function Reuse.....	209
Improvements for Incremental Code Generation and Model Referencing.....	210
Lifetime Analysis and Scope Reduction.....	210
Generic Multirate Code Generation.....	211
Using Data Store Memory Variables in Stateflow.....	212
Block Data Clipboard.....	212
Getting and Setting Block Properties via Structures.....	213
Simplified Referencing of Values from Data Dictionary.....	214
Native Autodoc Customization Block.....	214
BlackBox Mask Type.....	215
Enhancements to the Target Simulation Module.....	215
New TargetLink API commands.....	219
Code Generator Options.....	220
Improvement for Online Parameter Modification.....	221
Usability Improvements.....	222
General Enhancements and Changes.....	224
Testing Improvements.....	225

64-Bit Version of TargetLink

Handling large models

The 64-bit version of TargetLink allows you to address more than 4 gigabytes of memory, which helps you to handle even very large TargetLink models more efficiently. As is typical with 64-bit applications, this increases processing times (in this case for code generation) and memory consumption.

Function Reuse

Instance-specific entries as pointers

TargetLink's function reuse capability was significantly improved. As of TargetLink 3.3, you can add instance-specific entries not only directly as struct components but also as pointers to a generated reuse structure. This offers more freedom in implementation. For example, you can realize instance-specific variable names. Naming instance-specific variables with respect to the context in the model, e.g., predecessor and successor blocks, is also possible, refer to [Example of Instance-Specific Variable Naming via MATLAB Scripts \(Indirect Reuse\)](#) ([TargetLink Customization and Optimization Guide](#)).

Benefits and use-cases

- Parameters can be freely placed/accessed over the whole model including reused systems.
- User variables (extern,alias) and structure components can be used for reused system.
- Interface variables of instances can be merged with their predecessors-successors.

Code example The following table shows an example of generated code:

Indirect Reuse (new)	Direct Reuse
<pre>UInt16 myGain = 2; ISV_SIRP1_0_tp ISV_SIRP1_IR_Parameter1_0 = { &myGain /* pSIRP1_Gain1: */ }; Access: *(pISV->pSubStruct- >pSIRP1_Gain1)</pre>	<pre>ISV_SIRP1_0_tp ISV_SIRP1_IR_Parameter1_0 = { 2 /* myGain: */ }; Access: pISV->pSubStruct- >myGain</pre>

Explicit specification of instance-specific data

You can explicitly force a variable to reside in a reuse structure in the generated code by using new reuse-specific properties of the variable class in the dSPACE Data Dictionary. For details, refer to [How to Implement Instance-Specific Variables via Indirect Reuse](#) ([TargetLink Customization and Optimization Guide](#)).

Note

The explicit specification allows the implementation of instance-specific entries only as pointers in the reuse structure, not directly as struct components.

Related topics

Basics

[Reusing C Functions and Variables for Identical Model Parts \(Function Reuse\)](#)

( [TargetLink Customization and Optimization Guide](#))

Improvements for Incremental Code Generation and Model Referencing

Significantly reduced time/memory consumption

An internal change has significantly reduced both time and memory consumption for code generation of subsystems configured for incremental code generation.

Since they are also easier to handle, this makes subsystems configured for incremental code generation a viable alternative to using model referencing.

Note

From TargetLink 3.3 on, code for nested subsystems configured for incremental code generation has to be generated from the inside out.

More powerful model referencing

Model referencing is now more powerful. For example, you can now:

- Execute referenced models by means of edge-triggered ports
- Use implicit data types for interface variables
- Use scaling-invariant referenced models

Lifetime Analysis and Scope Reduction

Variables' lifetime analysis and rescheduling of state updates

TargetLink usually schedules the update of the values of state variables towards the end of the code of a function or of an atomic subsystem in order to calculate output variables at an early stage. If the calculations leading to the new state value cannot be moved to the state update, the respective intermediate variables have to hold their values from early in the function to rather late in the function: they have a long *lifetime*. This might lead to additional stack size or register blocking.

When the `ExtendedLifeTimeOptimization` option is enabled, TargetLink reschedules update statements towards the top of the function, near the calculation of the intermediate variables and the calculations involving the previous value of the updated variable. Apart from block state variables with a late update, variables that can be states (such as Data Store Memory block

variables) are also affected. This optimization can greatly influence the generated code, especially in conjunction with the `ExtendedVariableSharing` option.

The following table shows an example of code generated in TargetLink 3.3 as compared to earlier versions.

<= TargetLink 3.3	TargetLink 3.3
<pre>Float64 Aux_F64; Float64 Aux_F64_a; Float64 Aux_F64_b; static Float64 X_Unit_Delay = 0.; Aux_F64 = Sig1 + Sig2; Aux_F64_a = X_Unit_Delay + Aux_F64; RM = CalculateRM(Aux_F64_a); Aux_F64_b = (Aux_F64_a * 0.5) - PreDmp; SymDiff = Aux_F64_b; Dmp = (Aux_F64_b * 0.9) + (Corr * 0.1); X_Unit_Delay = Aux_F64;</pre>	<pre>Float64 Aux_F64; Float64 Aux_F64_a; static Float64 X_Unit_Delay = 0.; Aux_F64 = Sig1 + Sig2; Aux_F64_a = X_Unit_Delay + Aux_F64; X_Unit_Delay = Aux_F64; RM = CalculateRM(Aux_F64_a); Aux_F64 = (Aux_F64_a * 0.5) - PreDmp; SymDiff = Aux_F64; Dmp = (Aux_F64 * 0.9) + (Corr * 0.1);</pre>

For an overview of all code generator options refer to [Code Generator Options](#) ([TargetLink Model Element Reference](#)).

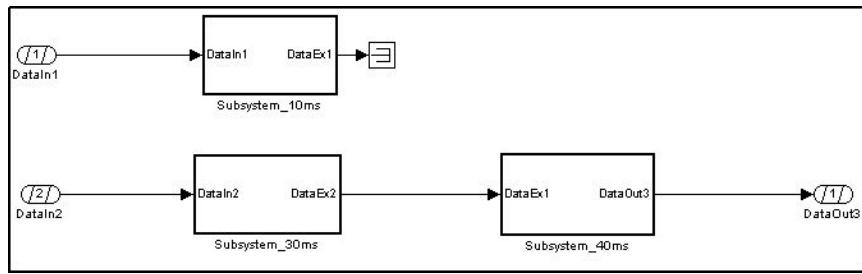
Generic Multirate Code Generation

Multirate code generation for standard and AUTOSAR models

With TargetLink 3.3, you can create multirate models containing subsystems with different sample times in Standard and AUTOSAR code generation modes. Earlier TargetLink versions support multirate code generation only in RTOS code generation mode, which maps your multirate model to a multitasking application when you generate code.

Unlike the RTOS approach with objects like tasks, events, interrupts, resources and alarm sets, the generic multirate approach does not create tasks and has no interrupts or intertask communication. Instead, different sample times are realized by downsampling in the generated code. While the RTOS code generation mode can be characterized as *multirate, multi task*, the Standard and AUTOSAR code generation modes can be characterized as *multirate, single task*.

The Code Generator generates a step function for each atomic subsystem. If an atomic subsystem contains a child atomic subsystem whose sample time is n -times the sample time of its parent subsystem, in the generated code the step function for the child subsystem is executed every n -th time inside the step function for the parent subsystem (downsampling).



For further information, refer to [Generic Multirate Code Generation](#)
([TargetLink Preparation and Simulation Guide](#)).

Using Data Store Memory Variables in Stateflow

Using data store memories in Stateflow

You can now use data store variables in Stateflow.

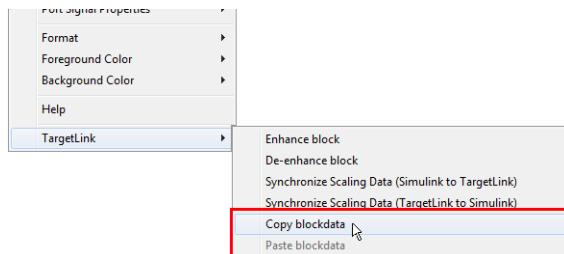
For more information refer to: [How to Use Data Store Memories with Stateflow](#)
([TargetLink Preparation and Simulation Guide](#)).

Block Data Clipboard

Copying block data via TargetLink's block data clipboard

You can use TargetLink's block data clipboard to transfer block data between blocks. Block properties that are not relevant for code generation (size, name, background color, etc.) are ignored.

Benefit You can quickly transfer block data from one block to another via the blocks' context menu, as shown below:



Note

With the context menu, you can copy block data only between *blocks of the same type*.

Note the following points:

- Strictly speaking, InPort/OutPort blocks and BusInport/BusOutport blocks are of the same type.
- However, when block data is copied from an InPort/OutPort block to a BusInport/BusOutport block, the BusInport/BusOutport block is automatically converted into a InPort/OutPort block and vice versa.

For more information refer to [Getting and Setting Block Properties via Structures](#) on page 213 and [Basics on Getting and Setting Block Properties](#) ([TargetLink Interoperation and Exchange Guide](#)).

Getting and Setting Block Properties via Structures

Getting and setting block properties via structures

You can now get and set block properties via structures.

Benefit: You can now transfer whole sets of block properties via the API by using the `tl_get` and `tl_set` commands which can now handle whole `blockData structs`.

Note

With the API, you can copy block data only between blocks that share the same properties.

Example: To change the data types at a busport:

```
% set type and lsb value for all signals in the bus
dataStruct = tl_get(hBusPort, 'BlockDataStruct');
for n = 1:dataStruct.numoutputs
    dataStruct.output(n).type = 'Int32';
    dataStruct.output(n).lsb = 2^-31;
end
[e, m] = tl_set(hBusPort, 'BlockDataStruct', dataStruct);
```

For more information refer to [Automating Tasks via TargetLink API](#) ([TargetLink Interoperation and Exchange Guide](#)), [Basics on Getting and Setting Block Properties](#) ([TargetLink Interoperation and Exchange Guide](#)), `tl_set`, `tl_get`, [Block Data Clipboard](#) on page 212.

Related topics

References

[tl_get](#) ([TargetLink API Reference](#))
[tl_set](#) ([TargetLink API Reference](#))

Simplified Referencing of Values from Data Dictionary

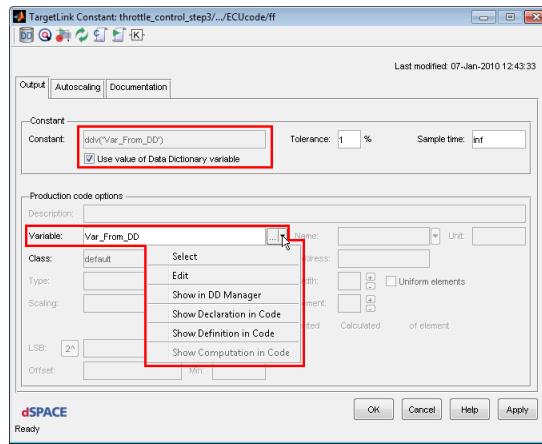
Simplified referencing of parameter values from the Data Dictionary

As of TargetLink 3.3 it is easier to read parameter values from the DD.

Simply select a variable on the Output pane of the block dialog and select the Use value of Data Dictionary variable checkbox.

Benefit This usability feature enables you to

- Manage parameter values better via the Data Dictionary
- Ensure consistency between model and DD
- Simplify variant coding



For more information refer to [How to Reference Values from DD Variable Objects to Block Variables](#) ([TargetLink Preparation and Simulation Guide](#)).

Native Autodoc Customization Block

Extending TargetLink's automatic documentation generation

With TargetLink 3.3 a *native* Autodoc Customization block is directly available in the TargetLink Block Library.

Benefits

- You can extend the automatic documentation by adding additional text, images, or hyperlinks.
- You can edit/search for your documentation settings via the Property Manager or via the API using the `tl_set`/`tl_get` commands.

Note

The old Autodoc Customization block is still supported. You do not have to take any steps to establish compatibility.

For more information refer to:

- Autodoc Customization Block ([TargetLink Model Element Reference](#))
- Automating Tasks via TargetLink API ([TargetLink Interoperation and Exchange Guide](#))
- Basics on Getting and Setting Block Properties ([TargetLink Interoperation and Exchange Guide](#))
- `tl_set` ([TargetLink API Reference](#))
- `tl_get` ([TargetLink API Reference](#))

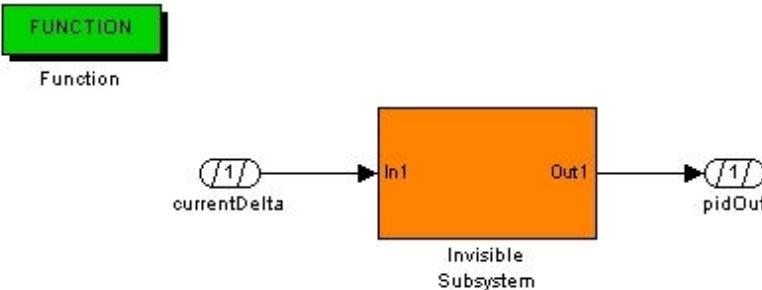
BlackBox Mask Type

Replacing specific subsystems by own implementations

The TL_BlackBox mask type allows you to exclude a subsystem from automatic code generation if you wish to provide your own implementation of the algorithm. To instruct the Code Generator to embed a function call in your generated code function, you have to model a function interface specification in the parent subsystem.

As a side effect, in MIL mode, you can even use unsupported Simulink blocks in the BlackBox subsystem. In SIL/PIL modes, a function call to an external function is executed instead.

The algorithm covered by the »Invisible Subsystem« is to be represented by a hand-coded function. The function block defines the function call to be inserted in the generated code. The interface variables are specified by:
* TargetLink input and output blocks
* Structure variables located in the Data Dictionary (refer to the Arguments page of the Function block dialog)
The subsystems MaskType property is set to TL_BlackBox so that the »Invisible Subsystem« is not analyzed by TargetLink's Code Generator.



Enhancements to the Target Simulation Module

(New) evaluation boards, microcontrollers, and compilers

The following table shows the combinations of evaluation boards, microcontrollers, and compilers supported by TargetLink 3.3 (TargetLink abbreviations). New evaluation boards, microcontrollers, and compiler versions

are underscored. Evaluation boards, microcontrollers, and compiler versions that are no longer supported are overscored.

Evaluation Board	Microcontroller Type	Compiler
Freescale HCS12EVB	Freescale MC9S12DP256	Cosmic 4.7
		Cosmic 4.8
		Metrowerks CodeWarrior 3.1
		Metrowerks CodeWarrior 5.0
		Metrowerks <u>CodeWarrior 5.1</u>
MCT HCS12 T-Board (DP256)	Freescale MC9S12DP256	Cosmic 4.7
		Cosmic 4.8
		Metrowerks CodeWarrior 3.1
		Metrowerks CodeWarrior 5.0
		Metrowerks <u>CodeWarrior 5.1</u>
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	Cosmic 4.7
		Cosmic 4.8
		Metrowerks CodeWarrior 3.1
		Metrowerks CodeWarrior 5.0
		Metrowerks <u>CodeWarrior 5.1</u>
Freescale 56F8367 Evaluation Module	Freescale MC56F8367	Metrowerks CodeWarrior 8.1
		<u>Metrowerks CodeWarrior 8.3</u>
Axiom CMD-0565	Freescale MPC565	Wind River Diab 5.5
		Wind River Diab 5.6
		Wind River Diab 5.7
Axiom CME-0555	Freescale MPC555	Green Hills 5.0
		Green Hills 5.1
		Wind River Diab 5.5
		Wind River Diab 5.6
		Wind River Diab 5.7
Axiom MPC5554DEMO	Freescale MPC5554	GNU 3.4

Evaluation Board	Microcontroller Type	Compiler
		GNU 4.1 Green Hills 5.0 Green Hills 5.1 <u>Green Hills 5.2</u> Metrowerks CodeWarrior 2.3 Metrowerks CodeWarrior 2.4 Metrowerks CodeWarrior 2.6 <u>Metrowerks CodeWarrior 2.8</u> Wind River Diab 5.5 Wind River Diab 5.6 Wind River Diab 5.7 <u>Wind River Diab 5.8</u> Wind River Diab 5.9
dSPACE DS1603	Freescale MPC5554	Microtec 3.2 Microtec 3.3 Microtec 3.5 Microtec 3.7 Wind River Diab 5.5 Wind River Diab 5.6 Wind River Diab 5.7 Wind River Diab 5.8
Freescale MPC5561EVB Freescale MPC5561EVB USB	Freescale MPC5561	Green Hills 5.0 Green Hills 5.1 <u>Green Hills 5.2</u> Metrowerks CodeWarrior 2.3 Metrowerks CodeWarrior 2.4 Metrowerks CodeWarrior 2.6 Metrowerks CodeWarrior 2.8 Wind River Diab 5.6 Wind River Diab 5.7 Wind River Diab 5.8 Wind River Diab 5.9
Freescale MPC5604BEVB	Freescale MPC5604B	Green Hills 5.1 <u>Green Hills 5.2</u> Wind River Diab 5.5 Wind River Diab 5.6 Wind River Diab 5.8 Wind River Diab 5.9
MCT S12X T-Board MCT S12X T-Board USB	Freescale MC9S12XDP512	Cosmic 4.7

Evaluation Board	Microcontroller Type	Compiler
		Cosmic 4.8 Metrowerks CodeWarrior 4.7 Metrowerks CodeWarrior 5.0 Metrowerks CodeWarrior 5.1
I+ME Promotion Package 166	Infineon c167	TASKING C166/ST10 Toolchain 8.6 TASKING C166/ST10 Toolchain 8.7
Infineon TriBoard TriCore 1766 Infineon TriBoard TriCore 1766 20 MHz	Infineon TC1766	TASKING TriCore VX-Toolset 2.5 TASKING TriCore VX-Toolset 3.2 TASKING TriCore VX-Toolset 3.4 TASKING TriCore VX-Toolset 3.5
Infineon TriBoard TriCore 1767	Infineon TC1767	TASKING TriCore VX-Toolset 2.5 TASKING TriCore VX-Toolset 3.2 TASKING TriCore VX-Toolset 3.4 TASKING TriCore VX-Toolset 3.5
Infineon TriBoard TriCore 1796	Infineon TC1796	HighTec GNU 3.3 HighTec GNU 3.4 TASKING TriCore VX-Toolset 2.5 TASKING TriCore VX-Toolset 3.2 TASKING TriCore VX-Toolset 3.4 TASKING TriCore VX-Toolset 3.5
Infineon EasyKit XC2287	Infineon XC2287	TASKING VX-toolset for C166 2.3 TASKING VX-toolset for C166 2.4 TASKING VX-toolset for C166 3.0
Renesas M3A-2154	Renesas M32192	Gaio 9 Gaio 10 Gaio 11 Renesas 4.3 Renesas 5.0 Renesas 5.1
Renesas EVB7058	Renesas SH-2E/SH7058	Renesas 9.0 Renesas 9.1 Renesas 9.3 Renesas 9.4
Renesas SH72513 System Development Kit	Renesas SH-2A-FPU/SH72513	Renesas 9.0 Renesas 9.1 Renesas 9.3 Renesas 9.4
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	Green Hills 5.0

Evaluation Board	Microcontroller Type	Compiler
		Green Hills 5.1
		Green Hills 5.3
		NEC 3.30
		NEC 3.40

For detailed information on the evaluation boards supported by TargetLink, refer to  [Evaluation Board Reference](#).

Note

For further PIL support combinations that are part of a valid Software Maintenance Service (SMS) contract, refer to dSPACE's [TargetLink PIL Support](#) website.

Discontinued board

No longer supported, still distributed The following board is no longer supported by TargetLink but still distributed by dSPACE:

- dSPACE DS1603 (Freescale MPC5554 microcontroller)

Note

To use the unsupported evaluation boards with TargetLink 3.3, please contact dSPACE.

New TargetLink API commands

API commands available as of TargetLink 3.3

In TargetLink 3.3, there are the following new TargetLink API commands.

- **tl_find_system**

Lets you search models or subsystems. It wraps the **find_system** API command and provides the **BlockDialogParams** search criteria for TargetLink block types.

Note

The **tl_find_system** command is deprecated since TargetLink 3.5. For details, refer to [Changes in TargetLink and TargetLink Data Dictionary API Functions](#) on page 359.

- **tl_access_logdata**

Lets you access logged simulation data. For details, refer to [tl_access_logdata](#) ( [TargetLink API Reference](#)).

- **tl_manage_blockset**
Provides access to information about TargetLink blocks/blocksets. For details, refer to [tl_manage_blockset](#) ([TargetLink API Reference](#)).
- **tl_repair_busdata**
Adapts a previously specified data set of TargetLink bus port blocks to the incoming bus signals. For details, refer to [tl_repair_busdata](#) ([TargetLink API Reference](#)).

Code Generator Options

New Code Generator options The following new Code Generator options are available with TargetLink 3.3:

Description	Explanation	Default Value
ExtendedLifeTimeOptimization		
Allows scheduling of code statements to reduce the lifetime of intermediate variables.	TargetLink usually schedules the update of the values of state variables towards the end of the code of a function or of an atomic subsystem in order to calculate output variables at an early stage. If the calculations leading to the new state value cannot be moved to the state update, the respective intermediate variables have to hold their values from early in the function to rather late in the function: they have a long <i>lifetime</i> . This might lead to additional stack size or register blocking. If this option is enabled, TargetLink reschedules update statements towards the top of the function, near the calculation of the intermediate variables and the calculations involving the previous value of the updated variable. Apart from block state variables with a late update, variables that can be states (such as Data Store Memory block variables) are also affected. This optimization can greatly influence the generated code, especially in conjunction with ExtendedVariableSharing.	on
GenerateFrameVariablesForAccessFunctions		
Generates auxiliary variables inside the simulation frame file for each production code variable accessed by access functions.	To provide access to possibly inaccessible variables (usually 'static global', i.e., module-local variables) that have <i>access functions</i> , TargetLink generates auxiliary variables inside the simulation frame file and creates access to the original variables through these auxiliary variables and the <i>access functions</i> during simulation.	off
Optimization		
Generates optimized production code.	If this option is enabled, the generated production code is optimized. The optimizations include: <ul style="list-style-type: none"> ▪ Elimination of intermediate variables ▪ Algebraic simplifications ▪ Scope reduction ▪ Control flow simplifications ▪ Moving code into conditionally executed control flow branches ▪ Dead code elimination Other options activate or influence single or groups of optimizations.	on

Description	Explanation	Default Value
RebuildStateflow		
Rebuilds Stateflow charts if necessary to update information about chart objects.	If this option is on, a Simulink/Stateflow rebuild of the chart is triggered to update the information in the Simulink/Stateflow API if necessary. If this option is off, no rebuild is triggered and under some circumstances not all the necessary information can be collected for TargetLink code generation. If this happens, warnings or errors might occur during TargetLink code generation.	off
SameIndirectReusePointerForSameDestination		
Generates only one indirect reuse pointer if the same variable is referenced from several places in the reused system.	If this option is off (default), a different pointer is generated for every block variable (output, parameter, etc.) specified for indirect reuse even if the pointer points to the same (merged) variable. If this option is on, only different pointer destination variables lead to different pointers.	off

Migration aspects regarding Code Generator options

Migration aspects include

- Obsolete Code Generator options
- Changed Code Generator options
- Recommended compatibility options for new Code Generator options to ensure best possible backward compatibility

Refer to [Migration Aspects Regarding Code Generator Options](#) on page 370 for more information.

Related topics

References

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

Improvement for Online Parameter Modification

Modifying calibration parameters in combination with access functions

TargetLink lets you modify the calibration parameter values of a simulation application in SIL and PIL simulation modes. You can modify parameter values online, i.e., without having to regenerate code, so that you can perform the simulation with parameter values different from those specified in the model.

As of TargetLink 3.3, you can modify calibration parameters (global, static or extern global, or macros) that are protected by read or write access functions. During code generation, TargetLink either calls the corresponding read or write access functions, or *does not use* access functions but handles the calibration parameters as unprotected parameters, depending on whether the `GenerateFrameVariablesForAccessFunctions` Code Generator option is set to on or off and on whether parameters are protected by access functions. For

further information, refer to [Basics on Modifying Parameter Values for Simulation](#) ([TargetLink Preparation and Simulation Guide](#)).

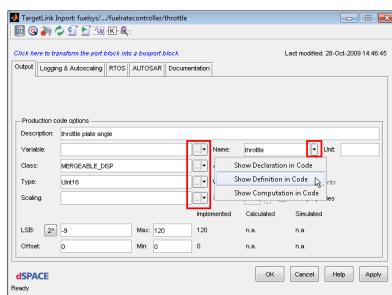
Usability Improvements

Better hyperlink functionality

TargetLink now provides improved hyperlink functionality.

Benefit This usability feature makes it easier to navigate to

- The Data Dictionary
- A declaration, definition, or computation in code



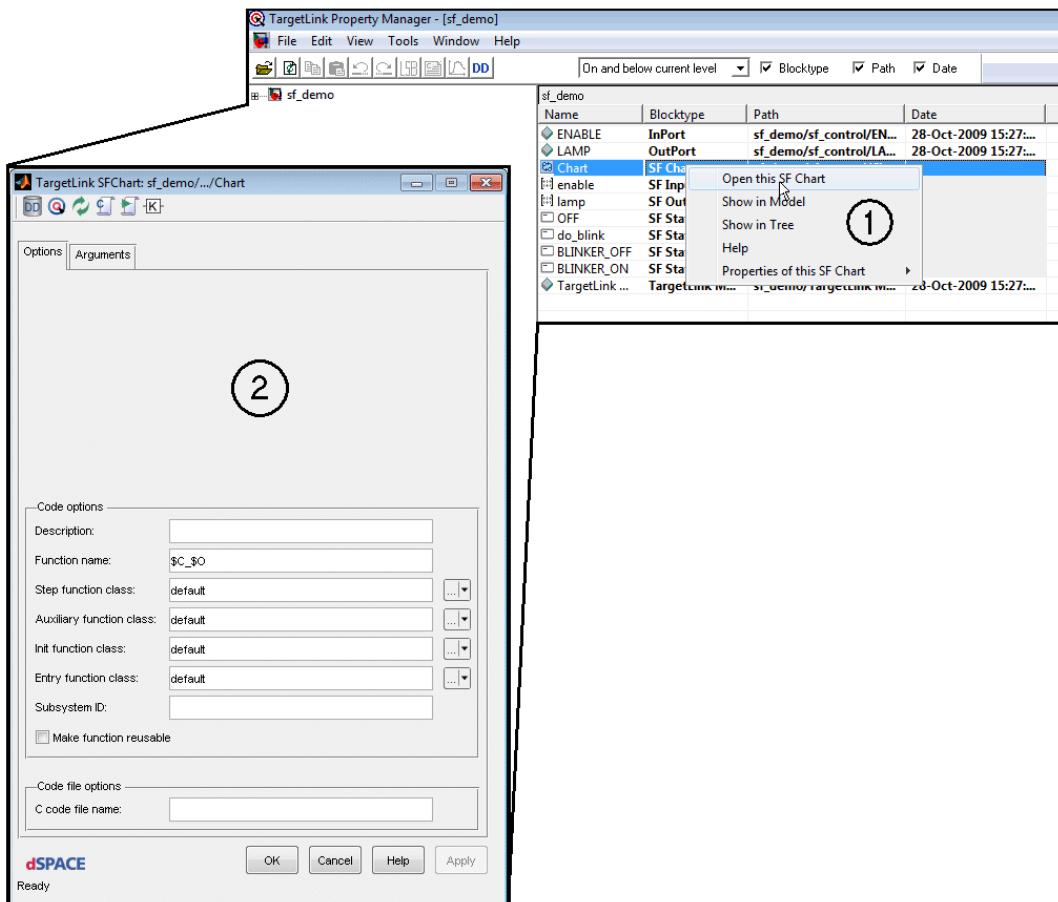
For more information, refer to [Common TargetLink Context Menu Options](#) ([TargetLink Model Element Reference](#)), [How to Trace Variables from the Model to the Code](#) ([TargetLink Preparation and Simulation Guide](#)), and [How to Reference Values from DD Variable Objects to Block Variables](#) ([TargetLink Preparation and Simulation Guide](#)).

Dedicated block dialogs for Stateflow charts

TargetLink 3.3 simplifies access to Stateflow charts by providing dedicated dialogs, similar to TargetLink's Function Block dialog.

Benefit This usability feature enables you to

- Access the Stateflow chart's dialog via the Stateflow chart's context menu in the Property Manager
- Edit code generation settings for Stateflow chart functions more directly

**Note**

The syntax of the name contained in the Function name field differs from the syntax of names contained in Function name fields of TargetLink's Function block. This is consistent with the naming scheme for Stateflow objects in general.

For more information refer to [Basics on Using Stateflow in TargetLink](#) ([TargetLink Preparation and Simulation Guide](#)), [Working with Stateflow](#) ([TargetLink Preparation and Simulation Guide](#)), [Starting with Stateflow in TargetLink](#) ([TargetLink Preparation and Simulation Guide](#)), [Basics on Using Name Macros](#) ([TargetLink Customization and Optimization Guide](#)), [Modifying Multiple Properties at Once via the Property Manager](#) ([TargetLink Preparation and Simulation Guide](#)).

Local block-context in \$E macros

Since TargetLink 3.3, instance-specific naming of variables by evaluating the \$E name macro is possible without using masks.

Information about the context of a block or object can now be used to evaluate the \$E name macro. This information is accessible from the MATLAB workspace by using the \$THIS macro.

Benefit:

- Easier handling, because the library systems no longer have to be masked
- Number of variables does not increase in the context of nested systems

For further information and an example refer to [Example of Instance-Specific Variable Naming via MATLAB Scripts \(Indirect Reuse\)](#) ( [TargetLink Customization and Optimization Guide](#)).

General Enhancements and Changes

Stand-Alone Model Manager	The Stand-Alone Model Manager is no longer contained in the Sample blocks section of the TargetLink Block Library. It can now be reached via the Tools page of the TargetLink Main Dialog .
----------------------------------	--

Shift Arithmetic block	The Shift Arithmetic block provides four new options (External source, Direction, Diagnostic for out-of-range shift value (MIL mode) and Protect against out-of-range shift value (production code)) that let you specify the shift arithmetics. For details, refer to Shift Arithmetic Block ( TargetLink Model Element Reference).
-------------------------------	--

Look-Up Tables (1D and 2D)	The Use Input Nearest and Use Input Above look-up methods have been replaced by the Use Input Below look-up method. For details, refer to Various Migration Aspects on page 372.
-----------------------------------	--

Parallel installation of 32-bit and 64-bit versions of MATLAB	On a 64-bit operating system, you can install both a 32-bit and a 64-bit version of a particular MATLAB release, e.g., R2009a. However, MATLAB release versions share the same preference settings folder, which means that you have to set all architecture-dependent settings such as the MEX compiler settings each time you switch between different versions of one MATLAB release.
--	---

Obsolete limitations	With TargetLink 3.3, several limitations have become obsolete. For details refer to Obsolete Limitations on page 427.
-----------------------------	---

Related topics

Basics

[Changes in TargetLink and dSPACE Data Dictionary API Functions](#)..... 368

Testing Improvements

M-API to access the TargetLink data server

You can use the **tl_access_logdata** API command to access the TargetLink data server.

Benefit: For example, you can

- Retrieve, load, or save simulation data from simulation runs in MIL/SIL/PIL mode
- Get information (model, time, start time, stop time, simulation lock, TargetLink subsystems) about simulations
- Display logged simulation data in a TargetLink plot overview window

Including precompiled modules into the build process

You now can include precompiled modules or additional libraries (OBJ and LIB files) in the build process.

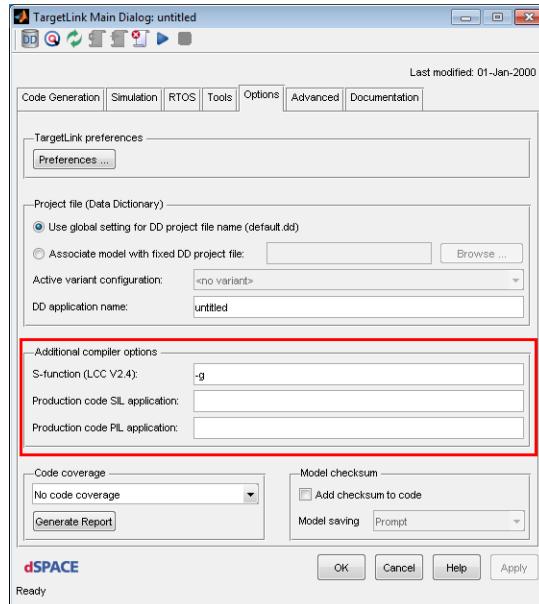
This functionality is provided via the hook functions
tl_pre_compile_host_hook.m for SIL simulations and
tl_pre_compile_target_hook.m for PIL simulations:

For further information refer to **tl_pre_compile_host_hook** (TargetLink File Reference), **tl_pre_compile_target_hook** (TargetLink File Reference).

Separate option sets for compiling production code and compiling S-functions

TargetLink now provides additional compiler options for you to specify separate option sets for compiling production code (SIL/PIL) and for compiling S-functions.

You can access these additional options by opening TargetLink's MainDialog and navigating to the Options page, as shown below:



Compiling and linking user files by providing file specifications

You can now compile and link user files (header files or source files) by providing file specifications in DD module objects.

Benefit This new feature allows you to

- Compile and link user files without using the AddFile block
- Provide user files independently of your model

For further information refer to [How to Build and Link External Modules via File Specifications](#) ([TargetLink Customization and Optimization Guide](#)).

Related topics

References

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

New AUTOSAR-Related Features

Features of the TargetLink AUTOSAR Module

Supported AUTOSAR Releases

Release	Version
4.0	4.0.2 ¹⁾
3.2	3.2.1 ²⁾
3.1	3.1.5 ²⁾
	3.1.4
	3.1.2
	3.1.0
3.0	3.0.7 ²⁾
	3.0.6
	3.0.4
	3.0.2
2.1	2.1.4

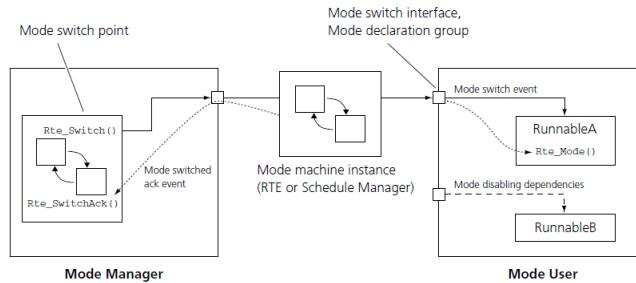
¹⁾ Available via the AUTOSAR 4.0 patch for TargetLink 3.3.

Contact dSPACE Support
(www.dspace.com/go/supportrequest) for further information.

²⁾ New in TargetLink 3.3

Support of mode management RTE functions

You can now generate code for the `Rte_Mode` and `Rte_Switch` RTE API functions.

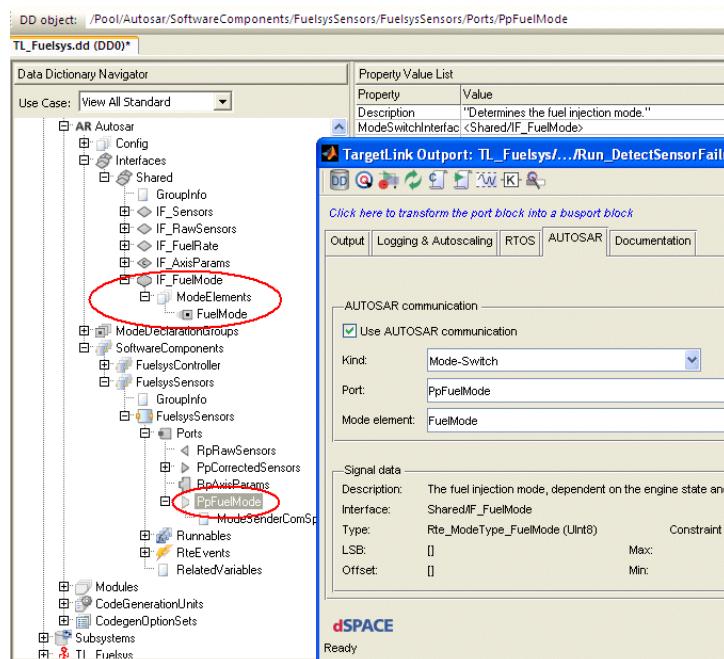


As you can see in the above illustration, a mode-managing software component requests a mode switch by the `Rte_Switch` API function. Mode-using software components read the currently active mode by the `Rte_Mode` API function.

In TargetLink, you can model modes similarly to sender-receiver communication. To generate the `Rte_Switch` API function, perform these steps:

1. Create a mode switch interface with a mode element in the Data Dictionary.
2. Create a mode sender port in the Data Dictionary and reference the mode switch interface.
3. Select the port and the mode element at an Outport block.

The illustration below shows an example.



For additional information on modeling modes, refer to [Modeling Modes](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

Enhanced SWC container exchange between TargetLink and SystemDesk

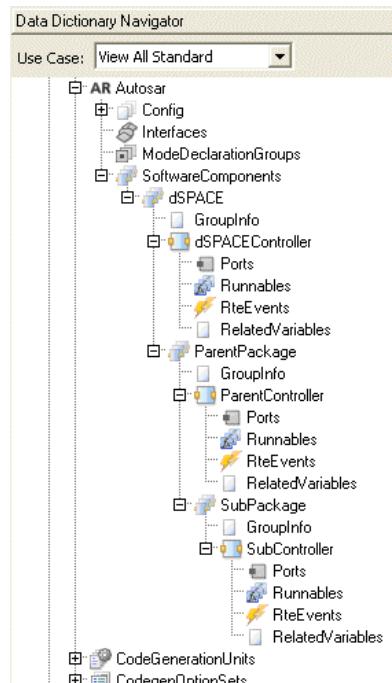
Enhanced container round trip between SystemDesk and TargetLink In the new version of the Data Dictionary, the container round trip between TargetLink and SystemDesk has been enhanced. You can now import SWC containers to a temporary DD workspace pane and diff/merge DD objects by using Data Dictionary functionalities.

Improved container export from TargetLink You can now export containers from TargetLink with full control of the container files. This allows improved AUTOSAR migration scenarios and bottom-up ECU software development workflows.

Improved import/export of AUTOSAR packages

The Data Dictionary now imports AUTOSAR files with package information according to their Package property, independently of the names of corresponding DD group objects. A package's full hierarchy, e.g., /dSPACE/ParentPackage/Subpackage, is taken into account during import.

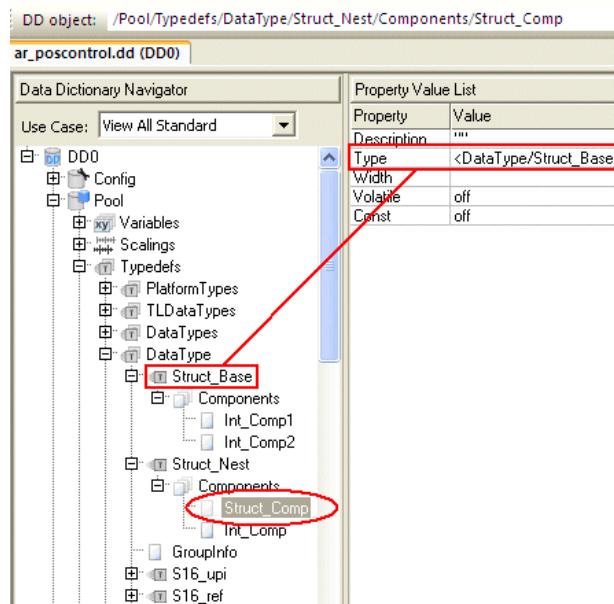
If a package does not exist in the Data Dictionary, but part of the package's hierarchy exists, the package tree is expanded under that part. For example, if a dSPACE package exists in the Data Dictionary, DD group objects for the ParentPackage and Subpackage packages are created under it. The following illustration shows controller atomic software components in a /dSPACE/ParentPackage/SubPackage hierarchy.



For additional information on importing/exporting package information, refer to [Basics on Importing AUTOSAR Files](#) ([TargetLink Interoperation and Exchange Guide](#)) and [Basics on Exporting AUTOSAR Files](#) ([TargetLink Interoperation and Exchange Guide](#)).

Support of nested structures

TargetLink now supports using structs for the components of a struct at an AUTOSAR interface. You can now use data types such as shown in the following illustration for a data element or operation argument.



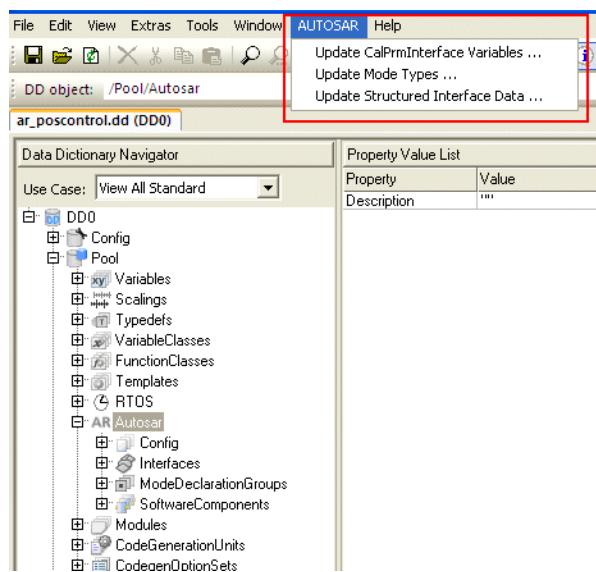
Improved modeling of getter/setter operations with struct-based operation arguments

You can model calls to get/set operations with TargetLink just by using an Inport/Outport block and connecting it with an implementation of the server operation, a signal generator, or a constant block.

With this version of TargetLink, you can additionally call get/set operations that have struct-based operation arguments.

Improved AUTOSAR modeling support in the Data Dictionary

This version of the Data Dictionary has an AUTOSAR menu for the AUTOSAR use case, which offers modeling support for AUTOSAR-related DD objects.



The AUTOSAR menu contains the following commands:

- Update CalPrmInterface Variables (continued)
- Update Mode Types (new)
- Update Structured Interface Data (new)

Update Mode Types You require a variable to represent the value of a mode declaration in your model if you want to connect a mode for communication. You can create these variables with the new Update Mode Types command from the Data Dictionary Manager's AUTOSAR menu.

Update Structured Interface Data You have to define operation arguments, data elements, and calprm elements that are consistent with their types. If you want to work with struct-based types, you can create consistent components of operation arguments, data elements, and calprm elements with the new Update Structured Interface Data command from the Data Dictionary Manager's AUTOSAR menu.

Specifying default options in the Data Dictionary

Import/export AUTOSAR files You can now specify your own default options for importing and exporting AUTOSAR files in the Data Dictionary. The Data Dictionary displays selected options for you. You can add further import/export options to specify their default values.

The following illustration shows an example.

DD object: /Pool/Autosar/Config/ImportExport
TL_Fuelsys.dd (DD0)

Data Dictionary Navigator

Use Case: View All Standard

- DDO
 - Config
 - Pool
 - Variables
 - Scalings
 - Typedefs
 - VariableClasses
 - FunctionClasses
 - Templates
 - RTOS
 - AR Autosar
 - Config
 - ImportExport
 - FrameModelGeneration
 - Interfaces
 - ModeDeclarationGroups
 - SoftwareComponents
 - Modules
 - CodeGenerationUnits
 - CodegenOptionSets
 - Subsystems
 - TL_Fuelsys

Property Value List

Property	Value
ForceFileOverwrite	"on"
Validate	"off"
Merge	"off"
ImportStrategy	"Merge"
ExportStrategy	"OneFileForAllPackages"
EnableOverwriteWarnings	"on"
SaveFileName	"off"
SetRteTypeModule	"on"
EnableDefaultContainerExportHook	"on"
EnablePackageSupport	"on"
Property	<n.a.>

Generating frame models with the `tl_generate_sw_c_model` command

You can now specify your own default options for generating a frame model in the Data Dictionary. The following illustration shows an example.

DD object: /Pool/Autosar/Config/FrameModelGeneration
TL_Fuelsys.dd (DD0)

Data Dictionary Navigator

Use Case: View All Standard

- DDO
 - Config
 - Pool
 - Variables
 - Scalings
 - Typedefs
 - VariableClasses
 - FunctionClasses
 - Templates
 - RTOS
 - AR Autosar
 - Config
 - ImportExport
 - FrameModelGeneration
 - Interfaces
 - ModeDeclarationGroups
 - SoftwareComponents
 - Modules
 - CodeGenerationUnits
 - CodegenOptionSets
 - Subsystems
 - TL_Fuelsys

Property Value List

Property	Value
UseCurrentDD	"off"
ForceOperationCallSubsystemUsage	"off"
TerminateInOutPorts	"on"
AddOperationCallTriggerPort	"off"
UseOneTbSubsystemForAllSwcs	"on"
GenerateStimuliSubsystems	"on"
MergeRunnableStimuli	"off"
GenerateStimuliLib	"off"
Property	<n.a.>

New dSPACE Data Dictionary Features

dSPACE Data Dictionary 3.3

The dSPACE Data Dictionary 3.3 (DD) has the following new features, enhancements and changes:

Where to go from here

Information in this section

New Data Dictionary Key Features.....	232
New DD MATLAB API Commands.....	236

New Data Dictionary Key Features

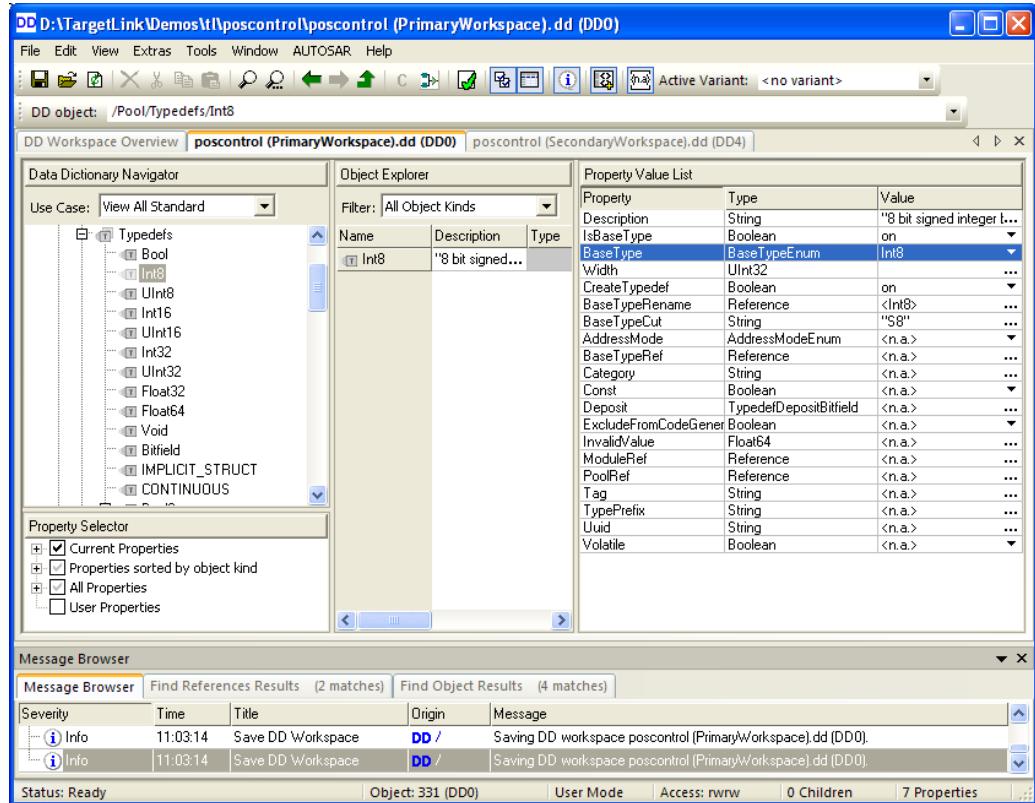
New key features of dSPACE Data Dictionary

Information on the new key features of dSPACE Data Dictionary 3.3 is provided below.

Multiple DD workspaces

Overview The Data Dictionary now supports an arbitrary number of DD workspaces (DD0 ... DD<n>). A DD workspace is an independent organizational unit (data container) and the largest entity that can be saved to file or loaded from a DD project file. In the DD Manager, each DD workspace can be represented by a *DD Workspace pane*, which is a predefined, fixed set of views, i.e., the Data Dictionary Navigator, the Property Selector, the Object Explorer and the Property Value List. The Message Browser is displayed only once, in a separate pane, shared between all DD workspaces.

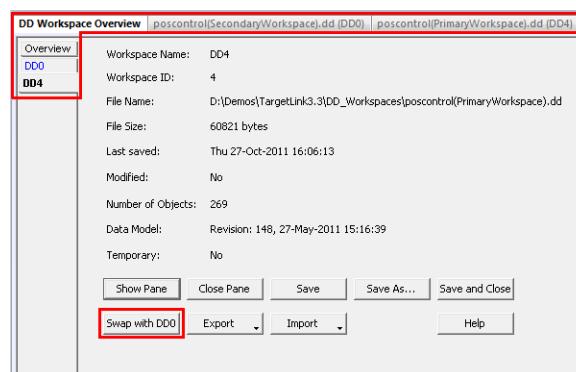
DD0 is the primary workspace which is implicitly linked with TargetLink models.



Benefits: DD workspaces enable you to

- Correctly update Data Dictionary projects with new data in an iterative development process (round trips)
- Try out different settings by swapping DD0 and DD<n>

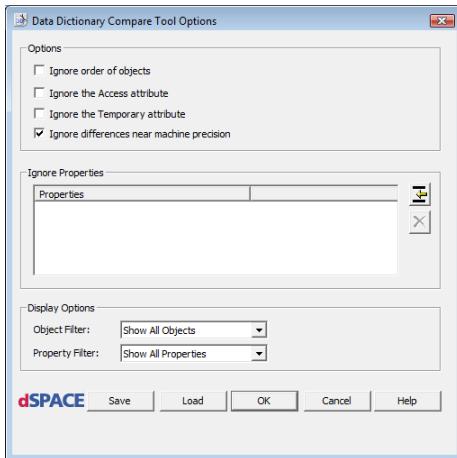
You can manage workspaces on the DD Workspace Overview pane shown below:



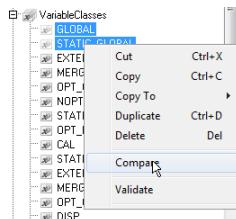
For details, refer to [Basics on the Data Dictionary Manager](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Compare Tool

The dSPACE Data Dictionary Compare Tool lets you compare selected DD objects within a DD workspace, DD objects in different DD workspaces, or different DD workspaces in their entirety. You can define which predefined options or attributes you want to ignore, and also define additional properties to ignore during the comparison, via the DD Compare Tool Options dialog. You can access the Data Dictionary Compare Tool Options dialog via Tools – Compare Tool Options ... in the DD Manager menu bar menu.



These settings are then used to compare two objects that you select in the DD Manager. You can start the DD Compare Tool via context menu.



If you have specified a report file (as shown above), the differences found are generated to that file and are automatically displayed in the Difference Report for dSPACE Data Dictionaries in the MATLAB Web Browser.

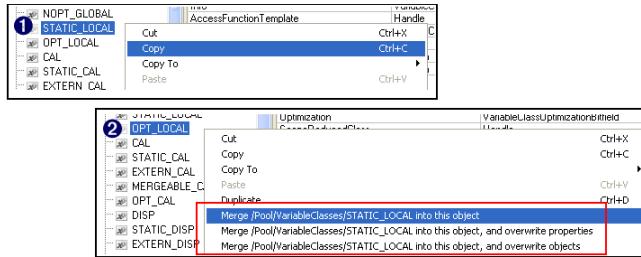
You can also compare DD objects residing in different DD workspaces. For details, refer to [Compare Tool Options](#) ([TargetLink Data Dictionary Manager Reference](#)).

Merge

The Data Dictionary allows you to merge the child objects and properties of one DD object into another, i.e., add them to a destination DD object.

The following merge methods are supported:

- Merge mode
- MergeOverwrite
- OverWrite



Merging child objects and properties is a multistep process. The objects to be merged must have the same type (object kind). Merging can be carried out via the context menu of the DD object concerned or via DD MATLAB API command as shown below:

```
[errorCode,mergeStruct] = dsdd('Merge',attributeName1,attributeValue1,...);
```

You can also merge DD objects between separate DD workspaces. For details, refer to [How to Merge or Replace DD Objects](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Copy to DD workspace

To quickly transfer object settings from one DD workspace to another you can copy DD objects to different DD workspaces. The dSPACE Data Dictionary provides the `Copy To <DD workspace name>` command for this. You can access it via the context menu of the DD object concerned or via DD MATLAB API command as shown below:

```
[hDDObject,errorCode] = dsdd('CopyToWorkspace',<objectIdentifier>,<DD_identifier>);
```



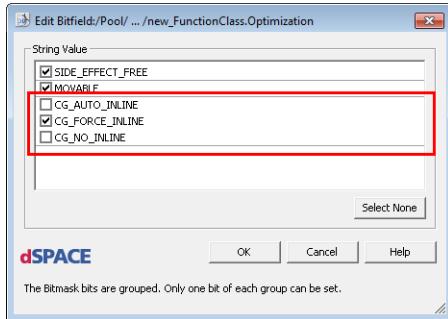
The selected objects are copied to the destination DD with their path retained, relative to the root object. If the parent object the copied child object belongs to does not exist in the destination DD, it is created automatically, otherwise the existing object is overwritten. Then the destination DD workspace opens with the new object highlighted.

For details, refer to [Copy To](#) ([TargetLink Data Dictionary Manager Reference](#)).

Significantly improved handling of bitfields

The handling of bitfields has been significantly improved:

- Values conflicting with each other are grouped together.
- It is not possible to select more than one value within one group.



Benefit: This usability improvement helps you to

- Set values more efficiently
- Avoid inconsistent value settings

For more information refer to [Managing Data with the Data Dictionary Manager](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Tip

It is not possible to create an inconsistent bitfield via the API. If you try to set conflicting values via the API, an error is generated (E06202).

Significantly improved handling of property dialogs

The handling of property dialogs, e.g., string property dialogs or scalar numeric property dialogs, was significantly improved. If an entered value is invalid, a message is shown in the status bar of the dialog and the OK button is grayed out.

For more information, refer to [Property Dialogs](#) ([TargetLink Data Dictionary Manager Reference](#)).

New DD MATLAB API Commands

Overview of new DD MATLAB API commands

The following new DD MATLAB API commands are available:

- `AutoLoad`
- `HasCurrentDataModelRevNr`
- `CheckPointsOfInclusion`
- `MakePointsOfInclusion`
- `IsDDFile`

In addition, there are two API commands that are described in [New Data Dictionary Key Features](#) on page 232:

- **Merge**
- **CopyToWorkspace**

AutoLoad

```
[errorCode,hDDOObject] = dsdd('AutoLoad',<fileName>);
[errorCode,hDDOObject] = dsdd('AutoLoad','file',<fileName>[, 'DDIdx', <DD_identifier>);
```

Loads the DD file depending on the type (object kind) of the file root object:

- If the file contains a complete DD, it is opened in the specified DD.
- If the file contains a partial DD whose file root has a unique position in /Pool or /Config, it is loaded into the /Pool or /Config area of the specified DD. The path to the default position in /Pool or /Config is created if necessary.
- Otherwise, the file is loaded to a /tmp object that is created if necessary. Partial DDs are always loaded so that no objects are overwritten. If the specified DD workspace does not exist, it is created.

HasCurrentDataModelRevNr

```
bHasCurrentDataModelRevNr
= dsdd('HasCurrentDataModelRevNr',<DD_identifier>);
```

Checks whether the specified DD's data model revision number equals the current data model revision number.

CheckPointsOfInclusion

```
nInvalidItems = dsdd('CheckPointsOfInclusion',<DD_identifier>)
```

Checks points of inclusion in the specified DD. Additionally, the internal point-of-inclusion list is updated. This list is used by the DD Manager to visualize points of inclusion.

MakePointOfInclusion

```
[errorCode] = dsdd('MakePointOfInclusion',<objectIdentifier>[,attributeName1,attributeValue1,...]);
```

Creates a DDIncludeFile for the specified object, making it a point of inclusion.

IsDDFile

```
[errorCode = dsdd('IsDDFile',<file>);
```

Checks whether the specified file is a Data Dictionary file.

Merge

```
[errorCode,mergeStruct] = dsdd('Merge',attributeName1,attributeValue1,...);
```

Merges the specified source into the specified destination object. Merging means that objects and properties that do not exist at the destination are copied from the source.

CopyToWorkspace

```
[hDDOObject,errorCode] = dsdd('CopyToWorkspace',<objectIdentifier>,<DD_identifier>);
```

Copies the specified object to the specified DD workspace. If the DD workspace does not exist, it is created. If the objects that constitute the path to the new object do not exist, they are created with their Data Model default settings. If the specified object exists in the target DD workspace, it is overwritten.

New Features of TargetLink 3.2

Where to go from here

Information in this section

New Production Code Generation Features.....	238
New AUTOSAR-Related Features.....	248
New Features of the dSPACE Data Dictionary.....	253

New Production Code Generation Features

Where to go from here

Information in this section

Online Parameter Modification.....	238
Debugging in SIL Simulation Mode.....	241
New TargetLink Sqrt Block.....	241
Enhancements to the Target Simulation Module.....	242
New TargetLink API commands.....	243
Code Generator Options.....	244
General Enhancements and Changes.....	245

Online Parameter Modification

Modifying parameter values online

TargetLink allows you to modify the parameter values of a simulation application in SIL and PIL simulation modes.

You can modify parameter values without having to regenerate code. This lets you perform the simulation with parameter values different from those specified in the model, which reduces the time needed for testing and lets you perform multiple tests of identical code with different parameter sets.

For further information, refer to [Basics on Modifying Parameter Values for Simulation](#) ( [TargetLink Preparation and Simulation Guide](#)).

Note

Online parameter modification does not support modifying the parameter values for the simulation in MIL mode. To modify the parameter values for the simulation in MIL mode, you have to set them in your model.

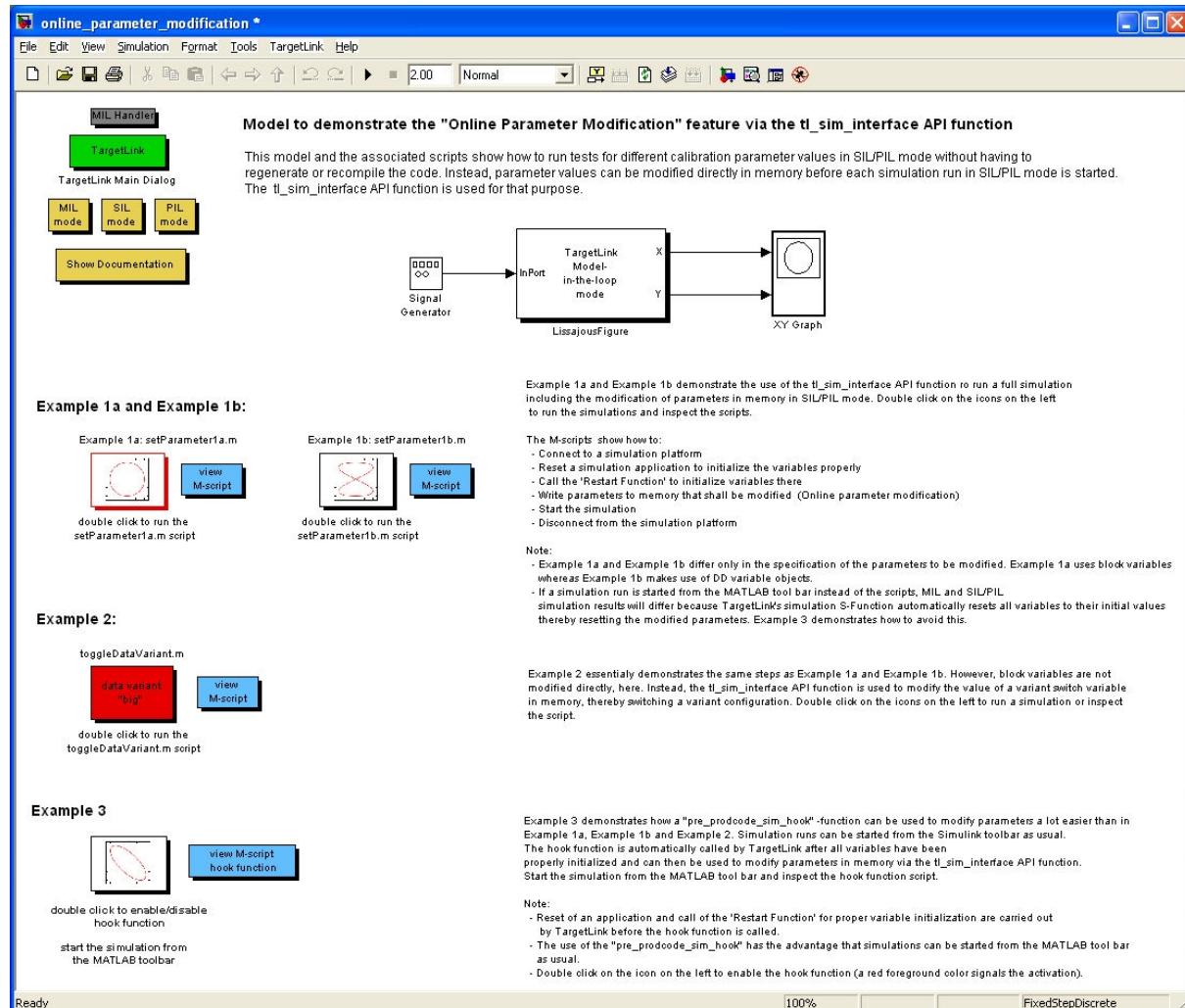
M-interface to the simulation application

With the `t1SimInterface` API function, TargetLink provides the M-interface to the simulation application. You can use it to access the simulation application downloaded to the evaluation board or loaded to the memory of the host PC (if it is a SIL application). The command has a number of parameters for performing various tasks, for example, connecting the simulation application to and disconnecting it from the simulation platform, resetting the simulation application, obtaining address information of variables and functions, reading and writing variables. You can use this command in the MATLAB command window or in M files to access the simulation application and modify parameter values online.

New demo model

TargetLink 3.2 provides a new demo model that helps you understand how to modify parameter values online.

The illustration below shows the root level of the `online_parameter_modification` demo model.



The different parts of the example show you the following aspects of modifying parameter values online:

- Modifying parameter values online and starting the simulation
- Toggling data variants and starting the simulation
- Modifying parameter values in a hook function

Related topics

References

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

Debugging in SIL Simulation Mode

Debugging production code

TargetLink supports debugging with the Microsoft® Developer Studio in SIL simulation mode, i.e., you can set breakpoints and debug the generated code while a SIL simulation is running. Debugging is possible with both the Microsoft® Visual Studio Professional Edition and the Microsoft® Visual Studio Express Edition (except Microsoft Visual Studio Express Edition 2010).

Debugging in SIL simulation mode is useful if you want to check whether and in which order code is executed, why the generated production code does not behave as expected or whether a task of a multi-rate system is calculated at all.

For details, refer to [How to Debug in SIL Simulation Mode](#) ([TargetLink Preparation and Simulation Guide](#)).

New TargetLink Sqrt Block

Sqrt block instead of function

As of TargetLink 3.2 and MATLAB R2010a, the `sqrt` function no longer appears in the Math block. If TargetLink 3.2 is used in combination with MATLAB R2010a (or newer), you can use a separate Sqrt block to perform square-root calculations. This block includes the `sqrt`, `sqrtsqrt` and `rsqrt` functions. If the `rsqrt` function is to be calculated, only a floating-point data type can be selected, otherwise integer data types are also allowed. Up to MATLAB R2009b, the `sqrt` function was provided by the Math block only. It depends on the TargetLink and MATLAB versions installed whether the Math block provides the Sqrt function, or if the Sqrt block is available instead.

If you open a model built with TargetLink < 3.1 in combination with MATLAB R2010a for the first time and an update is performed by the system, a Sqrt block is inserted for each Math block that computes the square root function. In all other cases, you have to perform a manual block upgrade by running the `t1_fix_sqrt` function on your model. `t1_fix_sqrt` replaces any Math block that uses `sqrt` with an equivalent Sqrt block that ensures the same behavior. This function does not affect libraries. To replace blocks in a library, open and unlock the library and apply this function to it.

Example

```
t1_fix_sqrt ('System',gcb);
```

Enhancements to the Target Simulation Module

(New) evaluation boards, microcontrollers, and compilers

The following table shows the combinations of evaluation boards, microcontrollers, and compilers supported by TargetLink 3.2 (TargetLink abbreviations). New evaluation boards, microcontrollers, and compiler versions are underscored. For details, refer to  [Evaluation Board Reference](#).

Evaluation Board	Microcontroller Type	Compiler ¹⁾
MCT HCS12 T-Board (DP256)	Freescale MC9S12DP256	Cosmic 4.5, 4.6, 4.7
		Metrowerks CodeWarrior 3.1
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	Cosmic 4.7, <u>4.8</u>
		No longer supported: Cosmic 4.5, 4.6
		Metrowerks CodeWarrior 3.1, <u>5.0</u>
Freescale 56F8367 Evaluation Module	Freescale MC56F8367	Metrowerks CodeWarrior 8.1
Axiom CMD-0565	Freescale MPC565	Wind River Diab 5.5, 5.6, 5.7 No longer supported: Wind River Diab 5.3
Axiom CME-0555	Freescale MPC555	Green Hills 5.0, 5.1 No longer supported: Green Hills 4.2
		No longer supported: Metrowerks CodeWarrior 8.1, 8.5, 8.7
		Wind River Diab 5.5, 5.6, 5.7
		No longer supported: Wind River Diab 5.3
Axiom MPC5554DEMO	Freescale MPC5554	Green Hills 5.0, 5.1 No longer supported: Green Hills 4.2
		Metrowerks CodeWarrior 2.3, 2.4, <u>2.6</u> No longer supported: Metrowerks CodeWarrior 2.2
		n/a GNU 3.4, 4.1
		Wind River Diab 5.5, 5.6, 5.7, <u>5.8</u> No longer supported: Wind River Diab 5.3
		Microtec 3.2, 3.3, 3.5, <u>3.7</u>
		Wind River Diab 5.5, 5.6, 5.7, <u>5.8</u> No longer supported: Wind River Diab 5.3
dSPACE DS1603		Green Hills 5.0, 5.1
		Metrowerks CodeWarrior 2.3, 2.4, <u>2.6</u>
		Wind River Diab 5.6, 5.7, <u>5.8</u>
Freescale MPC5604BEVB	Freescale MPC5600	Wind River Diab 5.5, 5.6, 5.8
		Green Hills 5.1
MCT S12X T-Board MCT S12X T-Board USB	Freescale MC9S12XDP512	Cosmic 4.7, <u>4.8</u> No longer supported: Cosmic 4.6
		Metrowerks CodeWarrior 4.7, <u>5.0</u> No longer supported: Metrowerks CodeWarrior 4.6
		Altium Tasking 8.6, 8.7
		No longer supported: Altium Tasking 7.5, 8.0, 8.5
I+ME Promotion Package 166	Infineon c167	

Evaluation Board	Microcontroller Type	Compiler ¹⁾
Infineon TBTC1766	Infineon TC1766	Altium Tasking 2.5, 3.2, 3.4 No longer supported: Altium Tasking 2.3, 3.0
Infineon TBTC1767	Infineon TC1767	Altium Tasking 2.5 (2.5r2p1 and younger), 3.2, 3.4 No longer supported: Altium Tasking 3.0
Infineon TBTC1796	Infineon TC1796	Altium Tasking 2.5, 3.2, 3.4 No longer supported: Altium Tasking 2.3, 3.0 HighTec GNU 3.3, 3.4
Infineon SK-EB XC2287	Infineon XC2287	Altium Tasking C166 VX 2.3, 2.4 No longer supported: Altium Tasking C166 VX 2.1, 2.2
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	Green Hills 5.0, 5.1 No longer supported: Green Hills 4.2 NEC 3.30, 3.40 No longer supported: NEC 3.10, 3.20
Renesas M3A-2154	Renesas M32192	Gaio 9, 10 Renesas 4.3, 5.0
Renesas EVB7058	Renesas SH-2E/7058	Renesas 9.0, 9.1, 9.3
Renesas SH72513 System Development Kit (SDK72513)	Renesas SH-2A- FPU/SH72513	Renesas 9.0, 9.1, 9.3

¹⁾ Compiler Suite Version Supported

For detailed information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ( [Evaluation Board Reference](#)).

Discontinued boards

No longer supported, no longer distributed The following boards are no longer supported by TargetLink and no longer distributed by dSPACE:

- Renesas EVB7055F (Renesas SH-2E/SH7055F microcontroller)
- Renesas EVB2633F (Renesas H8S/2633F microcontroller)

New TargetLink API commands

API commands available with TargetLink 3.2

In TargetLink 3.2, there are the following new TargetLink API commands.

- As of version 3.2, TargetLink allows you to modify the parameter values of a simulation application (Refer to [Online Parameter Modification](#) on page 238). The **t1_sim_interface** command provides the M-interface to the simulation application. It has a number of parameters (known as actions) for performing various tasks, for example, connecting the simulation application to and disconnecting it from the simulation platform, resetting the simulation application, modifying parameters, running the simulation.

For further information, refer to **t1SimInterface**.

Related topics**References**

[tlSimInterface](#) ( TargetLink API Reference)

Code Generator Options

New options for the Code Generator

With TargetLink 3.2 the following new Code Generator options are available.

Description	Explanation	Default Value	Min	Max
DoNotUseAssignArithmeticForAccumulation				
	<p>Controls how sums with more than two operands are implemented.</p> <p>If the option is not set (default), an auxiliary variable is introduced to hold intermediate results that are accumulated by assign additions or subtractions. If the option is set, TargetLink will not introduce an auxiliary variable and will not use assign arithmetic (<code>+=</code>, <code>-=</code>) for accumulations, so all terms are handled in a common statement.</p> <p>Example:</p> <p>Option not set:</p> <pre>Int16 Aux; Aux = In1; Aux += In2; Aux -= In3; Out = Aux + In4;</pre> <p>Option set:</p> <pre>Out = In1 + In2 - In3 + In4;</pre>	off	-	-

Description	Explanation	Default Value	Min	Max
SupportSinglePrecisionLibraries				
Allows the generation of ANSI-C99 single-precision, floating-point library operations.	<p>For statements with floating-point data types and math library functions, this option allows the Code Generator to use single-precision, floating-point library functions from the ANSI-C99 standard. These functions end with an <i>f</i> added to the normal math library functions, e.g. <i>sinf</i>.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ▪ No single-precision floating-point library functions are generated ▪ Single-precision floating-point library functions are generated if at least one operand or the result is floating-point but neither an operand nor the result is Float64 ▪ Single-precision floating-point library functions are always generated if at least one operand or the result is floating-point (either Float32 or Float64) 	0	-	-
SUPPRESSPERFORMANCEWARNINGFORTABLEDATAEVALUATION				
Suppresses a performance warning popup window, if the number of table data items exceeds a defined number of values.	If the table data exceeds a defined number of values, system performance can run low. A popup window informs you of this. However, popup windows can hinder automatic code generation. This option suppresses popup windows.	off	-	-

Obsolete and changed Code Generator options For information on obsolete and changed Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 380.

Related topics

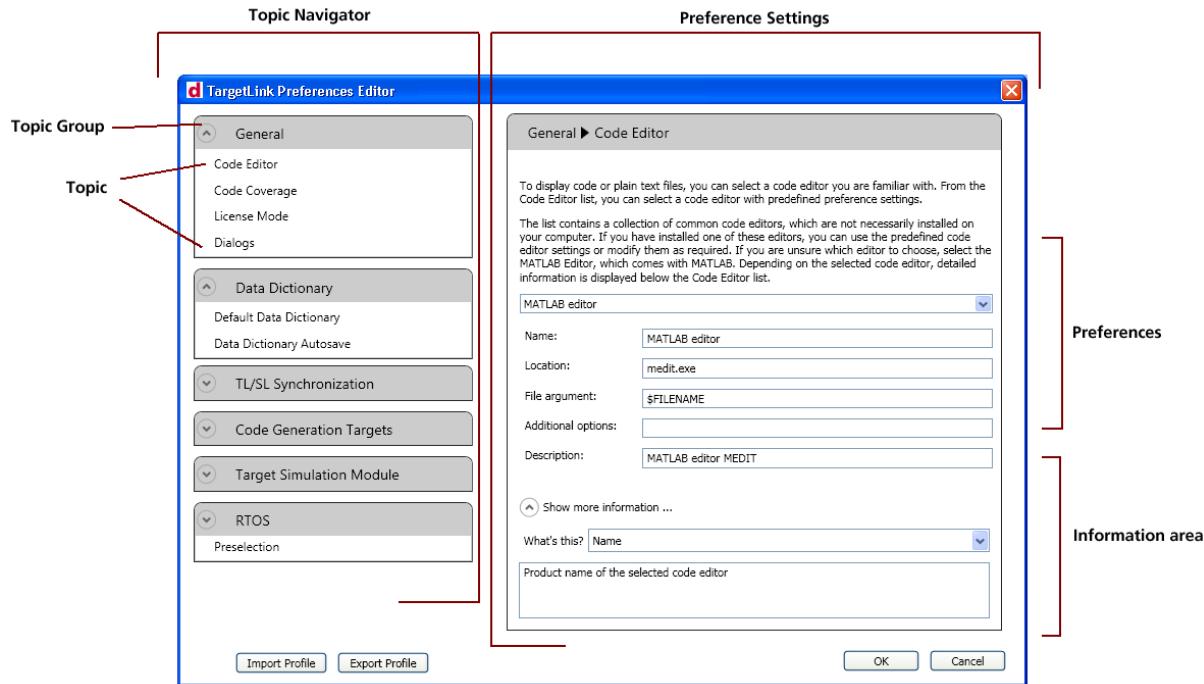
References

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

General Enhancements and Changes

Preferences Editor

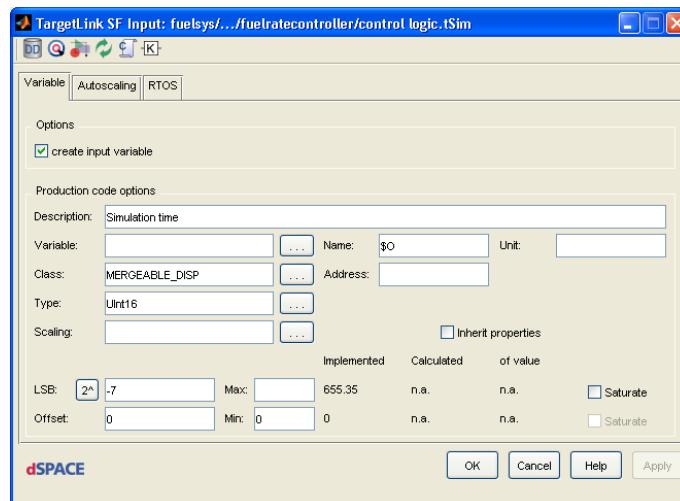
The new Preferences Editor replaces the former Preferences dialog, which was available up to TargetLink 3.1. The Preferences Editor contains configuration-relevant settings and allows you to easily modify settings such as the preferred code editor, synchronization properties and compiler installation paths.



For details, refer to [Basics on Using the Preferences Editor](#) ([TargetLink Customization and Optimization Guide](#)).

Stateflow Object dialogs

For a number of Stateflow objects, TargetLink now provides separate object dialogs. As shown in the TargetLink SF Input dialog below, these dialogs help you enter TargetLink data for Stateflow.



There are separate dialogs for the following Stateflow object types:

- SF Local
- SF Output

- SF Input
- SF Fcn Output
- SF Fcn Input
- SF Imported
- SF Exported
- SF Temporary
- SF Constant
- SF Parameter

For details, refer to [The Stateflow Objects and Their TargetLink Properties](#) ([TargetLink Model Element Reference](#)).

Option to switch between TargetLink and Simulink block dialogs

TargetLink provides a new option that controls whether the TargetLink or Simulink block dialog opens when you double-click a TargetLink block. You can use this option if you work with the TargetLink Blockset stand-alone and prefer to view and set block properties in Simulink dialogs.

You can set the option via TargetLink's [Preferences Editor](#) ([TargetLink Tool and Utility Reference](#)) or by entering `t1_pref('set', 'DialogProvider', 'Simulink')` in the MATLAB command window.

To switch back to TargetLink block dialogs, enter `t1_pref('set', 'DialogProvider', 'TargetLink')`.

Note

Although it is possible to switch to Simulink dialogs in TargetLink full-featured mode, it is strongly recommended to use this option only with the *TargetLink Blockset stand-alone*.

Improved code generation for floating-point processors

- With TargetLink, you can specify the default base data type for all the blocks copied from the TargetLink block library (tlib). For details, refer to [How to Preselect the Default Base Data Type](#) ([TargetLink Preparation and Simulation Guide](#)).
- If you specify a floating-point data type (Float32, Float64), this default base data type now also applies to parameters, for example, to gain parameters K_i or K_p used in the PIPT1 demo model.
- Support of 32-bit floating-point library functions according to the C99 standard. Via the `SupportSinglePrecisionLibraries` Code Generator option, you can allow the Code Generator to use single-precision, floating-point library functions from the C99 standard. For details, refer to [Code Generator Options](#) ([TargetLink Model Element Reference](#)).

Multiport Switch block

The Multiport Switch Block provides three new options (Zero-based contiguous, One-based contiguous and Specify indices) that let you specify the data port

order. For details, refer to [Multiport Switch Block](#) ([TargetLink Model Element Reference](#)).

Discontinued tool

Previous versions of TargetLink and the dSPACE Data Dictionary provided you with the **t1_export2dd** tool that let you import TargetLink model data into the dSPACE Data Dictionary. As of TargetLink 3.2, this tool is not part of the TargetLink Base Suite anymore.

If you want to use **t1_export2dd**, you can download it via the *TargetLink Product Support Center* (www.dspace.com/goto?TargetLinkProductSupportCenter).

Related topics

Basics

Changes in TargetLink API Functions.....	376
--	-----

New AUTOSAR-Related Features

Where to go from here

Information in this section

Features of the TargetLink AUTOSAR Module.....	248
Exchanging Software Component Containers with SystemDesk.....	250

Features of the TargetLink AUTOSAR Module

Supported AUTOSAR Releases

The TargetLink AUTOSAR Module supports the following AUTOSAR Releases:

- AUTOSAR Release 3.1 with Versions 3.1.0 (continued), 3.1.2 and 3.1.4 (new)
- AUTOSAR Release 3.0 with Versions 3.0.2 (continued), 3.0.4, and 3.0.6 (new)
- AUTOSAR Release 2.1 with Version 2.1.4 (continued)

AUTOSAR compiler abstraction

TargetLink now supports compiler abstraction as defined by AUTOSAR.

To make source code platform-independent, AUTOSAR has defined a set of macro definitions for source code elements such as functions, variables, and pointers. TargetLink can use these macros to define runnables, thus generating

software component code that complies with AUTOSAR's compiler abstraction definition.

Whenever you define a runnable, you can choose to generate runnable code with AUTOSAR's compiler abstraction macros. To generate code with compiler abstraction, you either do not change a runnable's FunctionClass, which is unset by default, or select **AUTOSAR/RUNNABLE**. If you select another FunctionClass for the runnable, such as **GLOBAL_FCN**, TargetLink generates code without compiler abstraction macros.

For details of how runnable code has changed in this TargetLink version, refer to [AUTOSAR-Related Migration Aspects](#) on page 378.

Improved support of RTE API functions

Rte_IRead TargetLink now supports generating code for implicit reading of composite data elements, i.e. arrays and structures.

Array passing

Versions 3.0.6 and 3.1.4 of AUTOSAR define the array passing scheme in detail. Whenever the RTE accesses an array there are two ways to implement corresponding RTE API functions:

- By pointing to the array type.
- By pointing to the array element type

TargetLink implements software components with RTE API functions that use the latter array passing scheme, i.e., they use pointers to array element types.

Note

Make sure that your RTE generator uses the same array passing scheme as TargetLink as described above.

If you use SystemDesk as an RTE code generator, you have to set SystemDesk's **RteUsePtr2ArrayBaseTypeForArgs** and the **RteUsePtr2ArrayBaseTypeForRetVal** RTE code generation options to True when working with TargetLink.

Improved generation of TargetLink subsystems

With the **t1_generate_swc_model** MATLAB API command, you can generate a TargetLink subsystem with imports/outputs and a predefined substructure from AUTOSAR data. The following improvements are provided by this version of TargetLink:

- TargetLink now generates buses with initial values for structured data that is exchanged with a TargetLink subsystem. This allows you to generate AUTOSAR-compliant code after TargetLink subsystem generation immediately.
- You can now directly generate one TargetLink subsystem for each software component that is defined in the AUTOSAR data.
- TargetLink now provides hook functions for customizing generation of TargetLink subsystems.

For details on generating TargetLink subsystems, refer to [Basics on Generating/Updating a Frame Model from Classic AUTOSAR Data](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#)).

Improved AUTOSAR import/export

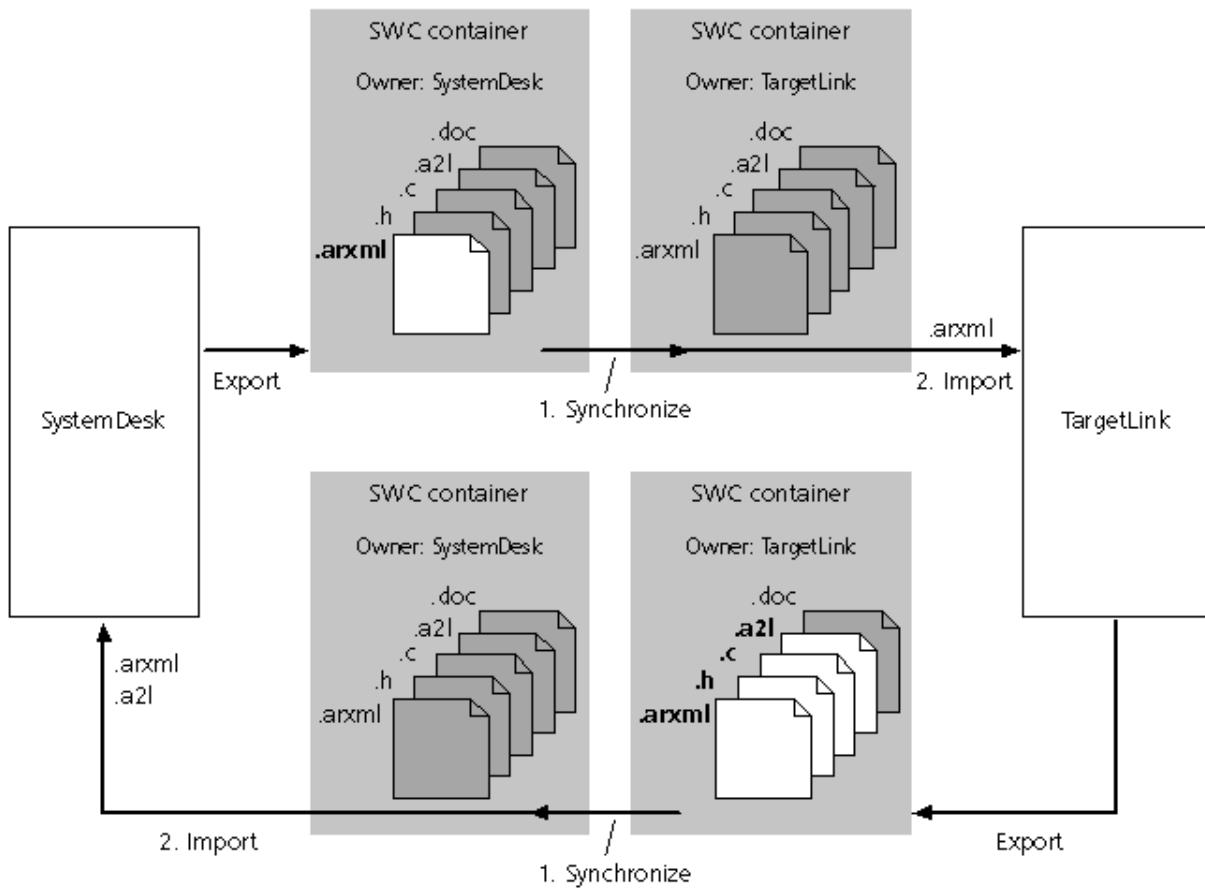
- With this version of TargetLink the AUTOSAR revision of an imported file is obtained directly from it. If you want to import/export AUTOSAR data from/to a file according to a revision that is not explicitly supported, you have to supply the corresponding AUTOSAR schema during import/export via the import/export configuration. However, your AUTOSAR data to be imported/exported must comply with one of the supported AUTOSAR Releases.
- TargetLink now lets you specify the overwrite/merge behavior of AUTOSAR elements during import. By configuring the AUTOSAR import you can select whether AUTOSAR elements such as type definitions, data access points, or runnables are overwritten during import, or whether additional/changed attributes are merged with the elements.

For details on importing and exporting AUTOSAR files, refer to [Exchanging AUTOSAR Files](#) ( [TargetLink Interoperation and Exchange Guide](#)).

Exchanging Software Component Containers with SystemDesk

Using software component containers for exchanging data

This version of TargetLink comes with a new approach for AUTOSAR-compliant development in combination with the architecture tool SystemDesk. It is based on the exchange of so-called SWC containers. The illustration below shows exchanging SWC containers between TargetLink and SystemDesk schematically.



Exchanging data using software component containers has the following benefits:

- Exchanging data safely, easily, and with a defined workflow.
- Integrating software component code into SystemDesk's simulation feature.
- Managing additional related files such as feature specifications, test specifications, etc.

Exporting data from TargetLink to SystemDesk

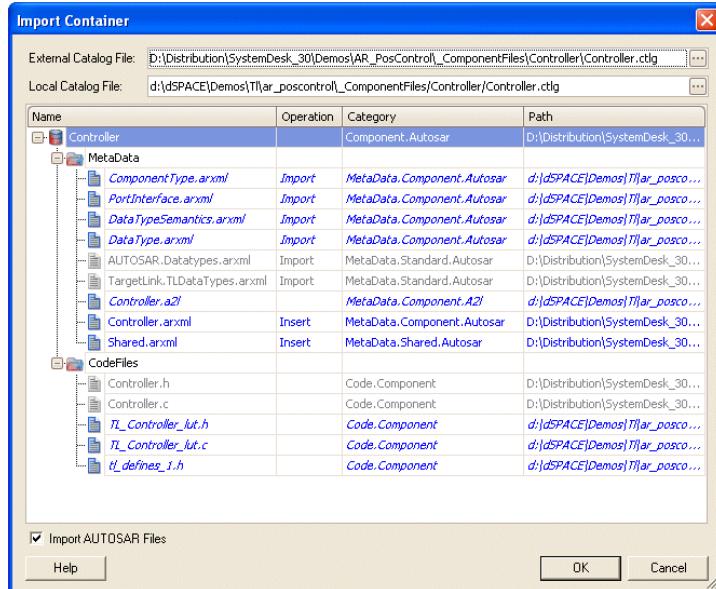
TargetLink lets you export a container, i.e., a bundle of files that completely represent the implementation of the software component you have generated code for. The files are described in a catalog file that is part of the container. You are now ready to import software component implementations to SystemDesk for further refinement.

Importing data to TargetLink from SystemDesk

You can import containers that have been exported from SystemDesk to TargetLink to create or update software components that reside in the /Pool1/Autosar area of the Data Dictionary.

During import the bundle of files from SystemDesk (external container) is synchronized with the files in TargetLink (local container). Updated AUTOSAR

files can be imported to TargetLink. The operations which are applied for synchronizing containers depend on configurable file categories. The default file categories are assigned automatically for a best-practice workflow.

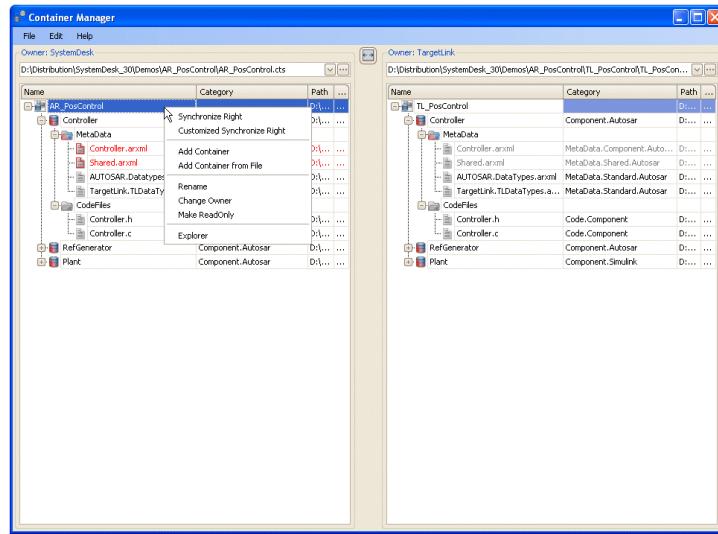


After importing the container, you are ready to refine the imported software components' behavior and generate software component implementations.

Managing containers

The Container Manager lets you manage containers exported from TargetLink and SystemDesk and is accessible from both tools. It provides the following features:

- Open container catalog files that are part of the containers, and review and synchronize the files in two containers.
- Add additional files, such as feature specifications for the software component, to the containers.
- Directly compare and synchronize a container with elements in SystemDesk's Package Manager while SystemDesk is running.



New Features of the dSPACE Data Dictionary

dSPACE Data Dictionary 3.2

The dSPACE Data Dictionary 3.2 (DD) has the following new features, enhancements and changes:

Where to go from here

Information in this section

New Key Features.....	253
New and Modified DD MATLAB API Commands.....	259

New Key Features

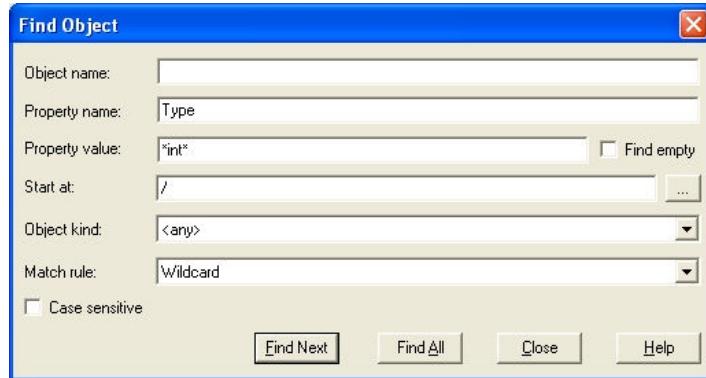
New key features of dSPACE Data Dictionary

Information on the new key features of dSPACE Data Dictionary 3.2 is provided below.

Improved Search

The search functionality in the Data Dictionary Manager was improved.

The Find Object dialog has been modified.



In addition to an object's name, you can now specify a property name and/or property value. You can even find objects with properties that are not set. Apart from regular expressions, you can now also use wildcards in the search strings to search for object names, property names and/or property values that you know only a part of.

If you want to find all the objects that match the specified search criteria, you can click **Find Object Results** (TargetLink Data Dictionary Manager Reference) pane opens and displays the search results.

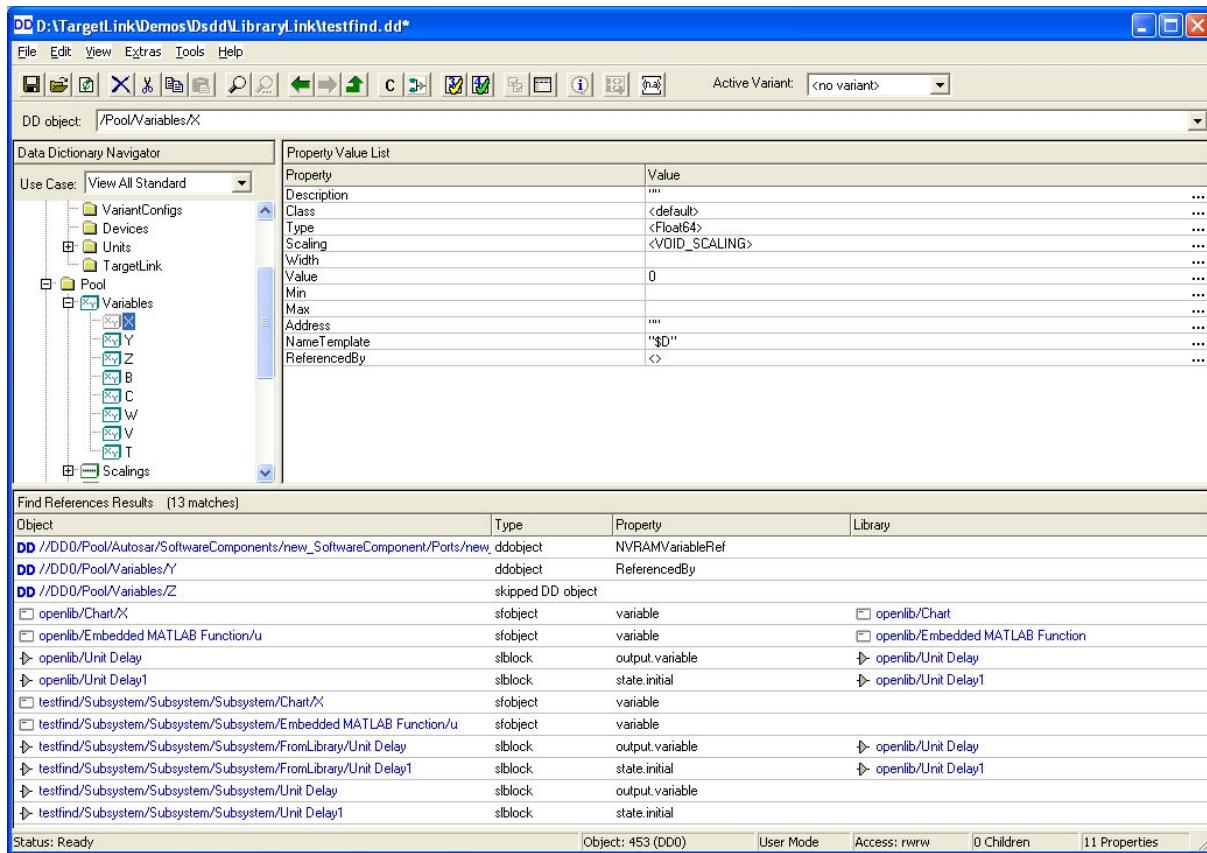
Find Object Results (29 matches)	
Object	Value
DD //DD0/Pool/Variables/RTOS/SystemTime.Type	Uln32
DD //DD0/Subsystems/picontroller/_tlDefines_a/Variables/STATIC_CAL.Type	signed_int
DD //DD0/Subsystems/picontroller/_tlDefines_a/Variables/STATIC_DISP.Type	signed_int
DD //DD0/Subsystems/picontroller/_tlDefines_a/Variables/_TL_FX_GROUND.Type	/Subsystems/picontroller/_tlBasetypes/Typedefs/ln32
DD //DD0/Subsystems/picontroller/picontroller_hi/Typedefs/_LOG_STRUCT_a_flags/Components/_Sa1_Inf8	

The hyperlinks in the Object column let you navigate to the DD objects in the Data Dictionary Navigator.

For further information, refer to [How to Find Data Dictionary Objects](#) (TargetLink Data Dictionary Basic Concepts Guide).

Finding references to DD objects

The Data Dictionary Manager lets you find other objects that a DD object is referenced by, for example, other DD objects, Simulink blocks and Stateflow objects. The functionality is available via context menus of Data Dictionary objects.



The new [Find References Results](#) ([TargetLink Data Dictionary Manager Reference](#)) pane displays the search results. The hyperlinks in the Object column let you navigate to the DD objects in the Data Dictionary Navigator.

For further information, refer to [How to Find Data Dictionary Object References](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Adding custom functionality to the Data Dictionary Manager

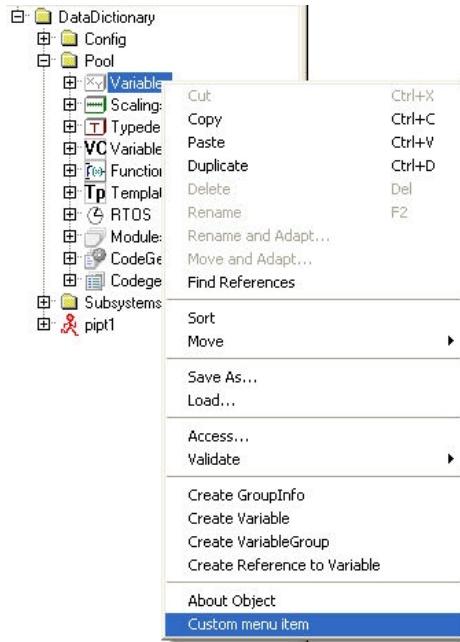
The dSPACE Data Dictionary lets you add custom functionality, i.e., user-defined MATLAB functions (M files) to the Data Dictionary Manager. You can add user-defined menu commands that invoke user-defined MATLAB functions (M files).

You can specify menu commands in

- The menus of the menu bar



- The context menus of objects in the Data Dictionary Navigator



- The context menus of properties in the property value list

Property	Value	
Description	"8 bit signed integer basetype"	Open Editor
IsBaseType	on	Rename
BaseType	Int8	Delete/Unset Property
Width		Goto Target
CreateTypedef	on	Edit Target
BaseTypeRename	<Int8>	
BaseTypeCut	"S8"	

Move
Show Property in Object Explorer
About Property
Custom menu item

The menu extensions are specified in the `DDManagerMenuExtension.xml` XML file in `%USERPROFILE%\Application`

`Data\dSPACE\<GUID>\TargetLink\DDMenuExtension` where `<GUID>` stands for a globally unique identifier that is provided by the TargetLink installation or in separate XML files that have to be stored in the current working folder.

For further information, refer to [Adding Custom Functionality to the Data Dictionary Manager](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Adding custom functionality to the Data Dictionary Manager

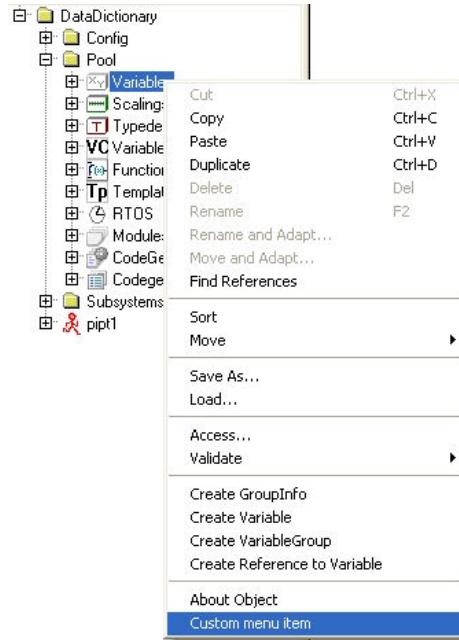
The dSPACE Data Dictionary lets you add custom functionality, i.e., user-defined MATLAB functions (M files) to the Data Dictionary Manager. You can add user-defined menu commands that invoke user-defined MATLAB functions (M files).

You can specify menu commands in

- The menus of the menu bar



- The context menus of objects in the Data Dictionary Navigator



- The context menus of properties in the property value list

Property	Value	
Description	"8 bit signed integer basetype"	Open Editor
IsBaseType	on	Rename
BaseType	Int8	Delete/Unset Property
Width		Goto Target
CreateTypedef	on	Edit Target
BaseTypeRename	<Int8>	
BaseTypeCut	"S8"	

Move	▶	
Show Property in Object Explorer		
About Property		
Custom menu item		

The menu extensions are specified in the `DDManagerMenuExtension.xml` XML file in `%USERPROFILE%\Application`

`Data\dSPACE\<GUID>\TargetLink\DDMenuExtension` where `<GUID>` stands for a globally unique identifier that is provided by the TargetLink installation or in separate XML files that have to be stored in the current working folder.

For further information, refer to [Adding Custom Functionality to the Data Dictionary Manager](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Custom messages and custom output views

To display feedback from your own tools, you can issue custom messages either in the Message Browser or in a custom output view, i.e., a separate pane that you can create in the Data Dictionary Manager.

The screenshot shows two tables side-by-side. The left table is titled 'Message Browser' and has columns for Severity, Time, Title, Origin, and Message. It contains several entries, including 'Info' messages about menu extensions and 'Unset' messages. The right table is titled 'CustomOutputView' and has columns for ID, Tag, Origin, and Message. It contains one entry with ID 1, Tag 'Info', Origin 'DD /Pool/Typedefs/Bool', and Message 'This is a custom message'.

Message Browser				
Severity	Time	Title	Origin	Message
Info	9:31:12	Menu extension specification XML file(s) are loaded	DD	The operation was completed successfully.
Info	9:31:12	E06808: Parsing DD Manager menu extension set	DD	Menu extension specification XML file "C:\Documents and Settings\UrsulaD\Appl
Info	9:31:12	DD Manager Started	DD	The DD Manager was successfully started.
Unset	9:35:18	MyTool	DD /Pool/Typedefs/Bool	This is a custom message

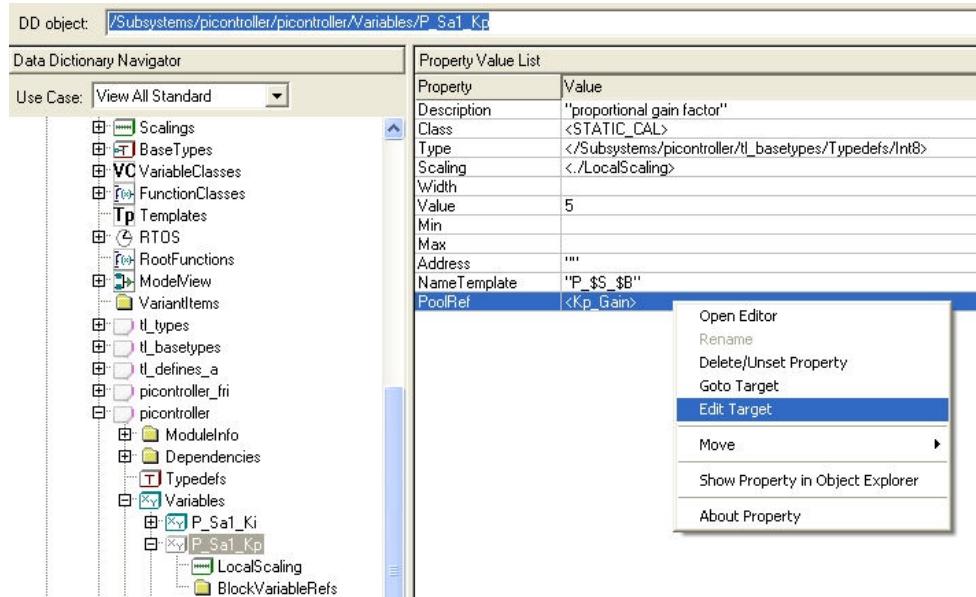
CustomOutputView			
ID	Tag	Origin	Message
1	Info	DD /Pool/Typedefs/Bool	This is a custom message

For further information, refer to [How to Create Custom Output Views](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)) and [How to Display Custom Messages](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Edit Target

There is a new Edit Target menu command in the context menu that is displayed when you right-click a reference property in the property value list. The command lets you open the object-specific dialog to edit the referenced object.

The command is available for references to variable or typedef objects.



For further information, refer to [Edit Target](#) ([TargetLink Data Dictionary Manager Reference](#)).

New and Modified DD MATLAB API Commands

Modified DD MATLAB API command: `dsddman`

The `dsddman` API command is the command-line interface to the Data Dictionary Manager. It has been extended to let you display feedback from your own tools, either in the Message Browser or a custom output view, i.e., a separate message pane in the Data Dictionary Manager.

- `dsddman('AddMessage', ...)` lets you issue a custom message in the Message Browser. For further information, refer to [How to Display Custom Messages](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).
- `dsddman('DemandCustomOutputView', ...)` lets you create a custom output view. For further information, refer to [How to Create Custom Output Views](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).
- `dsddman('AddCustomMessage', ...)` lets you issue a custom message in a custom output view. For further information, refer to [How to Display Custom Messages](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).
- `dsddman('ClearOutputView', ...)` lets you clear a custom output view.
- `dsddman('CloseOutputView', ...)` lets you close a custom output view.

New option for DD MATLAB API command `dsdd_compare`

The new `IgnoreAttributes` option with the `dsdd_compare` API command lets you exclude certain attributes from the comparison of two DD project files or selected DD objects

The following attributes can be excluded:

- `access`
- `numOfChildren`
- `objectKind`
- `temporary`

For further information, refer to [TargetLink Data Dictionary Reference](#).

New Features of TargetLink 3.1

Where to go from here

Information in this section

New Production Code Generation Features	260
New AUTOSAR-Related Features	278
New Features of the dSPACE Data Dictionary	282

New Production Code Generation Features

Where to go from here	Information in this section
	<ul style="list-style-type: none"> Code Generation from the dSPACE Data Dictionary..... 260 New TargetLink Bit Operation Blocks..... 263 Enhancements to the Target Simulation Module..... 264 Requirements Traceability down to Generated Code and Documentation..... 267 Code Generation with Variable Vector Widths..... 268 Enhancements and Changes to the TargetLink Main Dialog..... 268 Code Generator Options..... 272 General Enhancements and Changes..... 274

Code Generation from the dSPACE Data Dictionary

Overview TargetLink 3.1 offers production code generation straight from the dSPACE Data Dictionary. Beside incremental code generation and model referencing, this is another means of partitioning your models and mapping the model or model parts and the generated C code files. TargetLink allows you to generate code for code generation units (CGUs), which can be either TargetLink subsystems, subsystems configured for incremental code generation, referenced models, or CodeGenerationUnit objects in the DataDictionary. For further information, refer to [Decomposing Models for Distributed Development](#) ( [TargetLink Customization and Optimization Guide](#)) and [Basics on Generating Code from the Data Dictionary](#) ( [TargetLink Customization and Optimization Guide](#)).

Use cases of code generation from the dSPACE Data Dictionary With regard to DD-based code generation, the use cases can roughly be mapped to partitioning the model into separate components, central specification and generation of common modules, and support of A2L file generation as a whole.

Splitting models into more manageable components TargetLink lets you split large TargetLink models into smaller ones, i.e., more manageable components such as TargetLink subsystems, referenced models, or subsystems configured for incremental code generation, and generate code for them separately.

If you want to generate code for common interface variables that are used in different subsystems, for example, it is important that the resulting modules that

hold the production code do not overwrite each other unintentionally. To prevent mutual overwriting of modules by different model components, TargetLink lets you specify the module ownership of a module. This guarantees that a module is generated only together with its owner.

TargetLink lets you compile model components separately. If a model component requires variables from a module that is not yet generated and does not belong to the model component, code can be generated, but not compiled as the variable definitions are missing. Thus, for model components using objects from foreign modules, stub code versions of these modules are generated. Using these stub code versions allows you to compile and simulate the model components.

TargetLink uses production code as well as stub code for code compilation to guarantee the completeness of required code files. Thus, it lets perform tests in a component-oriented way, i.e., you can perform MIL and PIL simulations even with model components that are not the owner of certain modules, or interface variables, respectively. This allows you to test a single component, several components at a time, or to perform a complete system test.

Central specification and generation of globally used

modules TargetLink lets you assign variables to a specific module, for example, all calibration variables to a separate module. The generated module contains all project global data (variables) independent of their use in a specific model.

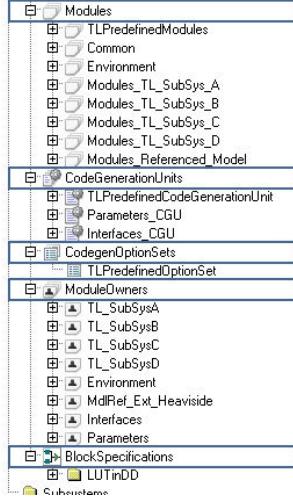
TargetLink lets you generate parameter code independently of the function algorithm. As parameters change more frequently than functional code and functional code does not have variants, it is advantageous to have the functional code separated from the variable code (variable declaration/definition).

Generating ASAP2 files from the Data Dictionary TargetLink allows you to provide a measurement and calibration (MC) system with all the information required for accessing an ECU for calibration. You can, for example, generate calibration or measurement variables used in multiple TargetLink subsystems into a single module, and generate parameter code independently of the function algorithm. Using Look-Up Table objects from the DD, you can generate ASAM-MCD 2MC (A2L) files listing all the parameters and measurement variables running on an ECU as a whole. In order to generate the A2L file, all the lookup table specifications of the Look-Up Table variables must also exist in the dSPACE Data Dictionary.

Integrating external code TargetLink allows the integration of external code. TargetLink modules can integrate external code by including external code files or by providing variable definitions for external code. Both can be specified in the Data Dictionary. This allows external modules to use, for example, calibratable variables that are not defined in the external modules themselves but in the calibration module that is to be generated by TargetLink.

DD objects for code generation from the dSPACE Data Dictionary

Technically, the above use cases are realized by objects in the pool area of the dSPACE Data Dictionary that are relevant for DD-based code generation.



Module Module objects allow you to describe your application on a file basis as a pair consisting of one *.c and one *.h file.

ModuleOwnership The ModuleOwnership object lets you define the owner of a module precisely. It specifies which code generation unit (CGU) generates the production code version of the module.

CodeGenerationUnits A code generation unit (CGU) is the starting point for code generation. A CGU is either a TargetLink subsystem, a subsystem configured for incremental code generation, a referenced model or a DD CodeGenerationUnit. You can generate production code for each CGU separately or for several code generation units simultaneously.

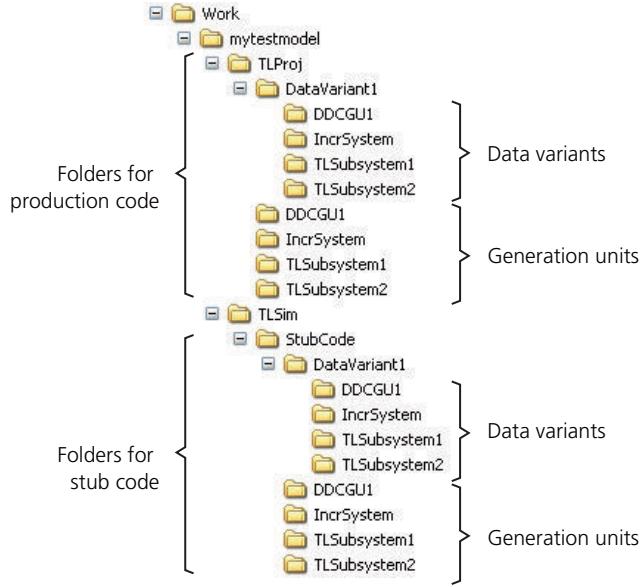
CodegenOptionsSet Code Generator option sets allow you to configure the Code Generator for production code generation when generating code for a DD CodeGenerationUnit.

BlockSpecifications Currently, you can use block specifications for table blocks such as Look-Up Table (1D), Look-Up Table (2D), Prelookup and Interpolation Using Prelookup.

For further information, refer to [Basics on Generating Code from the Data Dictionary](#) ([TargetLink Customization and Optimization Guide](#)).

New file deposition structure

In contrast to older versions, TargetLink 3.1 does not place the generated production code directly in the working directory, but in a specific file deposition structure. TargetLink creates a clear subfolder structure to provide completely separate files for the individual code generation units. The following illustration shows an example:

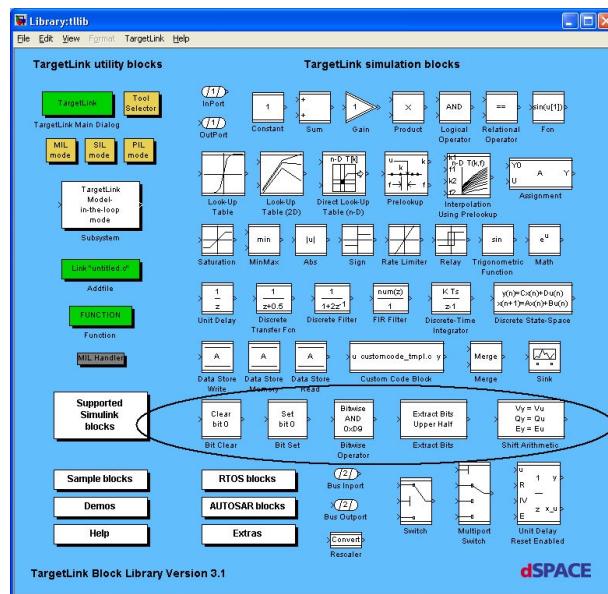


Refer to *File Deposition Structure of the Files Generated by the TargetLink Code Generator* contained in the [TargetLink File Reference](#) shipped prior to TargetLink 4.3 (Release 2017-B).

New TargetLink Bit Operation Blocks

Support of native bit-operation blocks

With TargetLink 3.1, native bit-operation blocks are introduced.



- Bit Clear block
To set a specified bit of an integer signal to zero. For further information, refer to [Bit Clear Block](#) ([TargetLink Model Element Reference](#)).
- Bit Set block
To set a specified bit of an integer signal to one. For further information, refer to [Bit Set Block](#) ([TargetLink Model Element Reference](#)).
- Bitwise Operator block
To perform a specified bitwise operation on the integer input signal. For further information, refer to [Bitwise Operator Block](#) ([TargetLink Model Element Reference](#)).
- Extract Bit block
To output a selection of contiguous bits from an integer input signal. For further information, refer to [Extract Bits Block](#) ([TargetLink Model Element Reference](#)).
- Shift Arithmetic block
To perform a bitshift operation on an integer signal. For further information, refer to [Shift Arithmetic Block](#) ([TargetLink Model Element Reference](#))

Enhancements to the Target Simulation Module

(New) evaluation boards, microcontrollers, and compilers

The following table shows the combinations of evaluation boards, microcontrollers, and compilers supported by TargetLink 3.1 (TargetLink abbreviations). New evaluation boards, microcontrollers, and compiler versions are underscored. For details, refer to [Evaluation Board Reference](#).

Evaluation Board	Microcontroller Type	Compiler ¹⁾
MCT HCS12 T-Board (DP256)	Freescale MC9S12DP256	Cosmic 4.5, 4.6, 4.7
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	No longer supported: Cosmic 4.4
MCT HCS12 T-Board (DP256)	Freescale MC9S12DP256	Metrowerks CodeWarrior 3.1
MCT HCS12 T-Board (DP512)	Freescale MC9S12DP512	No longer supported: Metrowerks CodeWarrior 1.2, 2.0
Freescale 56F8367 Evaluation Module	Freescale MC56F8367	Metrowerks CodeWarrior 8.1
Axiom CMD-0565	Freescale MPC565	Wind River Diab 5.3, 5.5, 5.6, <u>5.7</u> No longer supported: Wind River Diab 5.0, 5.2
Axiom CME-0555	Freescale MPC555	Green Hills 4.2, 5.0, <u>5.1</u> No longer supported: Green Hills 3.0, 3.5, 3.6, 4.0

Evaluation Board	Microcontroller Type	Compiler ¹⁾
Axiom CME-0555	Freescale MPC555	Metrowerks CodeWarrior 8.1, 8.5, 8.7 No longer supported: Metrowerks CodeWarrior 6.0
Axiom CME-0555	Freescale MPC555	Wind River Diab 5.3, 5.5, 5.6, <u>5.7</u> No longer supported: Wind River Diab 4.3, 4.4, 5.0, 5.2
Axiom MPC5554DEMO	Freescale MPC5554	Green Hills 4.2, 5.0, <u>5.1</u> No longer supported: Green Hills 4.0
Axiom MPC5554DEMO	Freescale MPC5554	Metrowerks CodeWarrior 2.2, 2.3, <u>2.4</u> No longer supported: Metrowerks CodeWarrior 1.5
Axiom MPC5554DEMO	Freescale MPC5554	n/a GNU 3.4, 4.1
Axiom MPC5554DEMO	Freescale MPC5554	Wind River Diab 5.3, 5.5, 5.6, <u>5.7</u> No longer supported: Wind River Diab 5.2
dSPACE DS1603	Freescale MPC5554	Microtec 3.2, 3.3, <u>3.5</u>
dSPACE DS1603	Freescale MPC5554	Wind River Diab 5.3, 5.5, 5.6, <u>5.7</u>
<u>Freescale MPC5561EVB</u>	<u>Freescale MPC5561</u>	<u>Green Hills 5.0, 5.1</u>
<u>Freescale MPC5561EVB USB</u>		
<u>Freescale MPC5561EVB</u>	<u>Freescale MPC5561</u>	<u>Metrowerks CodeWarrior 2.3, 2.4</u>
<u>Freescale MPC5561EVB USB</u>		
<u>Freescale MPC5561EVB</u>	<u>Freescale MPC5561</u>	<u>Wind River Diab 5.6, 5.7</u>
<u>Freescale MPC5561EVB USB</u>		
MCT S12X T-Board	Freescale MC9S12XDP512	Cosmic 4.6, 4.7
MCT S12X T-Board USB		
MCT S12X T-Board	Freescale MC9S12XDP512	Metrowerks CodeWarrior 4.6, 4.7
MCT S12X T-Board USB		No longer supported: Metrowerks CodeWarrior 4.1, 4.5
I+ME Promotion Package 166	Infineon c167	Altium Tasking 7.5, 8.0, 8.5, 8.6, 8.7 No longer supported: Altium Tasking 6.0
Infineon TBTC1766	Infineon TC1766	Altium Tasking 2.3, 2.5, 3.0, <u>3.2</u> No longer supported: Altium Tasking 2.2

Evaluation Board	Microcontroller Type	Compiler ¹⁾
Infineon TBTC1767	Infineon TC1767	Altium Tasking 2.5 (2.5r2p1 and younger), 3.0, 3.2
Infineon TBTC1796	Infineon TC1796	Altium Tasking 2.3, 2.5, 3.0, 3.2 No longer supported: Altium Tasking 2.2
Infineon TBTC1796	Infineon TC1796	HighTec GNU 3.3, 3.4
Infineon SK-EB XC2287	Infineon XC2287	Altium Tasking C166 VX 2.1, 2.2, 2.3
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	Green Hills 4.2, 5.0, 5.1 No longer supported: Green Hills 3.5, 4.0
NEC Fx3-CAN it!	NEC V850ES/FG3- μPD70F3377	NEC 3.10, 3.20, 3.30 No longer supported: NEC 2.50, 2.70
Renesas EVB2633F	Renesas H8S/2633F	Renesas 6.0, 6.2 No longer supported: Renesas 3.0
Renesas M3A-2154	Renesas M32192	Gaio 9, 10
Renesas M3A-2154	Renesas M32192	Renesas 4.3, 5.0 No longer supported: Renesas 2.0
Renesas EVB7055F	Renesas SH-2E/SH7055F	Renesas 9.0, 9.1, 9.3 No longer supported: Renesas 6.0, 7.0, 8.0
Renesas EVB7058	Renesas SH-2E/7058	Renesas 9.0, 9.1, 9.3 No longer supported: Renesas 6.0, 7.0, 8.0
Renesas SH72513 System Development Kit (SDK72513)	Renesas SH-2A- FPU/SH72513	Renesas 9.0, 9.1, 9.3

¹⁾ Compiler Suite Version Supported

For detailed information on the evaluation boards supported by TargetLink, refer to [Combinations of Evaluation Boards and Compilers](#) ( [Evaluation Board Reference](#)).

Discontinued boards

No longer supported, no longer distributed The following boards are no longer supported by TargetLink and no longer distributed by dSPACE:

- NEC Drivelt Evaluation Board
- Renesas Evaluation Board MSA2114
- FS Forth-Systeme STart276 Development Board
- Texas Instruments TMS470R1x Evaluation Board

No longer supported, still distributed The following board is no longer supported by TargetLink but still distributed by dSPACE:

- Infineon TriBoard TC1775 Evaluation Board

Tip

To use the unsupported evaluation boards with TargetLink 3.1, please contact dSPACE.

Still supported, no longer distributed The following boards are still supported by TargetLink but no longer distributed by dSPACE:

- Renesas EVB7055F Evaluation Board
- Renesas EVB7058 Evaluation Board
- Axiom CMD-0565 Evaluation Board
- MCT HCS12 T-Board (DP256) and Freescale M68EVB912DP256 Evaluation Boards

Requirements Traceability down to Generated Code and Documentation

Including requirement information in code and documentation

TargetLink supports the requirement management interface of Simulink Verification and Validation™ software by The MathWorks.

TargetLink blocks can be linked to software requirements, which are stored in requirement documents or requirement management tools in the same way as with Simulink and Stateflow objects. If a TargetLink model contains objects which are linked to requirements, TargetLink can include information on these requirements in the generated code and/or in the generated documentation.

This traceability between requirement, model and code is helpful if you want to verify whether a requirement is correctly implemented in the software, for example, when inspecting the generated code. For further information, refer to [Including Requirement Information in Code and Documentation](#) ( [TargetLink Interoperation and Exchange Guide](#)).

Requirements linked to TargetLink subsystems

TargetLink ignores requirements linked to a TargetLink subsystem via the subsystem's context menu. Therefore, you must not use a subsystem's context menu to link a requirement to it.

To link a requirement to a TargetLink subsystem, open it and use the commands in its menu.

Code Generation with Variable Vector Widths

Varying-width code for vectorized variables

TargetLink 3.1 can generate varying-width code for vectorized variables. The width of the variables can be adjusted at compilation time.

Example for code with variable vector widths:

```
Int16 Sa5_Gain[MY_WIDTH];
...
for (Aux_S32 = 0; Aux_S32 < MY_WIDTH; Aux_S32++) {
...
Sa5_Gain[Aux_S32] = ...
}
```

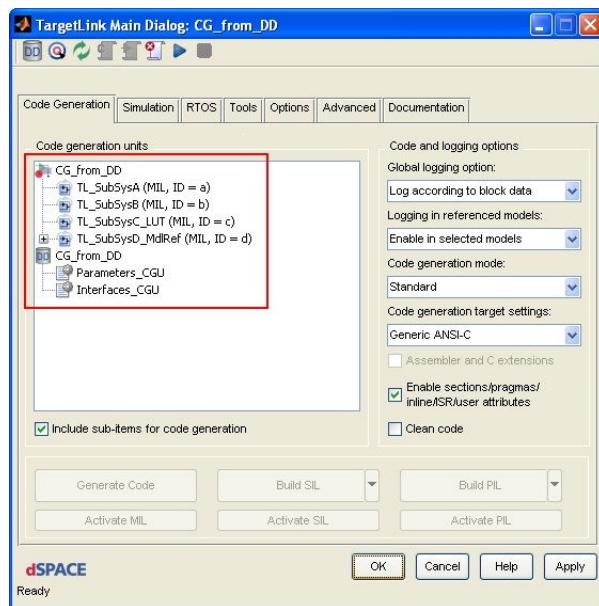
For further information, refer to [Specifying Variable Vector Widths via Width Macros](#) ([TargetLink Customization and Optimization Guide](#)).

Enhancements and Changes to the TargetLink Main Dialog

Redesign of the TargetLink Main Dialog

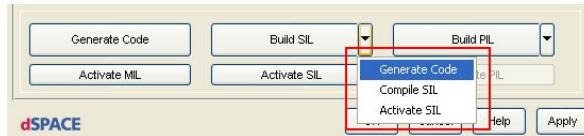
As of TargetLink 3.1, the TargetLink Main Dialog has been redesigned.

Code Generation page The TargetLink Main Dialog is now used to trigger code generation not just for TargetLink subsystems, but also for incremental subsystems, referenced models and Data Dictionary CodeGenerationUnit objects. For further information, refer to [Decomposing Models for Distributed Development](#) ([TargetLink Customization and Optimization Guide](#)).



The TargetLink Main Dialog offers new controls for the build process in MIL, SIL and PIL simulation modes. The Build SIL and Build PIL menu buttons let you

generate and compile the production code for the host and the target simulation application, respectively. The simulation application is then loaded to the RAM of your development PC (SIL) or to your evaluation board (PIL). You can invoke the build process steps separately by selecting the commands on the menu button.



The Activate MIL, Activate SIL, and Activate PIL buttons are an easy way to activate a specific simulation mode.

Note

The Activate SIL and Activate PIL buttons are active only if you built a corresponding simulation application beforehand.

The Code generation mode drop-down list lets you specify the code generation mode. The following code generation modes are available:

- Standard
- AUTOSAR
- RTOS

For details on the Code Generation Page of the TargetLink Main Dialog, refer to [TargetLink Main Dialog Block](#) ([TargetLink Model Element Reference](#)).

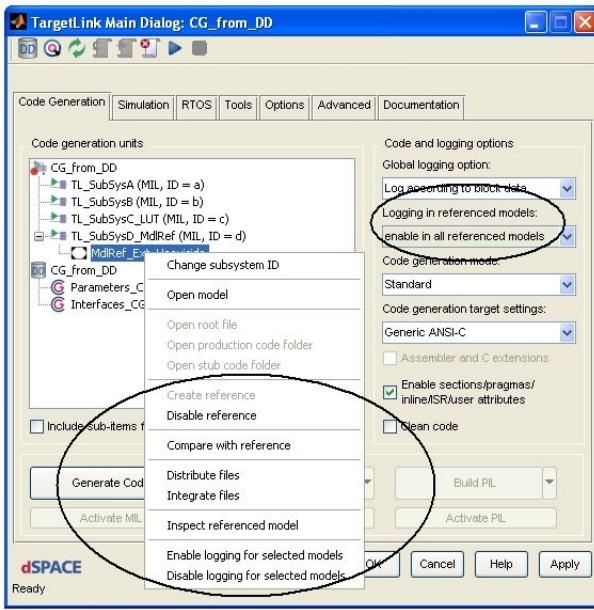
Note

With the Code generation mode drop-down list, the Enable multirate code generation checkbox on the RTOS page is obsolete and was therefore removed.

The Code generation options drop-down list was renamed Code generation target settings.

The Change ID button was removed. You can now change the ID of a code generation unit by selecting the Change subsystem ID command from the context menu of the TargetLink subsystem you selected in the Code generation units tree.

With TargetLink 3.1, the Model Referencing Control Center has been discontinued. To manage models containing model references, TargetLink now provides commands in the context menus of the referenced models listed in the Code generation units tree of the TargetLink Main Dialog.



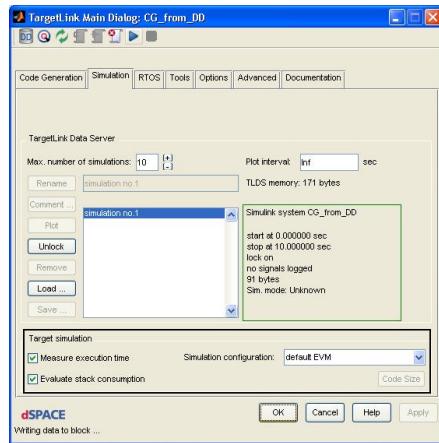
The new Logging in referenced models drop-down list lets you specify logging options for referenced models. The following logging options are available:

- Enable in all referenced models
- Disable in all referenced models
- Enable in selected models

For details on model referencing, refer to [Basics on Model Referencing](#) ([TargetLink Customization and Optimization Guide](#)).

Simulation page The new Target simulation frame comprises the following controls:

- Evaluate stack consumption
- Measure execution time
- Target simulation configuration



Note

The Evaluate stack consumption and Measure execution time checkboxes were moved from the Code Generation page to the Simulation page of the TargetLink Main Dialog.

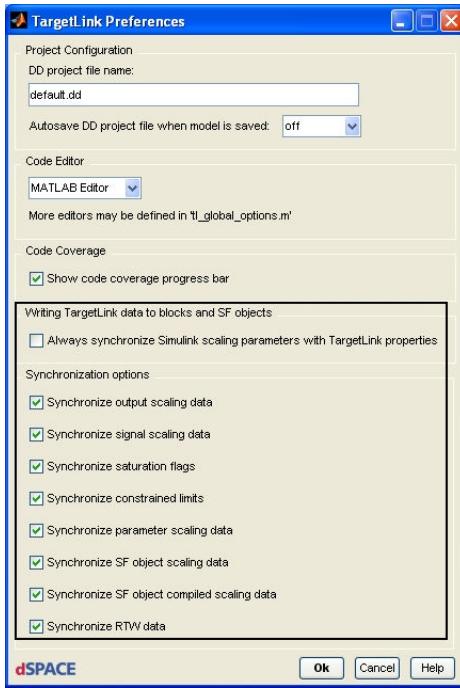
As of TargetLink 3.1, the Disable target timeout checkbox is no longer available. However, you can still set this option via the API command `t1_set(gcbh, 'frameopt.targettimeout', 'on')`. With this flag set to 1 (on), the host PC will always wait for a response from the EVB target, which may cause MATLAB to hang if the EVB does not respond. You should use this option if you run the EVB in a target debugger.

Tools page The ASAP 1b interface drop-down list allows you to generate information specifically for ASAM-MCD 1b interfaces into the ASAM-MCD 2MC (A2L) file. You can specify several ASAM-MCD 1b interfaces. The following ASAM-MCD 1b interfaces are supported:

- ADDRESS
- DIM
- CCP
- XCP
- DCI_GME1
- DCI_GSI1
- ETK
- CANAPE_EXT

For further information, refer to [TargetLink Main Dialog Block](#) ( [TargetLink Model Element Reference](#)).

Preferences dialog The Preferences dialog of TargetLink 3.1 provides new synchronization options for updating TargetLink property values to the values of corresponding Simulink properties, and vice versa. You can selectively control the synchronization process by selecting synchronization options.



Code Generator Options

Sets of Code Generator options

TargetLink lets you configure Code Generator options to tailor the code generation process. As of TargetLink 3.1, you can also configure different sets of options, each of which influences code generation in a different way. As a result, you can analyze and test the effects of different Code Generator options on the generated code. Depending on the type of code generation unit you want to create production code for, there are different ways of accessing these Code Generator options. You can set Code Generator options easily by clicking the All Options button on the Advanced Page of the TargetLink Main Dialog or by double-clicking a `CodegenOptions` object in the dSPACE Data Dictionary. The type of code generation unit also determines where the options are stored. For a TargetLink subsystem, a subsystem configured for incremental code generation, or a referenced model, the Code Generator options are stored at the model, but when you configure Code Generator options for producing C code for `CodeGenerationUnit` objects, they are stored in the dSPACE Data Dictionary. In contrast to the Code Generator options you specify via the TargetLink MainDialog, an option set object applies only to those CGU objects it is assigned to. Code Generator options you configured for TargetLink subsystems, incremental subsystems, or referenced models via the TargetLink Main Dialog do not affect `CodegenOptions` objects stored in the dSPACE Data Dictionary and vice versa.

TargetLink provides the API commands `dsdd_import_optionset` and `dsdd_export_optionset` to exchange Code Generator options between TargetLink and the dSPACE Data Dictionary, and to compare a `CodegenOptions` object (`dsdd_compare_optionset`) stored in the dSPACE Data Dictionary with Code Generator options you configured via the TargetLink Main Dialog. For further information, refer to [Basics on Configuring the Code Generator for Production Code Generation](#) ([TargetLink Customization and Optimization Guide](#)).

New options for the Code Generator

TargetLink 3.1 provides the following new Code Generator options.

Code Generator Option	Subject Area in the Code Generator Options Dialog	TargetLink 3.1 Default	Most Compatible TargetLink 2.x /3.x Behavior
AllowDuplicationOfImplicitAUTOSAR ARDDataAccess	Code Generation \ AUTOSAR	ON	OFF
AssumeFunctionCallSemanticsFor RteAPIArguments	Code Generation \ AUTOSAR	ON	ON
CodeGenerationMode ¹⁾	TargetLink Main Dialog	Standard	Standard
ForceCodeGenOfMisspecifiedVariableWidthsBlocks	Code Generation \ Pattern	OFF	OFF
EnableVariableVectorWidths	Code Generation \ Pattern	OFF	OFF
HandleUnscaledStateflowExpressionsWithTlType	Stateflow	ON	OFF
RequirementInfoAsCodeComment	Code Generation	ON	OFF
EmitMissingRequirementsWarningsAsNotes	General Settings	OFF	OFF
StrictRunnableInterfaceChecks	Code Generation \ AUTOSAR	ON	OFF
StrictTypedefPlacement	Code Generation \ Default Behavior	ON	OFF
CodeGenerationTargetSettings ²⁾	TargetLink Main Dialog	Generic ANSI-C	Generic ANSI-C
UseRootFileNameAsUserDefinedTypesHeaderBasename	Code Compatibility	OFF	ON

¹⁾ Replaces the `EnableMultirate` option, which is now obsolete. Obsolete and changed Code Generator options Possible values for `CodeGenerationMode`: Standard / AUTOSAR / RTOS.

²⁾ Combines the options `TargetName`, `TargetDir` and `Compiler` to accelerate setting multiple options simultaneously.

New code generation modes

The Code generation mode drop-down list lets you specify the code generation mode. The following code generation modes are available:

Value	Description
Standard	Lets you generate production code without any AUTOSAR or RTOS parts.
AUTOSAR	Lets you generate AUTOSAR-compliant code according to the supported AUTOSAR standard.
RTOS	Lets you generate code for multiple sample times.

For further information on the Code Generation Page of the TargetLink Main Dialog, refer to [TargetLink Main Dialog Block](#) ([TargetLink Model Element Reference](#)).

Obsolete and changed Code Generator options	For details on obsolete and changed Code Generator options refer to Migration Aspects Regarding Code Generator Options on page 391.
--	---

Related topics

References

[dsdd_compare_optionset](#) ([TargetLink API Reference](#))
[dsdd_export_optionset](#) ([TargetLink API Reference](#))
[dsdd_import_optionset](#) ([TargetLink API Reference](#))

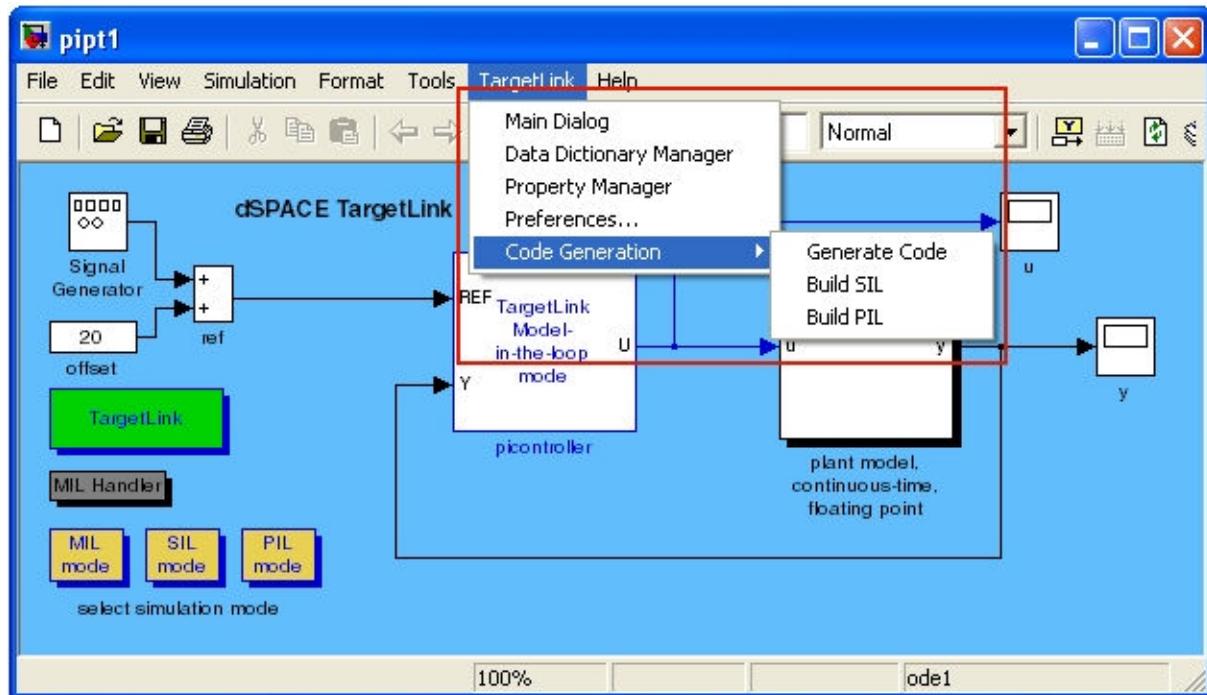
General Enhancements and Changes

Code Changes

TargetLink 3.1 provides enhanced and new features as well as bug fixes that all affect the generated code. The new features can be switched on or off by Code Generator options. The table in [Code Generator Options](#) on page 272 shows the options and their old and new values. To obtain code that is as close as possible to TargetLink 3.0, set the options' values according to the right-hand column.

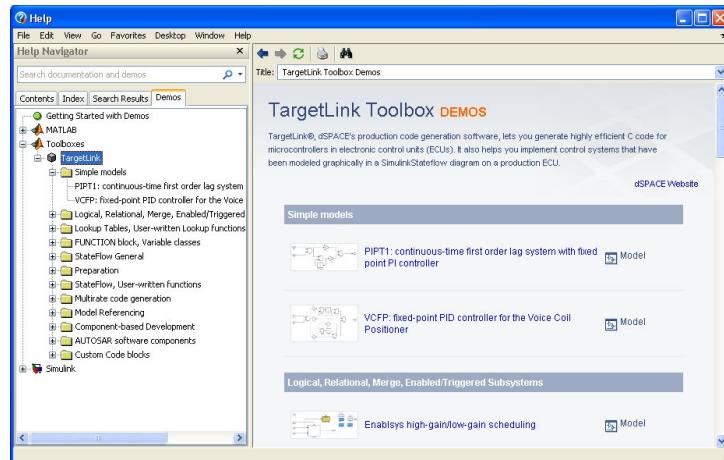
Enlargement of the Simulink menu bar

A set of frequently used TargetLink tools and commands is now accessible via the Simulink menu bar.



Integration of TargetLink Toolbox DEMOS

As of TargetLink 3.1, the TargetLink demo models are integrated in the MATLAB Help Browser. In addition, a short description of the features implemented in the models is given.



Algebraic simplifications

TargetLink 3.1 performs more algebraic optimizations than previous TargetLink versions.

This applies to:

- Bitwise operators (`&`, `|`, `^`, `~`, `<<`, `>>`, `&=`, `|=`)
- Logical operators (`&&`, `||`, `!`, `?:`)
- Arithmetical operators (`+=`, `-=`, `*=`, `/=`, `%`)
- Relations with a Boolean variable and a floating-point constant as operators

Changes in the code result primarily from making use of De Morgan's laws for logical NOT and the corresponding inversion of relations, such as `!(a <= b) => (a > b)`.

Avoiding the generation of multiple log macros

If the vectorized output of a predecessor block of a Switch block is logged, and the Switch block itself has a vectorized condition, the code generated for the predecessor block is not generated into the conditionally executed control flow branch. This avoids the generation of multiple log macros. The same behavior also applies to situations where multiple potential control flow branches exist or where the conditionally executed control flow branch (including the condition) is located within a loop.

Code generated with `EnableBlockDiagramBasedSwitchOptimization = OFF`

As of TargetLink 3.1, the difference between code generated with `EnableBlockDiagramBasedSwitchOptimization = OFF` and `EnableBlockDiagramBasedSwitchOptimization = ON` (default) is less significant than with previous TargetLink versions.

This applies especially to:

- Code with log macros
- Assignments to variables of a variable class whose ERASABLE property is not set.
- The `NoAssignmentOfBooleanExpressions` option when it is set to ON. If `Cleancode = ON` and `NoAssignmentOfBooleanExpressions = OFF` (default) are set, the code generated by TargetLink 3.1 is not different from code generated by an earlier TargetLink version.

Code generated with `HandleUnscaledStateflowExpressionsWithTlType = ON`

The standard settings for `HandleUnscaledStateflowExpressionsWithTlType` can result in code changes originating from Stateflow charts. To be able to apply the code generation rules TargetLink uses for Simulink code, the Stateflow expression autoscale mechanism calculates a result type and a result scaling for each scaled operation specified in a Stateflow chart. A scaled operation is an operation that has at least one operand with an LSB unequal 1.0 and/or an offset unequal 0.0. In previous versions of TargetLink, all unscaled Stateflow expressions were regarded as custom code, i.e., the code specified in the chart was copied to the generated code. As of this version of TargetLink, however, unscaled operations with at least one operand that has different TargetLink and Stateflow data types are now also included in the Stateflow expression autoscale mechanism. This changed behavior usually results in an additional cast operation in the code generated for Stateflow charts and in code that is more similar to code originating from Simulink parts.

Code comments

TargetLink 3.1 emits more code comments than previous TargetLink versions.

This applies especially to:

- Logical and Relation blocks
- Blocks that result in cast operations
- Boundaries of atomic subsystems and inlined functions
- Variables used in custom code that are replaced by corresponding interface variables

In some cases, TargetLink might emit significantly more comments.

Model referencing information in the headers of generated C files

With TargetLink < 3.1, the headers of generated C files which are not part of a TargetLink default module contain subsystem information

```
SUBSYS CORRESPONDING SIMULINK SUBSYSTEM
```

Example:

```
Sa1 pidcontroller
```

and the Stateflow information:

```
SF-NODE CORRESPONDING STATEFLOW NODE DESCRIPTION
```

Example:

```
Ca1 sf_control/Chart chart description
```

In the header of generated C files, TargetLink now also emits information about model blocks and referenced models in the following form:

```
SUBSYS CORRESPONDING MODEL BLOCK (REFERENCED MODEL)
```

Example:

```
Sc0 fuelratecontroller/r_AirflowCalculation (r_AirflowCalculation )
```

You can prevent TargetLink emitting information about model blocks and referenced models by setting the `modelref-block-list-comment` switch (subelement of the `<TL:header-comment>` attribute) in the `cconfig.xml` file to `false`.

Rate Limiter block

In TargetLink 3.1, the Rate Limiter block is able to process vectorized signals.

New AUTOSAR-Related Features

Features of the TargetLink AUTOSAR Module

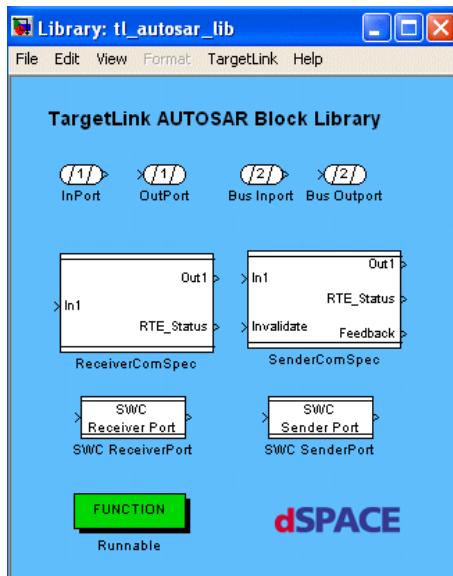
Support of AUTOSAR Release 3.1

The TargetLink AUTOSAR Module supports the following AUTOSAR Releases:

- AUTOSAR Release 3.1 with Version 3.1.0 (new)
- AUTOSAR Release 3.0 with Version 3.0.2 (continued)
- AUTOSAR Release 2.1 with Version 2.1.4 (continued)

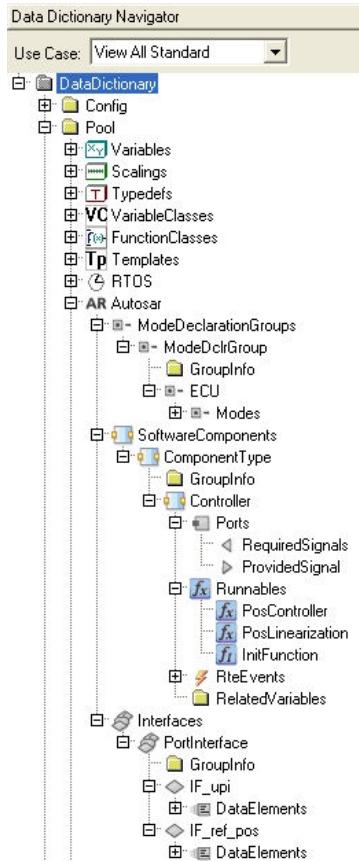
Redesign of the TargetLink AUTOSAR Module

Redesigned AUTOSAR Block Library In TargetLink 3.1 the AUTOSAR Block Library has been redesigned as shown in the illustration below.



Introduction of icons for AUTOSAR objects in the Data Dictionary

AUTOSAR-related DD objects are now visualized with specific Icons as shown in the illustration below.



Improved support of the TargetLink code generator software's standard features

- You can use Simulink models with AUTOSAR software components for prototyping.
- You can use models with AUTOSAR blocks in TargetLink's stand-alone mode.
- You can use AUTOSAR blocks in libraries.
- You can use TargetLink's documentation generation feature for software components.

Improved AUTOSAR import/export

Support of initialization constants You can now specify and import/export constants for initialization:

- For data elements via the InitRef property of the data element's communication specification. This allows you to initialize scalars, arrays, and structs. For instructions on initializing data elements, refer to [How to Model Initialization of Data Elements](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).

- For interruptible variables via the Value property of the variable object. The new properties InitConstantName, InitConstantUuid, and InitConstantPackage allow you to specify import/export settings for the constant. For instructions on creating interruptible variables, refer to [How to Create Interruptible Variables](#) ( [TargetLink Classic AUTOSAR Modeling Guide](#)).
- For calprm elements via the InitRef property of the calprm's communication specification. This allows you to initialize scalars, arrays, and structs.
- For SWC-internal calibration parameters via the Value property of the variable object. The new properties InitConstantName, InitConstantUuid, and InitConstantPackage allow you to specify import/export settings for the constant. If you want to initialize structs, you have to specify the import/export settings at the variable object and the values at the components.

Improved support of arrays TargetLink now supports AUTOSAR-compliant handling of arrays in the generated software component code and exported AUTOSAR file.

Improved support for software components and runnables

- Specialized software component types such as application software component type and sensor actuator software component type as defined by AUTOSAR. The software component type is specified in the new ComponentType property of a software component.
- You can model a software component's initialization and termination runnables with individual names via the new Kind property.
- Software components have new properties for specifying the names and UUIDs of their implementations and internal behaviors.
- Access points and read/write accesses of a runnable for sender-receiver communication, client-server communication, interruptible variables, calibration and measurement, modes, and per instance memories. The access points and read/write accesses are specified as subnodes of the runnable.

Support of admin data You can import/export and specify additional data for each AUTOSAR element as AdminData objects in the Data Dictionary.

Configuration of default import/export settings You can configure the default options of AUTOSAR file import/export by editing the global configuration file, which is located at `%DSpace_ROOT%\dsdd\dsdd_autosar_config.xml`. The options are valid for import/export via dialog and API command.

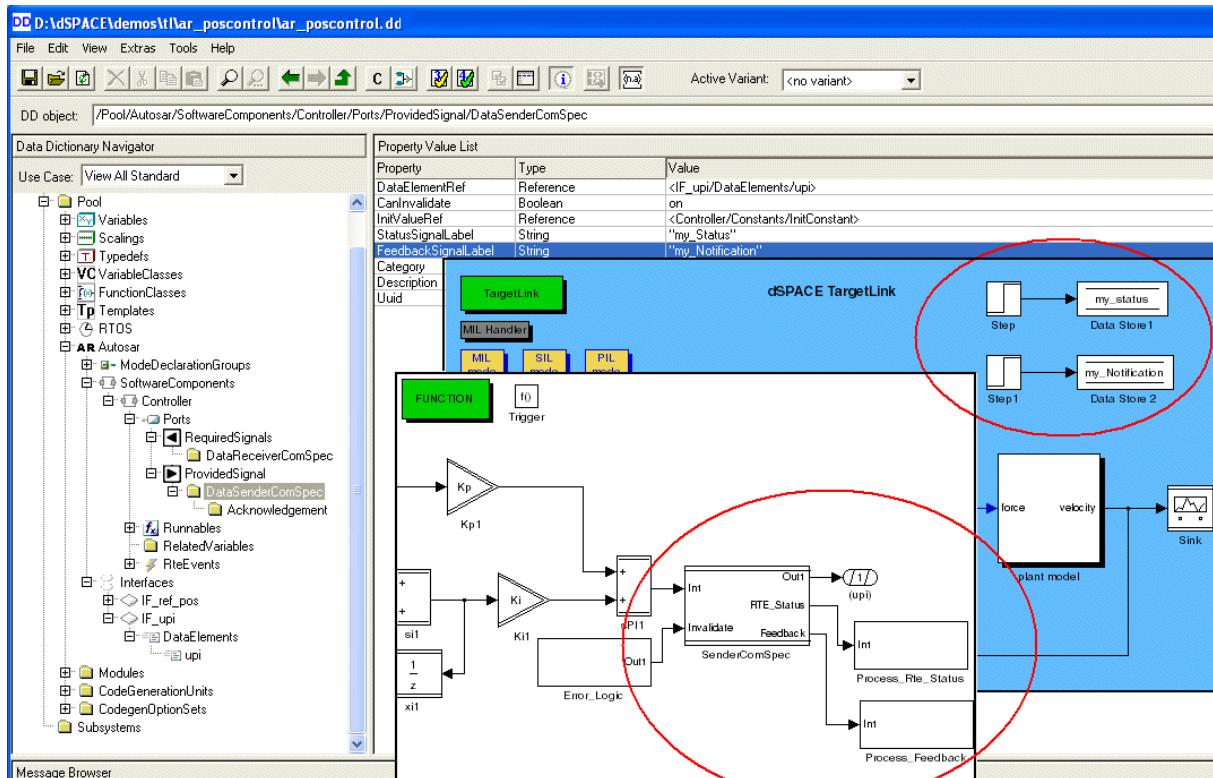
Additionally supported RTE API functions

Code generation is now supported for the following RTE API functions:

- Rte_Invalidate, for invalidating data elements in explicit sender-receiver communication.
- Rte_IlInvalidate, for invalidating data elements in implicit sender-receiver communication.
- Rte_Feedback, for accessing acknowledgement notifications from the RTE when explicitly sending data elements.

- Rte_IStatus, for accessing the RTE status of a data element that is to be read via Rte_IRead.
- Rte_Pim, for accessing per instance memories.

The illustration below shows how you can model sender-receiver communication with invalidation, acknowledgement notifications, and access to the RTE status using a SenderComSpec block of the redesigned AUTOSAR Block Library. The illustration also shows how you can stimulate acknowledgement notifications and the RTE status via Data Store blocks.



For instructions on modeling sender-receiver communication with invalidation, acknowledgement notifications, and access to the RTE status, refer to [How to Model Invalidation, Acknowledgment Notifications and Access to the RTE Status \(TargetLink Classic AUTOSAR Modeling Guide\)](#).

Improved modeling of AUTOSAR software components

Improved support for client-server communication You can now use client-server communication involving operations with composite operation arguments (structs and arrays).

Access to RTE status and acknowledgement notifications in simulations You can stimulate the RTE status of Rte_Send/Rte_Receive RTE API calls, etc., via Simulink Signal objects in simulations, refer to [How to Simulate the RTE Status \(TargetLink Classic AUTOSAR Modeling Guide\)](#). You can also stimulate the acknowledgement notifications that are provided by Rte_Feedback RTE API calls via Simulink Signal objects in simulations, refer to [How to Simulate](#)

Acknowledgment Notifications ( TargetLink Classic AUTOSAR Modeling Guide).

Modeling AUTOSAR and non-AUTOSAR controllers in one model

TargetLink now supports modeling AUTOSAR and non-AUTOSAR control algorithms in one Simulink/TargetLink model. A global option that is available via TargetLink's Main dialog allows you to switch between generating AUTOSAR and non-AUTOSAR code, i.e., standard code. For basic information on working with models for AUTOSAR and non-AUTOSAR controllers, refer to [Basics on Modeling Classic AUTOSAR and non-AUTOSAR Controllers in one Model](#) ( TargetLink Classic AUTOSAR Modeling Guide).

Improved migration of standard TargetLink models to AUTOSAR

AUTOSAR data is now included in Function, InPort, OutPort, Bus Import, and Bus Outport blocks from the TargetLink Block Library. This allows improved migration of standard TargetLink models to AUTOSAR. AUTOSAR-specific blocks are required only for RTE status, acknowledgement notification, and invalidation signals.

You can migrate a standard TargetLink model to AUTOSAR using the TargetLink AUTOSAR Migration Tool, which is available for download via http://www.dspace.de/goto?tl_ar_migration.

New Features of the dSPACE Data Dictionary

dSPACE Data Dictionary 2.0

The dSPACE Data Dictionary 2.0 (DD) has the following new features, enhancements and changes:

Where to go from here

Information in this section

New Key Features.....	283
New and Modified DD MATLAB API Commands.....	284

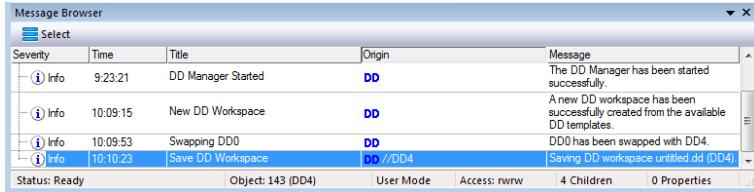
New Key Features

New key features of dSPACE Data Dictionary

Information on the new key features of dSPACE Data Dictionary 2.0 is provided below.

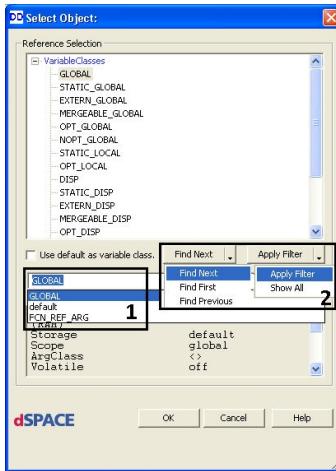
Message Browser

The Message Browser displays all the messages of the Data Dictionary Manager. It can be opened in the View menu.



Enhanced Object Selection dialog

The Object Selection dialog has been enhanced by a history (1) and the Find Next and Apply Filter buttons (2). The Find Next button lets you search for the next DD reference object which matches the specified string. The Apply Filter button lets you apply the string entered in the edit field filter the DD reference objects displayed in the Reference Selection window.



For further information, refer to [How to Reference Data Dictionary Objects](#) ([TargetLink Data Dictionary Basic Concepts Guide](#)).

Improved drag and drop handling in the Data Dictionary Navigator

As of TargetLink 3.1, you can see where an object will be dropped. You can also drop an object at an arbitrary place within a group, which was not the case with previous versions.

Loading data for expanded subtrees

When you click Refresh, the DD Navigator does not load all the data from the DD, but only the data for the expanded subtrees. This boosts performance with large data dictionaries (especially ones that have extensive subsystems).

New and Modified DD MATLAB API Commands

New DD MATLAB API commands: SetCurWidth / GetCurWidth

For direct access to the `Width` property, the Data Dictionary MATLAB API provides the new `SetCurWidth` and `GetCurWidth` commands. Their behaviors are identical to the old behaviors of `GetWidth` and `SetWidth`.

General Migration Information

Upgrading Models, Libraries, and Data Dictionaries

Where to go from here

Information in this section

Basics on Migrating Between TargetLink Versions.....	286
How to Upgrade a Data Dictionary with Included DD Files.....	289
How to Manually Upgrade Libraries and Models via the API.....	290
Migrating Data Dictionaries to CodeDecorationSets.....	291

Basics on Migrating Between TargetLink Versions

Automatic upgrade from TargetLink 3.1 or later

TargetLink 5.1 automatically upgrades models and TargetLink-compliant libraries if they were created with TargetLink 3.1 or later.

The automatic upgrade includes all the steps required by the intervening TargetLink versions. For example, an automatic upgrade from TargetLink 4.0 to TargetLink 5.1 comprises the steps 4.0 to 4.1 to 4.2 to 4.3 to 4.4.

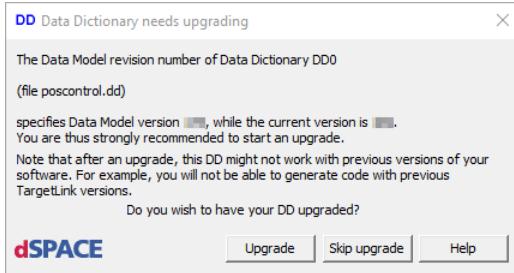
Note

Check the TargetLink migration documentation of the different TargetLink versions to see whether user interaction is required.

- User interaction required** In the following cases, for example, the automatic upgrade requires additional user interaction:
- Libraries must be TargetLink-compliant. Otherwise, no upgrade can be performed.
 - Style sheets for code generation are version-specific and subject to change from one TargetLink version to another. Thus, modified style sheets of older TargetLink versions have to be updated to match the current version (reapplying the modifications as intended). Refer to [Basics on Code Formatting](#) ([TargetLink Customization and Optimization Guide](#)).
 - Custom code S-functions built with 32-bit TargetLink versions do not work with 64-bit versions of TargetLink.
Initiate a rebuild of all custom code S-functions using the `t1Upgrade('Model',<MyModel>, 'CheckModel', 'FixIssues')` API function.

Upgrading Data Dictionaries

When you open a Data Dictionary created with a TargetLink version 2.x or later with TargetLink 5.1 for the first time, you are prompted for the upgrade. For example:



To upgrade DD files with included DD files, refer to [How to Upgrade a Data Dictionary with Included DD Files](#) on page 289.

Making new libraries TargetLink-compliant

Libraries that you create from scratch and that consist of TargetLink blocks must be made upward compatible so that you can upgrade them to a newer TargetLink version in the future. Otherwise, no upgrade can be performed.

Note

A library does not automatically become a TargetLink library if it contains TargetLink blocks. The library itself must be TargetLink-compliant.

Refer to [How to Make User Libraries Upgrade-Capable](#) ([TargetLink Orientation and Overview Guide](#)).

Making existing libraries TargetLink-compliant

The following two approaches let you make libraries created with earlier TargetLink versions compliant with the current TargetLink version 5.1:

The earlier TargetLink version is available Use the TargetLink version with which the library was created to make the library TargetLink-compliant. Refer to the TargetLink documentation of the earlier TargetLink version. You can then use this library with all later TargetLink versions because TargetLink automatically performs an upgrade. The library can still be used with TargetLink versions earlier than TargetLink 5.1 because the automatic upgrade does not save a library in the newer TargetLink version.

Only the current TargetLink version 5.1 is available Use TargetLink version 5.1 and the `t1Upgrade` API command to make the library TargetLink-compliant. Refer to [How to Manually Upgrade Libraries and Models via the API](#) on page 290. If you follow the instructions, the library is saved in TargetLink version 5.1. Therefore, it cannot be used with TargetLink versions earlier than TargetLink 5.1.

Manual upgrade from TargetLink 2.x or 3.0.x

Models and libraries created with TargetLink versions 2.x or 3.0.x have to be upgraded manually to the latest TargetLink version 3.x (3.1...3.5) you have. Afterwards, automatic upgrade is possible.

No backward compatibility

You cannot use models, libraries and Data Dictionaries in the format of newer TargetLink versions in earlier TargetLink versions.

Data model filter rule files

Existing data model filter rule files can contain invalid elements because the data model of the TargetLink Data Dictionary changed. The following files that were shipped with previous TargetLink versions can be affected:

- `DD_Filter_Admin.xml`
- `DD_Filter_AR_User.xml`
- `DD_Filter_NonAR_NonRTOS_User.xml`

You can check filter rule files via the API in the MATLAB Command Window:

Checking a Single File	Checking Filter Rule Sets ¹⁾
<code>dsdd_free;</code>	<code>dsdd_free;</code>
<code>dsdd('ReadFilterRuleSet', 'file', '<myFile>.xml');</code>	<code>dsdd('ReloadFilterRuleSets');</code>
<code>ds_error_register(dsdd('GetMessageList'));</code>	<code>ds_error_register(dsdd('GetMessageList'));</code>
<code>ds_msdlg('update');</code>	<code>ds_msdlg('update');</code>

¹⁾ All the files contained in the directory defined in Data Dictionary - Filter Rules in the Preferences Editor.

TargetLink informs you about errors in the TargetLink Message Browser. Each error contains the following information so that you can fix it in any XML-capable editor:

- File name
- Row number
- Column number

Related topics

Basics

[Basics on Code Formatting \(📖 TargetLink Customization and Optimization Guide\)](#)

HowTos

[How to Make User Libraries Upgrade-Capable \(📖 TargetLink Orientation and Overview Guide\)](#)
[How to Manually Upgrade Libraries and Models via the API..... 290](#)

References

[tlUpgrade \(📖 TargetLink API Reference\)](#)

How to Upgrade a Data Dictionary with Included DD Files

Precondition

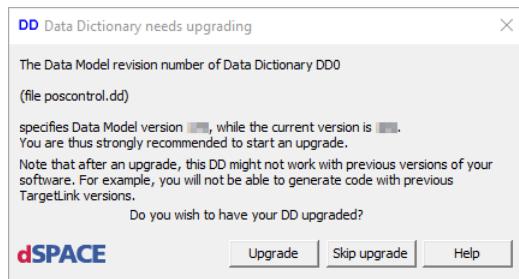
In the main DD file to be loaded, the AutoLoad property of the DDIncludeFiles objects is set to on.

Method

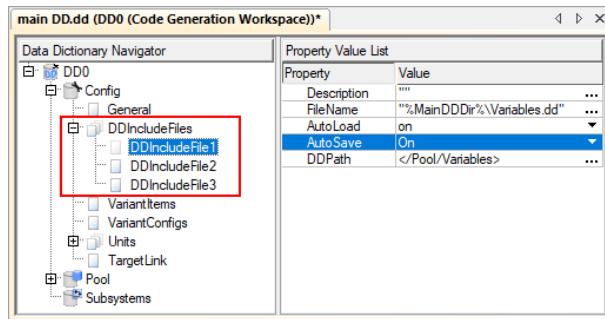
To upgrade a Data Dictionary with included DD files

- 1 Open the Data Dictionary Manager and the main DD via File - Open.

The Data Dictionary needs upgrading dialog automatically opens if an earlier data model revision is used.



- 2 Select Upgrade. The main DD and all included DD files are upgraded to the latest data model revision.
- 3 Set the AutoSave property of all the /Config/DDIncludeFiles/<DDIncludeFile> objects to on to save all included DD files together with the main DD.



If you do not want to save all included DD files together with the main DD, specify their AutoSave property as required.

- 4 Save the main DD file.

Result

You upgraded the main DD file and all included DD files. TargetLink adjusted the revision number of every upgraded DD file to the latest data model revision.

Related topics	Basics
	<p>Basics on Opening and Handling DD Files (TargetLink Data Dictionary Basic Concepts Guide)</p>
	<p>HowTos</p>
	<p>How to Include Partial Data Dictionary Files (TargetLink Data Dictionary Basic Concepts Guide)</p>
	<p>References</p>
	<p>Point of Inclusion (TargetLink Data Dictionary Manager Reference)</p>

How to Manually Upgrade Libraries and Models via the API

Objective	To prepare a central upgrade of libraries and models in a tool chain scenario with several users, for example.
Preconditions	<p>The model or library files are available on the MATLAB search path but they are not open.</p> <p>The required and upgraded DD project file has been opened, for example, via <code>dsdd_manage_project('Open', '<name>.dd')</code>. DD project files can be upgraded via <code>dsdd('Upgrade' [,<DD_Identifier>])</code>.</p>
Method	<p>To manually upgrade libraries and models via the API</p> <p>1 Type the following API command in the MATLAB Command Window:</p> <pre>tluUpgrade('Model', '<Model Library>.slx', 'CheckModel','FixIssues')</pre> <p>The model or library is upgraded.</p> <p>Note</p> <p>When you upgrade models and libraries, first upgrade models or libraries that do not reference any other libraries, i.e., the blocks and subsystems they contain have no links to other libraries. Start with the bottom library and then upgrade the libraries above it in ascending order.</p> <p>2 Save the upgraded model or library files, e.g., <code>Library.slx</code>.</p> <p>3 Repeat steps 1 and 2 for all other models or libraries.</p>

Result You upgraded the models and libraries.

Related topics References

[tlUpgrade \(TargetLink API Reference\)](#)

Migrating Data Dictionaries to CodeDecorationSets

Introduction of CodeDecorationSet and CodeDecoration objects

Since TargetLink 4.3 DD CodeDecorationSet and CodeDecoration objects are introduced.

Additionally, several properties were removed from the Data Dictionary data model:

DD Object	Change	Replacement
FunctionClass	Removal of the DeclarationStatements and SectionName properties.	The DeclarationStatements and SectionName properties of the DD CodeDecoration.Settings object.
VariableClass		
VariableClassTemplate.Filter	Removal of the WidthSpec property.	The WidthSpec property of the DD CodeDecoration.Filter object.

Automatic upgrade by TargetLink

Limitation TargetLink no longer supports width-specific type prefixes for variable classes. The automatic upgrade of the Data Dictionary fails if the original Data Dictionary contains variable class templates used to derive variable classes that have width-specific type prefixes.

Use declaration statements instead.

When you open a Data Dictionary whose data model is older than the latest revision, TargetLink prompts you to perform an automatic upgrade.

Object Kind	Trigger	Upgrade Action
VariableClass	DeclarationStatements or SectionName properties are set.	1. Creating a DD CodeDecorationSet object. 2. Creating a single DD CodeDecoration object for each DD CodeDecorationSet object. The settings of the CodeDecoration object and its child objects match the settings of the original objects. 3. Referencing the CodeDecorationSet object at the original object.
FunctionClass		
SubStructTemplate	Filter.VariableClass is set.	Transfer the values of the following properties from the variable class to the SubStructTemplate object's filter: <ul style="list-style-type: none">▪ DeclarationStatements▪ SectionName▪ TypePrefix

Object Kind	Trigger	Upgrade Action
VariableClassTemplate	<ul style="list-style-type: none"> ▪ Filter.FilterCondition property is set to ALL_TRUE. ▪ Settings.VariableClass references a DD VariableClass object whose DeclarationStatements or SectionName properties are set. ▪ The Filter.WidthSpec property is set for this DD VariableClassTemplate object or for another VariableClassTemplate object whose Filter.VariableClassSpec property has the same value. 	<ol style="list-style-type: none"> 1. Create a new DD VariableClass object in /Pool/VariableClasses/Templates. 2. Create a new DD CodeDecorationSet object in /Pool/CodeDecorations/Templates. 3. For each VariableClassTemplate object with the same value at the Filter.VariableClassSpec property, adding a CodeDecoration object to the CodeDecorationSet object. 4. Specifying the CodeDecoration object as required. 5. Referencing the CodeDecorationSet object at the VariableClass object created in step 1. 6. Referencing the VariableClass object created in step 1 via the VariableClassTemplate.Settings.VariableClass property.

Special considerations for variable class templates

If you specified DD VariableClassTemplate objects whose Filter.FilterCondition property is set to ALWAYS or NEVER, TargetLink deletes the object's Filter.WidthSpec property during the upgrade without replacement.

If you want to keep the property value, set the DD VariableClassTemplate object's Filter.FilterCondition property to ALL_TRUE before upgrading the Data Dictionary.

Limitation TargetLink does not upgrade DD VariableClassTemplate objects whose Filter.FilterCondition property is set to ONE_OR_MORE or ALL_FALSE.

Cleaning

The automatic upgrade retains the functionality that was specified in the previous Data Dictionary. You can clean it manually to reduce the number of objects in the new Data Dictionary.

Merging width-specific variable classes If the previous Data Dictionary contained width-specific VariableClassTemplate/VariableClass objects, the new Data Dictionary still contains all these variable classes.

Because the width-specific information is now stored in DD CodeDecoration objects, you can manually reduce the number of VariableClass objects in the Data Dictionary. For example, if you used variable classes in the form of <Name>_<Width>, you can replace them by a single <Name> variable class that references a suitable code decoration set.

Two methods are possible:

- Merging code decoration sets:

1. Copy all the DD CodeDecoration objects that were generated during the upgrade for each variable class called <Name>_<Width> to a single CodeDecorationSet object.
2. Make each CodeDecoration object width-specific via its filter.
3. Reference the resulting CodeDecorationSet object at the <Name> variable class.

- Using a code decoration set created for variable class templates:
 1. If the original <Name>_<Width> variable classes were referenced by variable class templates, the DD upgrade automatically creates a width-specific code decoration set in /Pool/CodeDecorationSets/Templates.
 2. You can reference this code decoration set at the resulting variable class called <Name>.

Note

Replace references from model elements to the variable classes called <Name>_<Width> with references to <Name>.

Retarget variable class templates After you merged the previous width-specific variable classes, you can use them again as the target of the variable class templates. You can then delete all the variable classes contained in /Pool/VariableClasses/Templates that were created during the upgrade.

Simplifying user-specified scope reduction chains (SRC) If you used a user-specified SRC to specify declaration statements or section names for variables with specific scopes, you can do the following:

1. Adjust the Filter.ScopeSpec property of the code decoration that belongs to the set referenced by the first variable class in the SRC (highest scope) as required.
2. Delete the other variable classes of the SRC.
3. If you also used the SRC to prevent static local variables, you can now use the AvoidStaticLocalScope Code Generator option instead.

Remove obsolete variable class templates Find DD VariableClassTemplate objects with the same value of the Filter.VariableClassSpec property and delete all but one.

Changes in the generated production code

Changes in CodeDecoration objects can influence the generated production code mainly in the following respects:

- [Changed code comments](#) on page 505
- [Sorting of variable definitions](#) on page 506

Refer to [Code Changes Between TargetLink 4.3 and TargetLink 4.4](#) on page 470.

Migration Steps

Where to go from here

Information in this section

Migrating from TargetLink 5.0 to 5.1.....	296
Migrating from TargetLink 4.4 to 5.0.....	304
Migrating from TargetLink 4.3 to 4.4.....	314
Migrating from TargetLink 4.2 to 4.3.....	318
Migrating from TargetLink 4.1 to 4.2.....	328
Migrating from TargetLink 4.0 to 4.1.....	337
Migrating from TargetLink 3.5 to 4.0.....	343
Migrating from TargetLink 3.4 to 3.5.....	359
Migrating from TargetLink 3.3 to 3.4.....	365
Migrating from TargetLink 3.2 to 3.3.....	368
Migrating from TargetLink 3.1 to 3.2.....	376
Migrating from TargetLink 3.0 to 3.1.....	387

Migrating from TargetLink 5.0 to 5.1

Where to go from here

Information in this section

Code Generator Options.....	296
AUTOSAR.....	298
Custom Code.....	299
API Functions and Hook Scripts.....	299
Messages.....	300
Reserved Identifiers.....	301
Other Migration Aspects.....	302
Optimization.....	303

Code Generator Options

Migration Aspects Regarding Code Generator Options

User interface

The OptimizedBoolType Code Generator option has been removed from the Advanced page of the TargetLink Main Dialog block. It is still available via the All options... button on this page.

Related documentation

- [OptimizedBoolType \(TargetLink Model Element Reference\)](#)

Basics on changed defaults

The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code Generator options in DD CodegenOptionSet objects (DD-based option storage). You can use DD CodegenOptionSet objects as a central source for overwriting and replacing the model-based option settings that were used since TargetLink 4.1.

If a model-based option value equals the old default value, it is automatically changed to the new default value during the upgrade. If a DD-based option value equals the old default value, it is not changed to the new default value during the upgrade but keeps the old value.

Option value = old default If Code Generator options were set to default values in the earlier TargetLink version, and the new TargetLink version uses modified default values, note the following points:

- Model-based option:

If you want to keep the old default values, you must reset them manually.

- DD-based option:

If you want to use the new default values, you must adjust them manually.

The following table is an example describing the impact of a TargetLink upgrade (TargetLink_{Old} to TargetLink_{New}) on three option values: 9, 11, and 13. The table illustrates two basic migration scenarios:

- Scenario #1: New default = old default

The default value of a Code Generator option has not changed in the new TargetLink version, i.e., the default value remains 9.

None of the option values is changed.

- Scenario #2: New default ≠ old default

The default value of a Code Generator option changed with the new TargetLink version, i.e., the default value changed to 11.

Option Storage	Option Value (TargetLink _{Old})	Option Value (\leq TargetLink _{New})	
		Default = 9 (Scenario #1)	Default = 11 (Scenario #2)
Model-based	9 ¹⁾	9 ¹⁾	11 ²⁾
	11	11	11 ¹⁾
	13	13	13
DD-based	9	9	9 ³⁾
	11	11	11
	13	13	13

¹⁾ The option value is not stored with the model because it equals the default.

²⁾ Manual reset might be necessary.

³⁾ Manual adjustment might be necessary.

Option value = new default If the Code Generator options were not set to default values in the former TargetLink version (A) but are in the new TargetLink version (B), TargetLink assumes that you intentionally specified the default value in the new TargetLink version. The same applies if the default changes again in the next TargetLink version (C).

Note

Upgrading TargetLink_A \Rightarrow TargetLink_B \Rightarrow TargetLink_C and upgrading TargetLink_A \Rightarrow TargetLink_C can cause different option values. Refer to the following table.

If the default values for TargetLink versions A, B, and C are 9, 11, and 13, and an option was set to 11 in version A, an upgrade to version C changes the option value as follows:

Upgrade Strategy	Option Value TargetLink _A Default = 9	Option Value TargetLink _B Default = 11	Option Value TargetLink _C Default = 13
A ⇒ B ⇒ C	11 (≠ default)	11 (= default) ¹⁾	13 (= default) ¹⁾
A ⇒ C	11 (≠ default)	—	11 (≠ default)

¹⁾ The option value is not stored with the model because it equals the default.

-
- New Code Generator options** For more information on new Code Generator options, refer to [New Code Generator Options](#) on page 37.

AUTOSAR

Migration Aspects Regarding AUTOSAR

-
- Syntax of AUTOSAR versions** The syntax for the specification of the AutosarVersion property of DD Config objects found under `/Pool/Autosar` has been adjusted to the naming convention of AUTOSAR: From AUTOSAR Standard R19-11 onward, the AUTOSAR Standard RYY-MM is specified as YY-MM.

With AUTOSAR Standard R19-11 `Rte_TransformerError` changes into `Std_TransformerError`.

Related documentation:

- [Basics on Data Transformation](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))

-
- Optional DD RteEvents and DD Runnables objects** With TargetLink 5.1, the following child objects of DD SoftwareComponent objects are optional:
- DD RteEvents object
 - DD Runnables object

These objects will no longer be automatically created when a DD SoftwareComponent object is created. Because neither RteEvents nor Runnables are relevant in Adaptive AUTOSAR, the related objects are superfluous in this context.

Custom scripts that create DD SoftwareComponent objects and expect the existence of the now optional child objects must be adjusted accordingly.

Existing objects and DD projects remain unchanged.

Custom Code

Migration Aspects Regarding Custom Code

tl_types.h no longer generated

The `tl_types.h` header file is no longer generated. If your custom code uses the TargetLink base types and contains an include statement for `tl_types.h`, replace it as follows:

Code Generation Mode	Replacement
Standard/RTOS	Include statement for the <code>tl_basetypes.h</code> header file.
Classic AUTOSAR	Include statement for the <code>Rte_Types.h</code> header file.
Adaptive AUTOSAR	Replace the TargetLink base types with the fixed-width data types of Adaptive AUTOSAR, such as <code>int8_t</code> .

API Functions and Hook Scripts

Changes in TargetLink and TargetLink Data Dictionary API Functions

tlCodeCoverage

New commands

Command	Purpose
GenerateCombinedReport (refer to <code>tlCodeCoverage('GenerateCombinedReport', propertyName, PropertyValue, ...)</code> (TargetLink API Reference))	Generates a combined report from all CTC files in and below the specified folders. Refer to How to Measure Code Coverage Over Multiple Simulation Application Builds via Third-Party Tools (Testwell CTC) (TargetLink Preparation and Simulation Guide).
MoveCTCFiles (refer to <code>tlCodeCoverage('MoveCTCFiles', propertyName, PropertyValue, ...)</code> (TargetLink API Reference))	Moves CTC files created during the build process and simulation to the specified folder. These files can later be used to create a combined report of code coverage analysis for multiple builds and simulations. By default, each time an application is (re)built, the existing CTC files are moved from the TargetLink build folder to the <code>.\CTCRESULTS\TLBuild_<BuildTimeStamp></code> folder.

tldoc **Removed properties** The **Encoding** property value pair is removed. UTF-8 encoding is used.

tl_generate_vecu_implementation **Removed properties** The **ExportFixedPointLibrary** property value pair is removed because it is no longer required.
Refer to [Improved Container Export](#) on page 31.

tlRebuildFixedPointLibrary

New properties

Property	Description
Assembler	Code generation target setting. Default: 'off'
CodeOpt	Code generation target setting. Default: 'Generic ANSI-C'

New output parameters

Parameter	Description
bError	Error flag
msgList	Error message

Removed output parameters The following output parameters have been removed:

- **succeededNr**
- **failedNr**
- **upToDateNr**

Messages

Changes in TargetLink Messages

A20449

The following messages changed their type to **Warning** in TargetLink 5.1 because of changes in the semantics of the **Optimization** property of DD VariableClass objects:

Old Message Number	New Message Number
A20449	W20449

For more information on the changed semantics, refer to [Migration Aspects Regarding Optimization](#) on page 303.

Reserved Identifiers

Migration Aspects regarding reserved identifiers

New reserved identifiers

With TargetLink 5.1, a number of prefixes in identifiers are reserved by TargetLink. The use of these identifiers leads to messages of different severity:

Identifier Use	Used Prefix	Result
▪ Function name ▪ Variable name ▪ Macro ▪ Enum value ▪ Type name	▪ tl_dsfpxp_ ▪ tl_sim_ ▪ tl_std_	Error
▪ Struct tag ▪ Struct component	▪ tl_	Warning
		Advice

Adjust the models accordingly.

Related documentation

- [Basics on TargetLink Base Types, Typedefs, and Header Files](#) ( [TargetLink Customization and Optimization Guide](#))

Reserved identifiers for limit macros

With TargetLink 5.1, additional identifiers for macros that represent floating-point limits are reserved:

tl_FLOAT32MAX	tl_sim_FLOAT32MAX	tl_dsfpxp_FLOAT32MAX
tl_FLOAT32NMIN	tl_sim_FLOAT32NMIN	tl_dsfpxp_FLOAT32NMIN
tl_FLOAT64MAX	tl_sim_FLOAT64MAX	tl_dsfpxp_FLOAT64MAX
tl_FLOAT64NMIN	tl_sim_FLOAT64NMIN	tl_dsfpxp_FLOAT64NMIN

In addition, the identifiers for fixed-point limit macros for the Fixed-Point Library and the simulation frame are reserved as well. These identifiers consist of one of the prefixes `tl_sim_` or `tl_dsfpxp_`, the base type, and one of the MAX or NMIN suffixes, e.g., `tl_dsfpxp_INT8MAX`.

Adjust the model accordingly.

Related documentation

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

Other Migration Aspects

Various Migration Aspects

DD ModuleTemplate objects	<p>With TargetLink 5.1, DD ModuleTemplate objects have been removed.</p> <p>If you used DD ModuleTemplate objects to change the file name extension of modules for Adaptive AUTOSAR modules to <code>cpp</code>, you can now set the programming language of a module via the Language property of the DD ModuleInfo object.</p> <p>By default, the Language property sets the programming language of modules depending on the code generation mode. For Adaptive AUTOSAR, it is C++, otherwise C. You can define the default file name extension for header files and source files containing C++ code via the following Code Generator options:</p> <ul style="list-style-type: none">▪ DefaultCppHeaderFileExtension ( TargetLink Model Element Reference)▪ DefaultCppSourceFileExtension ( TargetLink Model Element Reference)
TargetLink Autodoc Customization block	<p>With TargetLink 5.1, the Autodoc Customization block that was introduced with TargetLink 3.3 (green) is the only Autodoc Customization block that is supported. This block is available via the TargetLink block library.</p> <p>The Autodoc Customization block from TargetLink versions earlier than TargetLink 3.3 (yellow) is no longer supported and must be replaced.</p>
Void BaseTypeRename	<p>With TargetLink 5.1, the <code>Void</code> TargetLink base type can be renamed only via a <code>BaseTypeRename</code> if the <code>CodedType</code> property in the <code>TargetConfig.xml</code> for the selected code generation target is not specified as <code>Use standard C void type</code>.</p>
Leaf struct components at Data Store Memory blocks	<p>If you reference a leaf struct component of a struct variable whose <code>Scope</code> property is set to <code>ref_param</code>, TargetLink now displays E20009.</p> <p>With TargetLink versions earlier than 5.1, the error was displayed only if a plain variable or a struct variable whose <code>Scope</code> property was set to <code>ref_param</code> was referenced at a Data Store Memory block.</p>
Updating version-specific style definition file and style sheets	<p>During code generation, TargetLink 5.1 checks if the version of the used code output style definition file and code output style sheets matches the current TargetLink version to avoid unexpected behavior when using old versions of these files. If it does not match, TargetLink displays an error. Compare and update your currently used code output style definition file and code output style</p>

sheets, e.g., via a diff and merge tool, with the original files provided by TargetLink 5.1. In addition, make sure that the correct TargetLink version entry is specified.

Related documentation

- [Basics on Code Formatting](#) ( [TargetLink Customization and Optimization Guide](#))

Optimization

Migration Aspects Regarding Optimization

Optimization

If you set the Optimization property at the DD VariableClass object to MOVABLE, ERASABLE, or SCOPE_REDUCIBLE, you guarantee that there are no unknown variables accesses, i.e., there are no variable accesses outside of the CGU or variable accesses from external functions unless they have been specified at the interfaces.

Related documentation:

- [Basics on Optimizing Variables](#) ( [TargetLink Customization and Optimization Guide](#))

Migrating from TargetLink 4.4 to 5.0

Where to go from here

Information in this section

Code Generator Options.....	304
AUTOSAR.....	307
Custom Code.....	308
API Functions and Hook Scripts.....	309
Messages.....	310
Other Migration Aspects.....	311

Code Generator Options

Migration Aspects Regarding Code Generator Options

Removed Code Generator options

The following Code Generator options were removed from TargetLink:

Removed Option	Additional Information
DisableFunctionsAsAnalysisBoundaries	-
ShiftMode	As a replacement, use the new UtilizeBitwiseShiftOperations ( TargetLink Model Element Reference) option.
SideEffectFreeAnalysisThreshold	-
TreatAllForcedAtomicSubsystemsAsWeakAtomic	-
UseSfDataAsStateFlags	As a replacement, you can use active state data with child activity mode. Refer to Basics on Working with Active State Data ( TargetLink Preparation and Simulation Guide).

Changed Code Generator options

Code Generator Option	Old Default	New Default
None	-	-

Recommended compatibility settings for UtilizeBitwiseShiftOperations

ShiftMode option	UtilizeBitwiseShiftOperations option ¹⁾	Additional Information
0 (Automatic)	1 (Automatic)	-
1 (Don't shift right)	4 (AvoidAnyNonPortableShiftsBasedOnSignedness)	Differences to ShiftMode: ▪ Right-shifts are allowed for operands with unsigned data types.
17 (Don't shift right, keep explicit shift operations)		▪ Left-shifts are restricted to operands with unsigned data types.
2 (Don't shift left)	2 (AvoidPotentiallyUndefinedShiftsBasedOnSignedness)	Differences to ShiftMode: ▪ Left-shifts are allowed for operands with an unsigned data type. This is the default setting.
18 (Don't shift left, keep explicit shift operations)		
3 (Don't shift at all)	6 (Never)	If you want to avoid undefined or implementation-defined behavior, you can use option 4 as well.
19 (Don't shift at all, keep explicit shift operations)		

¹⁾ Explicitly modeled shift operations are taken into account. A message of the `Advice` type is displayed if an explicit shift operation does not comply with the Code Generator option.

With TargetLink 5.0, signed operands are no longer left-shifted by default. You can either allow TargetLink to utilize range information or set the `UtilizeBitwiseShiftOperations` option to `Automatic`. This extends or generally allows for the use of left shifts on signed operands. Refer to [Suppression of Shift Operations](#) ( [TargetLink Preparation and Simulation Guide](#)).

Note

The change in default behavior and semantics between the Code Generator options `ShiftMode` and `UtilizeBitwiseShiftOperations` leads to changes in the production code. Refer to [Shift Operations](#) on page 456.

Other Compatibility Settings

CombineControlFlowStatements Set the new `CombineControlFlowStatements` Code Generator option to `Off` to ensure downward compatibility.

Basics on changed defaults

The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code

Generator options in DD CodegenOptionSet objects (DD-based option storage). You can use DD CodegenOptionSet objects as a central source for overwriting and replacing the model-based option settings that were used since TargetLink 4.1.

If a model-based option value equals the old default value, it is automatically changed to the new default value during the upgrade. If a DD-based option value equals the old default value, it is not changed to the new default value during the upgrade but keeps the old value.

Option value = old default If Code Generator options were set to default values in the earlier TargetLink version, and the new TargetLink version uses modified default values, note the following points:

- Model-based option:

If you want to keep the old default values, you must reset them manually.

- DD-based option:

If you want to use the new default values, you must adjust them manually.

The following table is an example describing the impact of a TargetLink upgrade (TargetLink_{Old} to TargetLink_{New}) on three option values: 9, 11, and 13. The table illustrates two basic migration scenarios:

- Scenario #1: New default = old default

The default value of a Code Generator option has not changed in the new TargetLink version, i.e., the default value remains 9.

None of the option values is changed.

- Scenario #2: New default ≠ old default

The default value of a Code Generator option changed with the new TargetLink version, i.e., the default value changed to 11.

Option Storage	Option Value (TargetLink _{Old})	Option Value (≤ TargetLink _{New})	
		Default = 9 (Scenario #1)	Default = 11 (Scenario #2)
Model-based	9 ¹⁾	9 ¹⁾	11 ²⁾
	11	11	11 ¹⁾
	13	13	13
DD-based	9	9	9 ³⁾
	11	11	11
	13	13	13

¹⁾ The option value is not stored with the model because it equals the default.

²⁾ Manual reset might be necessary.

³⁾ Manual adjustment might be necessary.

Option value = new default If the Code Generator options were not set to default values in the former TargetLink version (A) but are in the new TargetLink version (B), TargetLink assumes that you intentionally specified the default value in the new TargetLink version. The same applies if the default changes again in the next TargetLink version (C).

Note

Upgrading TargetLink_A ⇒ TargetLink_B ⇒ TargetLink_C and upgrading TargetLink_A ⇒ TargetLink_C can cause different option values. Refer to the following table.

If the default values for TargetLink versions A, B, and C are 9, 11, and 13, and an option was set to 11 in version A, an upgrade to version C changes the option value as follows:

Upgrade Strategy	Option Value TargetLink_A Default = 9	Option Value TargetLink_B Default = 11	Option Value TargetLink_C Default = 13
A ⇒ B ⇒ C	11 (≠ default)	11 (= default) ¹⁾	13 (= default) ¹⁾
A ⇒ C	11 (≠ default)	—	11 (≠ default)

¹⁾ The option value is not stored with the model because it equals the default.

-
- | | |
|-----------------------------------|---|
| New Code Generator options | For more information on new Code Generator options, refer to New Code Generator Options on page 53. |
|-----------------------------------|---|

AUTOSAR

Where to go from here	Information in this section
	AUTOSAR 2.x/3.x no Longer Supported 307
	Changed Block Property for AUTOSAR Mode 308

AUTOSAR 2.x/3.x no Longer Supported

TargetLink no longer supports code generation for AUTOSAR 2.x/3.x. If you want to use TargetLink 5.0 to generate code for an old model, you must set the **DD /Pool/Autosar/Config** property from **2.x/3.x** to **>= 4.0**. This results in code changes.

If this code is to be compliant with AUTOSAR 4.x, adjust your project accordingly. If you want to create application data types, TargetLink assists you with its **ApplicationDataType Creation Wizard**.

Related documentation

- [Supported AUTOSAR Releases \(TargetLink Interoperation and Exchange Guide\)](#)
- [Code Changes Between TargetLink 4.4 and TargetLink 5.0 on page 456](#)
- [How to Create Application Data Types for Existing Implementation Data Types \(Classic AUTOSAR\) \(TargetLink Classic AUTOSAR Modeling Guide\)](#)

Changed Block Property for AUTOSAR Mode

With the support of Adaptive AUTOSAR, the `autosar.useautosarcommunication` block property was replaced by the `autosarmode` block property. This is shown in the following table:

TargetLink Version	Block Property	Data type
≤ 4.4	<code>autosar.useautosarcommunication</code>	<code>Boolean</code> (TargetLink API Reference)
5.0	<code>autosarmode</code>	<code>AutosarModeEnum</code> (TargetLink API Reference)

Adapt your user scripts and tool chain accordingly.

Custom Code

Dereferencing of Pointers

New behavior

With prior TargetLink versions it could be required to manually dereference pointers in custom code file sections. This no longer is the case: TargetLink now automatically determines whether a pointer in a custom code file section must be dereferenced by analyzing the context in which the Custom Code block resides.

Required migration of manually dereferenced pointers

If you used specialized custom code files whose custom code sections contain manually dereferenced pointers, you have to adjust these files to TargetLink's new behavior.

The following table show cases that have to be adapted to prevent false code (double dereferencing):

Expression in Custom Code File Section	Pointer	TargetLink ≤ 4.4	TargetLink 5.0
y = u + *p;	Sa8_CC_param	Sa8_CC_out = Sa8_CC_in + *Sa8_CC_param;	Sa8_CC_out = Sa8_CC_in + (*Sa8_CC_param); ¹⁾
x = p->x0 * u;	pPar	Sb3_CC_x = pPar->x0 * Sb3_CC_u;	Sb3_CC_x = (*pPar)->x0 * Sb3_CC_u; ¹⁾

¹⁾ False code that contains double dereferencing.

Recommended practice

We strongly recommend to let TargetLink decide whether dereferencing is required. This is shown in the following table:

Expression in Custom Code File Section	Pointer	TargetLink ≤ 4.4	TargetLink 5.0
y = u + p;	Sa8_CC_param	Sa8_CC_out = Sa8_CC_in + Sa8_CC_param; ¹⁾	Sa8_CC_out = Sa8_CC_in + (*Sa8_CC_param); ²⁾
y = u + p;	-	Sa8_CC_out = Sa8_CC_in + Sa8_CC_param;	Sa8_CC_out = Sa8_CC_in + Sa8_CC_param;
x = p.x0 * u;	pPar	Sb3_CC_x = pPar.x0 * Sb3_CC_u; ¹⁾	Sb3_CC_x = (*pPar).x0 * Sb3_CC_u; ²⁾

¹⁾ False code with missing dereferencing.

²⁾ Correct code with dereferencing.

API Functions and Hook Scripts

Changes in API Functions

tlOperationMode

With TargetLink 5.0, the `tlOperationMode` API uses the following values for the `OperationMode` parameter:

- `ModelingOnly`
- `FullFeatured`

For the easy adjustment of scripts, TargetLink 5.0 introduces the new command `IsFullFeatured`. It is recommended to adjust the scripts as shown in the following example:

TargetLink < 5.0	TargetLink 5.0
<code>if (strcmp(tlOperationMode('get'), 'full-featured')), ...</code>	<code>if tlOperationMode('IsFullFeatured') ...</code>

Related documentation

- [tlOperationMode \(TargetLink API Reference\)](#)

Messages

Message changes

Changed message for keyword use

With TargetLink 5.0, a different error message is displayed if you use a C90 keyword as identifier:

TargetLink < 5.0	TargetLink 5.0
E17056	E17553

Related documentation

- [Basics on Generating Code Compliant with C99/C11/C++ \(TargetLink Customization and Optimization Guide\)](#)

VariableClass properties

In TargetLink < 5.0, invalid properties of DD VariableClass objects lead to a warning and the invalid property is ignored during code generation.

With TargetLink 5.0, invalid properties of DD VariableClass objects lead to an error.

TargetLink < 5.0	TargetLink 5.0
W17293	E17555

For components of explicit structs, the properties Const, Volatile, and TypePrefix are determined by the specification of the DD TypedefComponent objects of the struct Typedef, not the respective properties of the DD VariableClass object.

Related documentation

- [Basics on Arranging Data in Structured Variables \(TargetLink Customization and Optimization Guide\)](#)

Other Migration Aspects

Various Migration Aspects

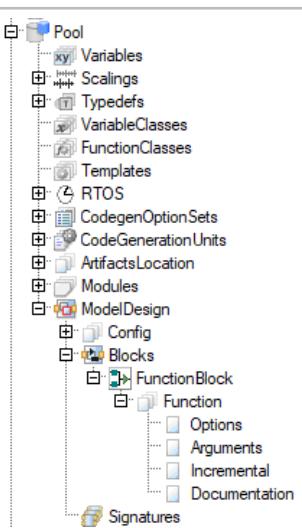
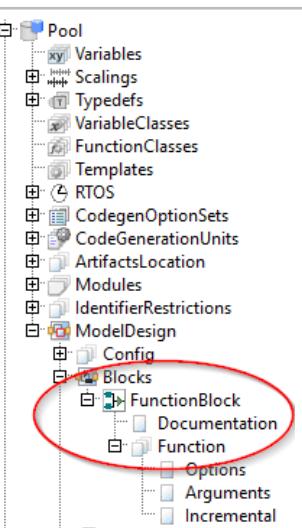
DD Documentation objects

DD Documentation objects changed:

In TargetLink < 5.0, the DD Documentation object is a child object of the DD Function object, that is a child object of the DD Block object. DD Documentation objects were available for DD Block objects whose BlockType property is set to `TL_Function`.

TargetLink 5.0 introduces new DD Documentation objects that are of object kind `BlockDocumentation`. The new objects are child objects of DD Block objects.

This leads to differences in the API functions. Refer to [Obsolete API Functions](#) on page 404.

TargetLink < 5.0	TargetLink 5.0
	

The new DD Documentation objects of object kind `BlockDocumentation` are available for all DD Block objects, regardless of the value of the `BlockType` property. They will be evaluated for DD Block objects whose `BlockType` property is set to `TL_Function` or `DataStoreMemory`.

The new objects have the new `CodeCommentPlacement` property.

Related documentation

- [Basics on the Data Model as Shown in the DD Object Tree](#) ( [TargetLink Data Dictionary Basic Concepts Guide](#))
- [Obsolete API Functions](#) on page 404

Stateflow paths	<p>With TargetLink 5.0, the syntax for paths of Stateflow elements changes for the following API commands:</p> <ul style="list-style-type: none"> ▪ <code>dsdd_get_block_path</code> ▪ <code>dsdd('GetPathToSLObject', ...)</code> <p>Elements are now separated by slashes ('/') as shown in the following example:</p> <pre>TL < 5.0 'sf_demo/sf_control/Subsystem/sf_control/Chart.do_blink.BLINKER_ON' TL 5.0 'sf_demo/sf_control/Subsystem/sf_control/Chart/do_blink/BLINKER_ON'</pre> <p>Related documentation</p> <ul style="list-style-type: none"> ▪ dsdd_get_block_path ( TargetLink API Reference) ▪ Documentation on <code>dsdd('GetPathToSLObject', ...)</code> available only in dSPACE Help.
Inhomogeneous scalings and constrained ranges	<p>Variables that describe the matrices of the Discrete State-Space block can have inhomogeneous scalings and inhomogeneous constrained ranges. With TargetLink 5.0, this is only possible if all of the following conditions hold:</p> <ul style="list-style-type: none"> ▪ The variables do not reference DD Variable objects. ▪ The variables do not reference DD ReplaceableDataItem objects. ▪ The block property <code>Keep matrix structure</code> is not set. <p>In this case, TargetLink generates a structured variable for the Discrete State-Space block whose scalar components contain the values of the matrices.</p> <p>Related documentation</p> <ul style="list-style-type: none"> ▪ Code Generation That Keeps Coefficient Vectors/Matrices in Struct Variables ( TargetLink Preparation and Simulation Guide)
ArbitraryLSB and autoscaling	<p>Controlling the autoscaling mode for each block is now realized with the two new autoscaling modes <code>Calculate power-of-two scaling</code> and <code>Calculate arbitrary scaling</code>. These modes can be set via the <code>Logging & Autoscaling</code> page of the respective block or via the <code>autoscaling.mode</code> property of the API.</p> <p>The TargetLink block property <code>variable.arb</code> is no longer available. Instead, TargetLink 5.0 lets you set the LSB to an arbitrary value without adjusting the <code>ArbitraryLSB</code> property beforehand. In the block dialog, the <code>2^/arb</code> button still conveniently lets you enter power-of-two scaling factors and converts scaling factors between decimal and power-of-two scaling scaling.</p> <p>Models are migrated automatically.</p> <p>If you want to check the scaling of a block via scripts, you are recommended to test if the value of the <code>.lsb</code> property can be represented as a power of two as shown in the following example:</p> <pre>log2Value = log2(lsb); isPowerOfTwo = round(log2Value) == log2Value;</pre>

Related documentation

- [Basics on Autoscaling of Output Variables](#) ([TargetLink Preparation and Simulation Guide](#))

Fixed-Point Library

With TargetLink 5.0, the Fixed-Point Library is no longer delivered as a pre-compiled library. Instead, it will be compiled automatically during the build process in the following cases:

- The first time a build process for a model is started if the generated code depends on the Fixed-Point Library.
- Each time the source files for the library have changed.

In these cases, the compilation of the Fixed-Point Library leads to increased build times.

Additionally, you can manually start the compilation via the new API function `tlRebuildFixedPointLibrary` ([TargetLink API Reference](#)).

Related documentation

- [Basics on Specifying the Location of the Sources and Binaries of the TargetLink Fixed-Point Library](#) ([TargetLink Customization and Optimization Guide](#))
- [tlRebuildFixedPointLibrary](#) ([TargetLink API Reference](#))

Migrating from TargetLink 4.3 to 4.4

Where to go from here

Information in this section

Code Generator Options.....	314
API Functions and Hook Scripts.....	316
Messages.....	317
Other Migration Aspects.....	317

Code Generator Options

Migration Aspects Regarding Code Generator Options

Removed Code Generator option

The following Code Generator options were removed from TargetLink:

Removed Option	Additional Information
CreateRestartFunctions	As a replacement, you can use the new InitializeVariablesViaRestartFunction option.

Changed Code Generator options

Code Generator Option	Old Default	New Default
None	-	-

Recommended compatibility settings

CreateRestartFunctions option	InitializeVariablesViaRestartFunction option	Additional information
off	Never	-
on	AsSpecified	This is the default setting.

Basics on changed defaults

The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code

Generator options in DD CodegenOptionSet objects (DD-based option storage). You can use DD CodegenOptionSet objects as a central source for overwriting and replacing the model-based option settings that were used since TargetLink 4.1.

If a model-based option value equals the old default value, it is automatically changed to the new default value during the upgrade. If a DD-based option value equals the old default value, it is not changed to the new default value during the upgrade but keeps the old value.

Option value = old default If Code Generator options were set to default values in the earlier TargetLink version, and the new TargetLink version uses modified default values, note the following points:

- Model-based option:

If you want to keep the old default values, you must reset them manually.

- DD-based option:

If you want to use the new default values, you must adjust them manually.

The following table is an example describing the impact of a TargetLink upgrade (TargetLink_{Old} to TargetLink_{New}) on three option values: 9, 11, and 13. The table illustrates two basic migration scenarios:

- Scenario #1: New default = old default

The default value of a Code Generator option has not changed in the new TargetLink version, i.e., the default value remains 9.

None of the option values is changed.

- Scenario #2: New default ≠ old default

The default value of a Code Generator option changed with the new TargetLink version, i.e., the default value changed to 11.

Option Storage	Option Value (TargetLink _{Old})	Option Value (≤ TargetLink _{New})	
		Default = 9 (Scenario #1)	Default = 11 (Scenario #2)
Model-based	9 ¹⁾	9 ¹⁾	11 ²⁾
	11	11	11 ¹⁾
	13	13	13
DD-based	9	9	9 ³⁾
	11	11	11
	13	13	13

¹⁾ The option value is not stored with the model because it equals the default.

²⁾ Manual reset might be necessary.

³⁾ Manual adjustment might be necessary.

Option value = new default If the Code Generator options were not set to default values in the former TargetLink version (A) but are in the new TargetLink version (B), TargetLink assumes that you intentionally specified the default value in the new TargetLink version. The same applies if the default changes again in the next TargetLink version (C).

Note

Upgrading TargetLink_A ⇒ TargetLink_B ⇒ TargetLink_C and upgrading TargetLink_A ⇒ TargetLink_C can cause different option values. Refer to the following table.

If the default values for TargetLink versions A, B, and C are 9, 11, and 13, and an option was set to 11 in version A, an upgrade to version C changes the option value as follows:

Upgrade Strategy	Option Value TargetLink_A Default = 9	Option Value TargetLink_B Default = 11	Option Value TargetLink_C Default = 13
A ⇒ B ⇒ C	11 (≠ default)	11 (= default) ¹⁾	13 (= default) ¹⁾
A ⇒ C	11 (≠ default)	—	11 (≠ default)

¹⁾ The option value is not stored with the model because it equals the default.

New Code Generator options For more information on new Code Generator options, refer to [New Code Generator Options](#) on page 68.

API Functions and Hook Scripts

Changes in API Functions

`tl_get_config_path`

Migration issue All paths must be defined as absolute paths. Do not use .. in conjunction with MATLAB's fullfile and addpath API commands.

Related documentation

- [tl_get_config_path](#) ([TargetLink API Reference](#))
- [How to Define TargetLink's Search Path for M-API-Related Customization Files](#) ([TargetLink Customization and Optimization Guide](#))

With TargetLink 4.4, new API functions were introduced. Refer to [New API Functions](#) on page 69.

Messages

Message changes

Changed message type

The following messages changed their type to **Advice** in TargetLink 4.4 because they only inform you about your responsibility for the initialization of external variables:

Old Message Number	New Message Number
W17055	A17055

Other Migration Aspects

Various Migration Aspects

Rate Transition block default settings

With TargetLink 4.4, the default settings of the Rate Transition block have changed. The checkboxes for Ensure data integrity during data transfer and for Ensure deterministic data transfer (maximum delay) are now cleared by default if you use the Rate Transition block via the TargetLink block library.

TargetLink does not support these settings. Before starting code generation, TargetLink checks if the checkboxes are selected. If so, TargetLink displays message W03839.

Migration issue None

Related documentation [Code-Relevant Simulink Blocks \(TargetLink Model Element Reference\)](#)

Non-TargetLink-compliant libraries and referenced models

You must not use libraries or referenced models that are not TargetLink-compliant. If you do use them, a message occurs in the System Checker.

Related documentation [E03887](#)

[Basics on Migrating Between TargetLink Versions](#) on page 286

Migrating from TargetLink 4.2 to 4.3

Where to go from here

Information in this section

API Functions and Hook Scripts.....	318
Code Generator Options.....	319
MATLAB-Related Changes.....	321
Other.....	322

API Functions and Hook Scripts

Changes in TargetLink and TargetLink Data Dictionary API Functions

tlSimInterface

The `tlSimInterface` API function now provides two commands that let you use TargetLink's online parameter modification with AUTOSAR calibration parameters:

Command	Purpose
<code>DisableRteStartBySimulationSFcn</code>	Disables a call of the AUTOSAR <code>Rte_Start()</code> function when the simulation of the simulation application (SIL/PIL) starts. TargetLink automatically enables a call of the <code>Rte_Start()</code> function after simulation (SIL/PIL) finishes. The <code>Rte_Start()</code> function is called only if AUTOSAR code is simulated.
<code>EnableRteStartBySimulationSFcn</code>	Enables a call of the AUTOSAR <code>Rte_Start()</code> function when the simulation of the simulation application (SIL/PIL) starts. This applies only if the call of the AUTOSAR <code>Rte_Start()</code> function was previously disabled by the <code>DisableRteStartBySimulationSFcn</code> command.

Related documentation

- [Basics on Modifying Parameter Values for Simulation](#) ([TargetLink Preparation and Simulation Guide](#))
- [How to Provide Manual Parameter Updates via the MATLAB API](#) ([TargetLink Preparation and Simulation Guide](#))
- [tlSimInterface](#) ([TargetLink API Reference](#))

tlUpgrade

`tlUpgrade` now lets you specify the TargetLink version to that you want to upgrade.

Related documentation

- [tlUpgrade](#) ([TargetLink API Reference](#))

Adapting the new workflow for distributed development

Migration issue With TargetLink 4.3, the old workflow of distributed development for referenced models is no longer available. The related API functions `tl_distribute_refmodel_files` and `tl_integrate_refmodel_files` were removed from TargetLink.

Solution To distribute and integrate the files, specify their locations in the Data Dictionary.

To create separate development frame models, use `tlExtractSubsystem`.

Related documentation

- [Specifying the Location of Artifacts Generated or Used by TargetLink](#) ([TargetLink Customization and Optimization Guide](#))
- [tlExtractSubsystem](#) ([TargetLink API Reference](#))

Related topics**References**

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

Code Generator Options

Migration Aspects Regarding Code Generator Options

Removed Code Generator option

The following Code Generator options were removed from TargetLink:

Removed Option	Additional Information
None	-

Changed Code Generator options

Code Generator Option	Old Default	New Default
None	-	-

Recommended compatibility settings

None

Basics on changed defaults

The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code Generator options in DD CodegenOptionSet objects (DD-based option storage). You can use DD CodegenOptionSet objects as a central source for overwriting and replacing the model-based option settings that had been used since TargetLink 4.1.

If a model-based option value equals the old default value, it is automatically changed to the new default value during the upgrade. If a DD-based option value equals the old default value, it is not changed to the new default value during the upgrade but keeps the old value.

Option value = old default If Code Generator options were set to default values in the former TargetLink version, and the new TargetLink version uses modified default values, note the following points:

- Model-based option:

If you want to keep the old default values, you must reset them manually.

- DD-based option:

If you want to use the new default values, you must adjust them manually.

The following table is an example describing the impact of a TargetLink upgrade (TargetLink_{Old} to TargetLink_{New}) on three arbitrary option values: 9, 11, and 13. The table illustrates two basic migration scenarios:

- Scenario #1: New default = old default

The default value of a Code Generator option has not changed in the new TargetLink version, i.e., the default value remains 9.

None of the option values is changed.

- Scenario #2: New default ≠ old default

The default value of a Code Generator option changed with the new TargetLink version, i.e., the default value changed to 11.

Option Storage	Option Value (TargetLink _{Old})	Option Value (\leq TargetLink _{New})	
		Default = 9 (Scenario #1)	Default = 11 (Scenario #2)
Model-based	9 ¹⁾	9 ¹⁾	11 ²⁾
	11	11	11 ¹⁾
	13	13	13
DD-based	9	9	9 ³⁾
	11	11	11
	13	13	13

¹⁾ The option value is not stored with the model because it equals the default.

²⁾ Manual reset might be necessary.

³⁾ Manual adjustment might be necessary.

Option value = new default If the Code Generator options were not set to default values in the former TargetLink version (A) but are in the new TargetLink version (B), TargetLink assumes that you intentionally specified the default value

in the new TargetLink version. The same applies if the default changes again in the next TargetLink version (C).

Note

Upgrading TargetLink_A ⇒ TargetLink_B ⇒ TargetLink_C and upgrading TargetLink_A ⇒ TargetLink_C can cause different option values. Refer to the following table.

If the default values for TargetLink versions A, B, and C read 9, 11, and 13, and an option was set to 11 in version A, an upgrade to version C changes the option value as follows:

Upgrade Strategy	Option Value TargetLink _A Default = 9	Option Value TargetLink _B Default = 11	Option Value TargetLink _C Default = 13
A ⇒ B ⇒ C	11 (≠ default)	11 (= default) ¹⁾	13 (= default) ¹⁾
A ⇒ C	11 (≠ default)	—	11 (≠ default)

¹⁾ The option value is not stored with the model because it equals the default.

New Code Generator options

For more details on new Code Generator options, refer to [New Code Generator Options](#) on page 82.

MATLAB-Related Changes

Modified features in MATLAB R2016b

Changed startup behavior

MATLAB has changed the execution order of startup scripts.

- Up to MATLAB R2016a, the `startup.m` script is executed first, before the dSPACE-specific initialization scripts `dsstartup.m` and `dspoststartup.m`.
- As of MATLAB R2016b, the `startup.m` script is executed last, after the dSPACE-specific initialization scripts.

This modification might result in a changed startup behavior.

Other

Where to go from here	Information in this section
	Property Manager - Migration Aspects..... 322 Various Migration Aspects..... 324

Property Manager - Migration Aspects

Major changes See the following table for the main differences between the old version and the new version of the Property Manager.

Previous Property Manager	New Property Manager
Block Filter to specify the block types to be displayed.	Different new filter options of the Property View. For more information, refer to How to Filter and Sort Property Values in the Property View .
Separate Property Editor to edit property values.	Editing property values directly in the respective cell in the Property View. For more information, refer to Editing Property Values in the Property View (TargetLink Preparation and Simulation Guide)
Property Selector to define which properties of the block types selected in the Block Filter are to be displayed. Additionally, you can select and clear properties that are shown in the Block Explorer.	Column Chooser to select the model elements to be displayed in the Property View. For more information, refer to Column Chooser Dialog (TargetLink Tool and Utility Reference).
Displaying multiple models, each model is displayed in its own window.	Display multiple models and/or libraries simultaneously as individual nodes in the Model Navigator. For more information, refer to How to Load Models, Libraries, or Referenced Subsystems to the Model Navigator (TargetLink Preparation and Simulation Guide)
Status bar to display comments and information about the TargetLink/Simulink blocks, and how many TargetLink block types are selected in the Block Filter.	Messages control bar to display system messages in a chronological order. It provides a history of all the info, advice, error, and warning messages. This helps you check the system state. It lets you search for messages and filters the messages to be displayed. For more information, refer to Message Viewer (TargetLink Tool and Utility Reference).

Previous Property Manager	New Property Manager
TargetLink API names of properties as column headers.	<p>Describing terms as column headers, because identical properties of different model elements are merged in one column.</p> <p>Hover over a cell in the Property View to display the property's TargetLink API name as a tooltip. The tooltips of header cells also provide a description of the property.</p> <p>For more information, refer to Display of Properties in the Property View (TargetLink Preparation and Simulation Guide).</p> <p>In the Column Chooser you can search for the property's TargetLink API name and a description is provided as a tooltip.</p> <p>For more information, refer to Column Chooser Dialog (TargetLink Tool and Utility Reference).</p>
TargetLink internal content of TargetLink subsystems is hidden.	<p>TargetLink internal content of TargetLink subsystems is displayed, for example, TargetLink simulation frames including additional S-functions.</p> <p>You cannot edit TargetLink internal data in the Property Manager.</p>
Invalid inputs are rejected immediately.	Advanced validation of data.

Related Documentation

- [New Property Manager](#) on page 83
- [Modifying Multiple Properties at Once via the Property Manager](#) ([TargetLink Preparation and Simulation Guide](#))
- [Property Manager](#) ([TargetLink Tool and Utility Reference](#))

Changes concerning the CSV Export

There are some structural changes in the CSV export of the new Property Manager:

- Only values containing special characters or line breaks are written in quotation marks.
- The first three columns always are:
 - Status
 - Name (ModelElement)
 - VariableName (ModelElement)
- Columns are renamed and columns with identical properties of different model elements are merged. For example, gain.value and input.value are now both called initialValue.
- Line breaks in model element names are included in the CSV export.

DIALOGS HIDDEN IN THE BACKGROUND

Migration issue If you open TargetLink dialogs in the Property Manager, the Simulink model window might not display in the foreground but in the background. Instead, the Windows taskbar displays a flashing message referring to the Simulink model window.

Hint Keep in mind that additional windows might open in the background unexpectedly. Arrange the window panes in a way that they are displayed side by side without covering each other.

Related documentation

- [Open Model](#) (TargetLink Tool and Utility Reference)
- [How to Trace Model Elements Back to the Simulink Model](#) (TargetLink Preparation and Simulation Guide)

Reduced access rate of TargetLink API functions

Migration issue If a model is loaded in the Property Manager, the access rate of TargetLink API functions (for example, `tl_get` or `tl_set`) could be reduced. This can also affect the performance of tool chains.

Solution Close the Property Manager via the `tlPropman('Exit')` API command before the execution of tool chains.

For more information, refer to [tlPropman](#).

Related topics**References**

[tlPropman](#) (TargetLink API Reference)

Various Migration Aspects

Recommended signal delay modelings with the newly supported Delay block

Migration issue The Delay block is supported as of TargetLink 4.3. In new models, use the Delay block with appropriate settings rather than the following modeling options:

- Unit Delay Reset Enabled block
- Chain of Unit Delay blocks
- Custom Code block (type II) which has a Simulink Delay block under its mask

For the Unit Delay Reset Enabled block, take into account the following differences between these blocks:

Delay block	Unit Delay Reset Enabled block
<p>One output</p> <p>Reset behavior:</p> <p>The internal condition is <i>not</i> reset if all of the following conditions apply:</p> <ul style="list-style-type: none"> ▪ The enable signal is 0. ▪ The External reset is not set to None. ▪ The reset signal triggers a reset. <p>If a state reset is successfully triggered, the initial value (IV) will be written to the output and the data signal (U) will be written to the state.</p>	<p>Two outputs (one for the delayed signal and one for the state signal)</p> <p>Reset behavior:</p> <p>The internal condition is reset if the following two conditions apply:</p> <ul style="list-style-type: none"> ▪ The enable signal is 0. ▪ The reset signal should trigger a reset. <p>If a state reset is successfully triggered, the initial value (IV) will be written to the output and the state.</p>

Related documentation

- [Delay Block](#) ([TargetLink Model Element Reference](#))
- [CUSTOM_ENHANCEMENT](#) ([TargetLink Demo Models](#))

Variable descriptions in A2L files

Migration issue A2L files generated with TargetLink 4.3 contain descriptions of only calibratable/displayable variables that are used in the production code of the code generation unit (CGU) for which the A2L file was generated.

Solution To include descriptions of variables that are used in different CGUs in a single A2L file, generate code for all the CGUs and generate one A2L file from all of the DD Subsystem objects.

To include descriptions of externally defined variables in the A2L file, select the External variables checkbox.

Related documentation

- [Basics on Specifying the Variables to Export to A2L Files](#) ([TargetLink Interoperation and Exchange Guide](#))
- [Export as A2L File](#) ([TargetLink Data Dictionary Manager Reference](#))

Encoding of A2L files

Migration issue With Version 1.6.1 of the ASAM MCD-2 MC standard, A2L files now have to be encoded as **UTF-8 with BOM**.

To ensure downward compatibility, the A2L files generated by TargetLink are not encoded as in previous TargetLink versions according to the OS Locale (default) or character encoding setting specified in the DD.. This might cause problems with [measurement and calibration systems](#) that require the A2L files to be encoded as specified by Version 1.6.1 of the ASAM MCD-2 MC standard.

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

Related documentation

- [Overview of the Supported Character Sets](#) ([TargetLink Interoperation and Exchange Guide](#))
- [How to Specify the Used Character Set](#) ([TargetLink Interoperation and Exchange Guide](#))

Installing multiple TargetLink instances

By default, you can no longer install multiple instances of the same TargetLink version. This also applies to versions with different patch levels. You can still install different TargetLink versions on the same host PC.

If you need to install several instances of TargetLink 4.3, contact dSPACE Support (www.dspace.com/go/supportrequest).

Changes in code output configuration	<p>Migration issue With TargetLink 4.3, the code output configuration file (<code>cconfig.xml</code>) was changed.</p>				
	<p>Reason To clarify the comments of code statements.</p>				
	<p>Solution If you changed the code output configuration in a prior version, adapt your changes to the new version supplied with TargetLink 4.3.</p>				
	<p>Related documentation Customizing TargetLink's Code Formatting (TargetLink Customization and Optimization Guide)</p>				
New build directory and location of RTE frame files	<p>With TargetLink 4.3, the default for the build directory changed from <code>.\TLSim\<Application>\<Board>_<Compiler></code> to <code>.\TLBuild\<Application>\<Board>_<Compiler></code>.</p>				
	<p>The build folder is used to gather the files of the RTE frame, which are not CGU-specific. These files are used only for simulating the AUTOSAR compliant code.</p>				
Changed behavior with scope reduction and initialization	<p>Migration issue TargetLink's initialization behavior for variables changed. This applies to restart initialization and initialization at definition. If the variables are reduced along a user-specified scope reduction chain (SRC) that ends in a module-local variable class, TargetLink behaves as shown in the following table:</p>				
	<table border="1" data-bbox="555 992 1411 1140"> <thead> <tr> <th data-bbox="555 992 992 1034">TargetLink ≤ 4.2</th><th data-bbox="992 992 1411 1034">TargetLink 4.3</th></tr> </thead> <tbody> <tr> <td data-bbox="555 1034 992 1140">Removal of variable/restart initialization if a reduction to automatic storage duration is possible.</td><td data-bbox="992 1034 1411 1140">Variable/restart initialization even if a reduction to automatic storage duration is possible.</td></tr> </tbody> </table>	TargetLink ≤ 4.2	TargetLink 4.3	Removal of variable/restart initialization if a reduction to automatic storage duration is possible.	Variable/restart initialization even if a reduction to automatic storage duration is possible.
TargetLink ≤ 4.2	TargetLink 4.3				
Removal of variable/restart initialization if a reduction to automatic storage duration is possible.	Variable/restart initialization even if a reduction to automatic storage duration is possible.				
	<p>Reason TargetLink now treats user-defined SRCs that do not end with a variable class that has automatic storage duration as intended by the user: For example, to work with variable accesses by address, that are not visible to TargetLink.</p>				
	<p>Solution Resolve potential problems as follows:</p>				
	<table border="1" data-bbox="555 1267 1411 1541"> <thead> <tr> <th data-bbox="555 1267 944 1309">No Reason for Shortened SRC</th><th data-bbox="944 1267 1411 1309">Reason for Shortened SRC but Initialization not Necessary</th></tr> </thead> <tbody> <tr> <td data-bbox="555 1309 944 1541">Change the module-local variable class as follows:<ul style="list-style-type: none">▪ Optimization = SCOPE_REDUCIBLE▪ ScopeReducedClass = <>¹⁾</td><td data-bbox="944 1309 1411 1541">Change the variable classes forming the SRC as follows:<ul style="list-style-type: none">▪ InitAtDefinition = off▪ RestartFunctionName = ""</td></tr> </tbody> </table>	No Reason for Shortened SRC	Reason for Shortened SRC but Initialization not Necessary	Change the module-local variable class as follows: <ul style="list-style-type: none">▪ Optimization = SCOPE_REDUCIBLE▪ ScopeReducedClass = <>¹⁾	Change the variable classes forming the SRC as follows: <ul style="list-style-type: none">▪ InitAtDefinition = off▪ RestartFunctionName = ""
No Reason for Shortened SRC	Reason for Shortened SRC but Initialization not Necessary				
Change the module-local variable class as follows: <ul style="list-style-type: none">▪ Optimization = SCOPE_REDUCIBLE▪ ScopeReducedClass = <>¹⁾	Change the variable classes forming the SRC as follows: <ul style="list-style-type: none">▪ InitAtDefinition = off▪ RestartFunctionName = ""				
	<p>¹⁾ Or references a variable class with static storage duration.</p>				
Interpolation in look-up table blocks	<p>The Look-Up Table and Look-Up Table (2D) block support the Use Input Nearest look-up method again.</p>				
Changed value of FilePath property	<p>Migration issue With TargetLink 4.3 the value of the FilePath property of DD Module objects contained in DD Subsystem objects changed. The value</p>				

now contains the placeholder `$(CGURootOutputFolder)` that is not evaluated by the `dsdd('GetFilePath',...)` DD API function.

Reason To make artifact locations user-specifiable.

Solution Use the `dsdd('GetCompiledFilePath',...)` DD API function instead.

1x1 signals

Migration issue During code generation, TargetLink displays the E03004 message when a signal whose dimension is `1x1` is fed into one of the following blocks:

- FIR Filter
- Discrete Filter
- Discrete Transfer Fcn
- Discrete-Time Integrator

Reason Bug fix

Solution Place Simulink Reshape blocks on the signal lines connected to the TargetLink block and set their Output dimensionality to `1-D array`.

Tip

Simulink's Selector block outputs a `1x1` matrix signal when selecting a single element of a matrix or vector.

TL_ExternBlock mask type is no longer supported

Migration issue With TargetLink 4.3, the `TL_ExternBlock` mask type is no longer supported.

Reason The `TL_BlackBox` mask type was improved.

Solution Use the `TL_BlackBox` mask type instead. Adapt the mask type changes to the new version supplied with TargetLink 4.3.

Related documentation [Overview of Methods for Preparing Unsupported Simulink Blocks for TargetLink Code Generation](#) ([TargetLink Preparation and Simulation Guide](#))

Migrating from TargetLink 4.1 to 4.2

Where to go from here	Information in this section
	API Functions and Hook Scripts..... 328
	AUTOSAR-Related Migration Aspects..... 330
	Code Generator Options..... 333
	MATLAB-Related Changes..... 335
	Other..... 336

API Functions and Hook Scripts

Changes in TargetLink and TargetLink Data Dictionary API Functions

tl_set	<p>Migration issue You can now use <code>tl_set</code> to change the block properties of library items from inside mask callbacks and similar callbacks in the same API call, by setting the <code>AllowLibraryBlockModification</code> property to <code>on</code>:</p> <pre>tl_set(<block>,..., 'AllowLibraryBlockModification', 'on',...)</pre> <p>Related documentation tl_set (TargetLink API Reference)</p>
tl_generate_swc_model	<p>Migration issue With the introduction of asynchronous client-server communication, the <code>AddOperationCallTriggerPort</code> property of the <code>tl_generate_swc_model</code> API function was renamed to <code>AddOperationSubsystemTriggerPort</code>.</p> <p>Solution Adapt your user scripts. TargetLink automatically changes the name of the <code>/Pool/Autosar/Config/FrameModelGeneration/AddOperationCallTriggerPort</code> property to <code>AddOperationSubsystemTriggerPort</code> when upgrading the Data Dictionary file/contents to the latest revision.</p>

Related documentation

- [Asynchronous Client-Server Communication](#) on page 97
- [Modeling Client-Server Communication](#) ([TargetLink Classic AUTOSAR Modeling Guide](#))
- [tl_generate_sw_c_model](#) ([TargetLink API Reference](#))

tldoc

The generation of RTF documentation is no longer supported. The `CreateRTFFile` property of the `tldoc('Convert')` command is obsolete. If you used this property in the past, you must edit your scripts and remove the property from the `tldoc('Convert')` commands. Use HTML or PDF instead.

To postprocess the documentation manually, use an HTML editor or a PDF editor of your choice. However, you should consider to customize the documentation generation which might render manual edit steps unnecessary. New features of the TargetLink documentation generation process might help you. Refer to [General Enhancements and Changes](#) on page 104.

Related documentation

- [Basics on Customizing the Generated Documentation](#) ([TargetLink Interoperation and Exchange Guide](#))
- [tldoc\('Convert', propertyName, PropertyValue, ...\)](#) ([TargetLink API Reference](#))
- [Discontinued TargetLink Features](#) on page 411

Data model filter rule files

Migration issue Existing data model filter rule files can contain invalid elements because the TargetLink Data Dictionary's data model changed. The following files shipped with TargetLink 4.1 are affected:

- `DD_Filter_Admin.xml`
- `DD_Filter_AR_User.xml`
- `DD_Filter_NonAR_NonRTOS_User.xml`

Solution You can check filter rule files via the API in the MATLAB Command Window:

Checking a Single File	Checking Filter Rule Sets ¹⁾
<pre>dsdd_free; dsdd('ReadFilterRuleSet', 'file', '<myFile>.xml'); ds_error_register(dsdd('GetMessageList')); ds_msgdlg('update');</pre>	<pre>dsdd_free; dsdd('ReloadFilterRuleSets'); ds_error_register(dsdd('GetMessageList')); ds_msgdlg('update');</pre>

¹⁾ All the files contained in the directory defined in Data Dictionary - Filter Rules in the Preferences Editor.

TargetLink informs you about errors in TargetLink's Message Browser. Each error contains the following information, so that you can fix it in an XML-capable editor of your choice:

- Filename
- Row number
- Column number

AUTOSAR-Related Migration Aspects

Where to go from here	Information in this section
	Migrating Interrunnable Variable Specifications..... 330
	Migrating Transformer Error Code Specifications..... 332

Migrating Interrunnable Variable Specifications

Migration issue	Interrunnable variables (IRVs) are now modeled via dedicated DD InterRunnableVariable objects and not via DD Variable objects that reference one of the predefined DD VariableClass objects called EXPLICIT_IRV, DISP_EXPLICIT_IRV, IMPLICIT_IRV and DISP_IMPLICIT_IRV (access function mechanism).
Preparing your old Data Dictionary	<p>Before you can migrate your IRV specifications to TargetLink 4.2, you have to prepare your old Data Dictionary:</p> <ul style="list-style-type: none">▪ Every DD VariableGroup object that contains DD Variable objects that represent IRVs must be referenced by <i>exactly one</i> DD SoftwareComponent object via its RelatedVariables/InterRunnableVariablesRef property.▪ For each IRV that you want to be initialized, make sure that an AUTOSAR constant specification is well defined by the following properties of the DD Variable object:<ul style="list-style-type: none">▪ Value▪ InitConstantName▪ InitConstantFileName▪ InitConstantPackage▪ InitConstantUuid▪ Save your Data Dictionary.
Upgrading your Data Dictionary	<p>You can now upgrade your Data Dictionary. TargetLink performs the following steps for each DD Variable object contained in a DD VariableGroup object that is referenced via a DD SoftwareComponent object's InterrunnableVariablesRef property, contained in the RelatedVariables subtree:</p> <ul style="list-style-type: none">▪ Creates a DD InterRunnableVariable object and adds it to the software components' InterRunnableVariables subtree.

- Copies the IRV-specific property values and all the custom properties from the DD Variable object to the DD InterRunnableVariable object.
- References the old DD Variable object at the DD InterRunnableVariable object's UpgradeFromVariableRef custom property.
- If the old DD Variable object defines a constant specification to initialize the IRV, it searches the DD Variable object that matches the IRV's constant specification:
 - If it finds a suitable Variable object, TargetLink references it at the InterRunnableVariable object's InitValueRef property.
 - If it does not find a suitable Variable object, TargetLink creates a new object and references it at the InterRunnableVariable object's InitValueRef property.
- Sets the old DD Variable object's Class property as follows after all objects were processed:

Old Variable Class	New Variable Class
EXPLICIT_IRV	default
IMPLICIT_IRV	
DISP_EXPLICIT_IRV	DISP
DISP_IMPLICIT_IRV	

- Removes the old DD VariableClass objects from the Data Dictionary.

Note

- Do not delete or unset the UpgradeFromVariableRef property or the DD Variable object it references.
This information is required during model upgrade.
- If you want to create a fresh DD workspace that contains only the data imported from an AUTOSAR file or SWC container, make sure to merge each DD InterRunnableVariable object's UpgradeFromVariableRef property and the DD Variable object it references into your new workspace.

Upgrading your model

After you upgraded your Data Dictionary, you can upgrade your model. The model upgrade is automatically performed the first time a model is opened in a new TargetLink Version. If the attached Data Dictionary is not upgraded, TargetLink will prompt you to upgrade the Data Dictionary.

During model upgrade, TargetLink searches on the AUTOSAR page of port blocks and the Output page of Data Store Memory blocks for references of DD Variable objects that are referenced via a DD InterRunnableVariable object's UpgradeFromVariableRef property and makes the following settings:

Port Blocks	Data Store Memory Blocks
Replaces the references to the old DD Variable objects with references to the new DD InterRunnableVariable objects.	<p>Deactivates references to the old Variable objects from the block's Output page and makes the following settings on the block's AUTOSAR page:</p> <ul style="list-style-type: none"> ▪ Selects the Use AUTOSAR communication checkbox. ▪ Sets the Kind property to InterRunnable. ▪ References the new DD InterRunnableVariable object.

Purging old variables

After you made sure that the model upgrade succeeded, you can purge the DD from the old DD Variable objects and delete the DD InterRunnableVariable objects' UpgradeFromVariableRef property.

TargetLink provides the following API for this:

```
errorCode = dsdd(
    'DeleteObsoleteInterRunnableVariables',
    <DdSwcObjectIdentifier>);
```

Note

Executing this API function performs the following steps without warning. This cannot be undone.

- Deletes all DD Variable objects that are referenced via DD InterRunnableVariable objects' UpgradeFromVariableRef property.
- Deletes the DD VariableGroup objects that contained the DD Variable objects if they are empty.
- Deletes the UpgradeFromVariableRef property from the DD InterRunnableVariable objects.

Related documentation

- [Modeling Interrunnable Communication \(TargetLink Classic AUTOSAR Modeling Guide\)](#)
- [Non-Scalar Interrunnable Variables](#) on page 98
- [DeleteObsoleteInterRunnableVariables](#)

Migrating Transformer Error Code Specifications

Migration issue

AUTOSAR changed transformer error codes from revision 4.2.1 to 4.2.2.

These changes are not resolved during the automatic upgrade but require your attention.

Solution

The latest version of the dsdd_master_autosar4.dd [System] DD template contains the new transformer error codes.

You can open this template in a new workspace and merge the error codes into your DD project file using the DD comparison pane. They are located in the following DD VariableGroup:

`/Pool/Variables/AUTOSAR/TransformerError`

Code Generator Options

Migration Aspects Regarding Code Generator Options

Removed Code Generator options

The following Code Generator options were removed from TargetLink:

Removed Option	Additional Information
ConsiderFunctionClassesForBlockDiagramBasedSwitchOptimization	In TargetLink 4.1 this option was set to on by default. TargetLink now always behaves as if this option was set to on .
ConsiderStateflowAuxiliariesForVariableSharing	In TargetLink 4.1 this option was set to on by default. TargetLink now always behaves as if this option was set to on .
EnableVariableVectorWidths	In TargetLink 4.1 this option was set to on by default. TargetLink now always behaves as if this option was set to on . This has only an effect if you use ExchangeableWidth objects in the TargetLink DataDictionary and your model.
TreatAllStateflowFunctionsAsWeakAtomic	In TargetLink 4.1 this option was set to on by default. TargetLink now always behaves as if this option was set to on .

Changed Code Generator options

Code Generator Option	Old Default	New Default
EnableReportGeneration	off	on
ReportFailedFunctionReuseVariablePropagation	on	off
ReportMismatchingPortBlockSignalSpecifications	on	off
ReportVariablesExcludedFromPIM	on	off

Recommended compatibility settings Specify the following settings for new TargetLink 4.2 Code Generator options to ensure best possible downward compatibility:

- Set ForceBooleanOperandsInBooleanOperations to off.

Basics on changed defaults

The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code Generator options in DD CodegenOptionSet objects (DD-based option storage). You can use DD CodegenOptionSet-objects as a central source for overwriting and instead of model-based option settings since TL 4.1.

If a model-based option value equals the old default value, it is automatically changed to the new default value during the upgrade. If a DD-based option value equals the old default value, it is not changed to the new default value during the upgrade but keeps the old value.

Option value = old default If Code Generator options equal default values in the former TargetLink version and the new TargetLink version uses modified default values, note the following points:

- Model-based option:

If you want to keep the old default values, you must reset them manually.

- DD-based option:

If you want to use the new default values, you must adjust them manually.

The following table is an example describing the impact of a TargetLink upgrade (TL_{Old} to TL_{New}) on three arbitrary option values: 9, 11, and 13. The table illustrates two basic migration scenarios:

- Scenario #1: New default = old default

The default value of a Code Generator option has not changed in the new TargetLink version, i.e., the default value remains 9.

None of the option values is changed.

- Scenario #2: New default \neq old default

The default value of a Code Generator option changed with the new TargetLink version, i.e., the default value changed to 11.

Option Storage	Option Value (TL_{Old}) Default = 9	Option Value ($\leq TL_{New}$)	
		Default = 9 (Scenario #1)	Default = 11 (Scenario #2)
Model-based	9 ¹⁾	9 ¹⁾	11 ²⁾
	11	11	11 ¹⁾
	13	13	13
DD-based	9	9	9 ³⁾
	11	11	11
	13	13	13

¹⁾ The option value is not stored with the model because it equals the default.

²⁾ Manual reset might be necessary.

³⁾ Manual adjustment might be necessary.

Option value = new default If the Code Generator options did not equal default values in the former TargetLink version (A) but do in the new TargetLink version (B), TargetLink assumes that you intentionally specified the default value in the new TargetLink version. The same applies if the default changes again in the next TargetLink version (C).

Note

Upgrading $TL_A \Rightarrow TL_B \Rightarrow TL_C$ and upgrading $TL_A \Rightarrow TL_C$ can cause different option values (see the following table).

If the default values for TargetLink versions A, B, and C read 9, 11, and 13, and an option value equaled 11 in version A, an upgrade to version C would change the option value as follows:

Upgrade Strategy	Option Value TL_A Default = 9	Option Value TL_B Default = 11	Option Value TL_C Default = 13
$A \Rightarrow B \Rightarrow C$	11 (\neq default)	11 (= default) ¹⁾	13 (= default) ¹⁾
$A \Rightarrow C$	11 (\neq default)	—	11 (\neq default)

¹⁾ The option value is not stored with the model, because it equals the default.

New Code Generator options For more details on new Code Generator options, refer to [New Code Generator Options](#) on page 102.

Related topics References

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

MATLAB-Related Changes

Modified features in MATLAB R2016b

Changed startup behavior MATLAB has changed the execution order of startup scripts.

- Up to MATLAB R2016a, the `startup.m` script is executed first, before the dSPACE-specific initialization scripts `dsstartup.m` and `dspoststartup.m`.
- As of MATLAB R2016b, the `startup.m` script is executed last, after the dSPACE-specific initialization scripts.

This modification might result in a changed startup behavior.

Other

Various Migration Aspects

Requirement information	<p>Requirement information is no longer stored as a visible block property. Instead, TargetLink blocks now have the <code>HasRequirementInfos</code> flag. This lets you obtain a list of blocks that carry requirement information via the <code>tl_find</code> API function.</p> <p>To add or remove requirement information, use the <code>tlRequirementInfo</code> API function.</p>
Remapping TargetLink scaling properties during system clearance	<p>Migration issue When TargetLink specifies a scaled variable (<code>LSB != 2^0</code> or <code>Offset != 0.0</code>) and Simulink specifies an unscaled datatype (for example, one of the built-in datatypes <code>int8</code>, <code>int16</code>, ..., <code>single</code>, <code>double</code>, <code>boolean</code>), TargetLink 4.2 does not set Simulink's datatype to a scaled type, but displays message E03819.</p> <p>Solution If you want to know, which TargetLink scaling property could not be mapped to a Simulink scaling parameter, select the <code>Verbose</code> output checkbox in the <code>Clear System</code> from TargetLink dialog.</p> <p>In verbose mode, TargetLink displays a message for each property that could not be mapped.</p> <p>For example:</p> <ul style="list-style-type: none">▪ Simulink's settings already match▪ Simulink specifies property inheritance▪ Simulink specifies a bus or enum datatype <div style="background-color: #f0f0f0; padding: 10px; border-radius: 5px;"><p>Note</p><p>Big models can result in a huge number of messages.</p></div>

Related topics

References

[tl_find](#) ( TargetLink API Reference)
[tlRequirementInfo](#) ( TargetLink API Reference)

Migrating from TargetLink 4.0 to 4.1

Where to go from here

Information in this section

API Functions and Hook Scripts.....	337
AUTOSAR-Related Migration Aspects.....	338
Code Generator Options.....	339
Other.....	341

API Functions and Hook Scripts

Changes in TargetLink and TargetLink Data Dictionary API Functions

Custom look-up functions

For the `tlscript` API function, a new property is available which relates to the dimension of input signals. This lets you implement a last index state variable for the Local search table search method, for example.

- `InheritDimensionFromInput`

Related documentation

- Permissible Properties for Variables ([TargetLink API Reference](#))
- Basics on Using Custom Look-Up Functions ([TargetLink Preparation and Simulation Guide](#))

AUTOSAR-Related Migration Aspects

AUTOSAR-Related Migration Aspects

Name macros in name templates of runnable objects With TargetLink 4.1, the NameTemplate property of Runnable objects must not contain name macros other than \$D.

Removed options in import/export The Merge and EnablePackageSupport options are obsolete. The corresponding properties in /Pool/Autosar/Config/ImportExport were removed. TargetLink now always does the following:

- Merges Data Dictionary elements that belong to one software component but reside in different subsystems into one merged file (Merge = On).
- Imports and exports the package information provided in AUTOSAR files and the Data Dictionary (EnablePackageSupport = On).

Modify your scripts accordingly.

RTE error code macros in the Data Dictionary The variables representing RTE error code macros in the /Pool/Variables/AUTOSAR/Std_ReturnType variable group and the predefined variable class /Pool/VariableClasses/AUTOSAR/RTE/RTE_ERROR_CODE now have their ModuleRef property set to Rte_Frame. This is in accordance with the AUTOSAR standard, which requires these macros to be defined in Rte.h. The new value of each variables' ModuleRef property is set automatically during a Data Dictionary upgrade. This can result in an #include Rte.h in production code if one of these variables is referenced at a block in your model.

Code Generator Options

Migration Aspects Regarding Code Generator Options

Removed Code Generator option The MPC5xx-specific (TOM-specific) EnableLogicalOperationOptimisation Code Generator option was removed from TargetLink:

Removed Option	Replacement Option	Compatibility Setting
EnableLogicalOperationOptimisation	None	None

With TargetLink 4.1, this option was removed together with MPC5xx-support.

Changed Code Generator options The following Code Generator options changed with TargetLink 4.1:

- None

Recommended compatibility settings Make the following settings for new TargetLink 4.1 Code Generator options to ensure best possible downward compatibility:

- None

Basics on changed defaults The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code Generator options in DD CodegenOptions objects (DD-based option storage). You can use DD CodegenOptions-objects as a central source for overwriting model-based option settings.

If a model-based option value equals the old default value, it is automatically changed to the new default value during upgrade. If a DD-based option value equals the old default value, it is not changed to the new default value during upgrade but keeps the old value.

Option value = old default If Code Generator options equal default values in the former TargetLink version and the new TargetLink version uses modified default values, note the following points:

- Model-based option:
If you want to keep the old default values, you must reset them manually.
- DD-based option:
If you want to use the new default values, you must adjust them manually.

The following table is an example describing the impact of a TargetLink upgrade (TL_{Old} to TL_{New}) on three arbitrary option values: 9, 11, and 13. The table illustrates two basic migration scenarios:

- Scenario #1: New default = old default

The default value of a Code Generator option has not changed in the new TargetLink version, i.e., the default value remains 9.

None of the option values is changed.

- Scenario #2: New default \neq old default

The default value of a Code Generator option changed with the new TargetLink version, i.e., the default value changed to 11.

Option Storage	Option Value (TL_{Old}) Default = 9	Option Value ($\leq TL_{New}$)	
		Default = 9 (Scenario #1)	Default = 11 (Scenario #2)
Model-based	9 ¹⁾	9 ¹⁾	11 ²⁾
	11	11	11 ¹⁾
	13	13	13
DD-based	9	9	9 ³⁾
	11	11	11
	13	13	13

¹⁾ Option value is not stored with the model because it equals the default.

²⁾ Manual reset might be necessary.

³⁾ Manual adjustment might be necessary.

Option value = new default If Code Generator options did not equal default values in the former TargetLink version (A) but do in the new TargetLink version (B), TargetLink assumes that you intentionally specified the default value in the new TargetLink version. The same applies if the default changes again in the next TargetLink version (C).

Note

Upgrading $TL_A \Rightarrow TL_B \Rightarrow TL_C$ and upgrading $TL_A \Rightarrow TL_C$ can cause different option values (see the following table).

Suppose the default values for TargetLink versions A, B, and C read 9, 11, and 13. If an option value equaled 11 in version A, an upgrade to version C would change the option value as follows:

Upgrade Strategy	Option Value TL_A Default = 9	Option Value TL_B Default = 11	Option Value TL_C Default = 13
$A \Rightarrow B \Rightarrow C$	11 (\neq default)	11 (= default) ¹⁾	13 (= default) ¹⁾
$A \Rightarrow C$	11 (\neq default)	—	11 (\neq default)

¹⁾ Option value is not stored with the model, because it equals the default.

New Code Generator options

For more details on new Code Generator options, refer to [New Code Generator Options](#) on page 125.

Related topics**References**

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

Other

Various Migration Aspects

TargetLink Main Dialog block	The Compare with reference command has been removed from the context menu of the code generation units listed on the block's Code Generation Page (refer to TargetLink Main Dialog Block ( TargetLink Model Element Reference)) page.
-------------------------------------	--

Converting referenced models	TargetLink's behavior when converting referenced models has changed. Referenced model to subsystem configured for incremental code generation When converting a referenced model to a subsystem configured for incremental code generation, the created subsystem is named as shown in following table:
-------------------------------------	---

Referenced Model Reused?	< TargetLink 4.1	TargetLink 4.1
Yes	-	Subsystem name = Model block name
No	Subsystem name = Model block name	Subsystem name = model name

Accordingly, there is a new storage location for the code that is generated for the model configured for incremental code generation:

Referenced Model Before Conversion (TL4.0 and TL4.1)	Converted Subsystem Configured for Incremental Code Generation (TL4.0)	Converted Subsystem Configured for Incremental Code Generation (TL4.1)
Generated code modules: ▪ <ModelName>.c ¹⁾ ▪ <ModelName>.h ¹⁾ Name of the DD Subsystems object: <ModelName>	Generated code modules: ▪ <ModelBlockName>.c ¹⁾ ▪ <ModelBlockName>.h ¹⁾ Name of the DD Subsystems object: <ModelBlockName>	Generated code modules: ▪ <ModelName>.c ¹⁾ ▪ <ModelName>.h ¹⁾ Name of the DD Subsystems object: <ModelName>

¹⁾ If you did not directly specify a module name at the model's Function block.

Subsystem configured for incremental code generation to referenced model TargetLink sets the simulation mode of the referenced model as shown in the following table:

< TargetLink 4.1	TargetLink 4.1
Accelerator	If the subsystem was converted from a referenced model in TargetLink 4.1, the simulation mode of the reconverted referenced model is the same as the simulation mode of the original referenced model. If the subsystem was converted with another TargetLink version or was not converted at all, the simulation mode of the referenced model is set to Normal .

Related documentation

- [t1_refmodel_to_subsystem](#) ([TargetLink API Reference](#))
- [t1_subsystem_to_refmodel](#) ([TargetLink API Reference](#))

Explicitly modeled saturation involving Float32 and fixed-point

TargetLink cannot guess your intent if you model saturation involving fixed-point or Float32 without using TargetLink's Saturation block (e.g., by using the MinMax block instead). In such cases, keep in mind to check the rounding effects of double values to single precision in Float32 comparisons by the C compiler. If needed as an alternative, either use the Saturation block or specify the correctly rounded single precision value already in the model.

Note that TargetLink's block output code saturation is not affected, because TargetLink internally calculates the correctly rounded single precision value as needed. The changes relate to user-specified values. Also refer to [Rounding of doubles in Float32 comparisons](#) on page 533.

Migrating from TargetLink 3.5 to 4.0

Where to go from here

Information in this section

API Functions.....	343
AUTOSAR-Related Migration Aspects.....	344
Changes With Access Functions.....	347
Code Generator Options.....	350
Messages.....	351
Other.....	352
Stateflow-Related Changes.....	357

API Functions

Changes in TargetLink and TargetLink Data Dictionary API Functions

Renamed API functions

The following TargetLink API functions were replaced:

TargetLink ≤3.5	TargetLink 4.0 ¹⁾
<code>tl_extract_subsystem</code>	<code>tlExtractSubsystem</code> (TargetLink API Reference)
<code>tl_sim_interface</code>	<code>tlSimInterface</code>
<code>tl_switch_blockset</code>	<code>tlOperationMode</code> (TargetLink API Reference)
<code>tl_upgrade</code>	<code>tlUpgrade</code> (TargetLink API Reference)

¹⁾ The commands are case-sensitive.

Related topics

References

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

AUTOSAR-Related Migration Aspects

Where to go from here	Information in this section
	2-D Matrix Data Elements and Operation Arguments..... 344
	Application Data Types..... 345
	AUTOSAR Export..... 345
	Constraints of Array and Matrix Types..... 346
	Replaced IsQueued Property..... 346
	Separate Installation of Container Manager..... 347

2-D Matrix Data Elements and Operation Arguments

Improving code readability using auxiliary pointers

TargetLink supports passing array types via a pointer to the array's first scalar element, as described by the AUTOSAR standard. This holds for parameters and return values of the RTE API.

Code examples: To improve code readability, TargetLink locally introduces the following auxiliary pointers:

2-D array passed to RTE API function

```
Rte_Write_SP_MyDataElem(&A[0][0]);
```

2-D array passed to non-RTE API function

```
CalcSRV(A);
```

2-D array with base type sint16 and 8 columns returned by RTE API function

```
p_MyDataElem =
  (const sint16 (*)[8]) Rte_IRead_Run_RP_MyDataElem();
...
Sa4_Selector = p_MyDataElem[IndexX][IndexY];
```

2-D array with non-RTE API access function of kind ADDRESS

```
Sa4_Selector = (GetAddressOfA())[IndexX][IndexY];
```

Application Data Types

Changes according to AUTOSAR

With TargetLink 4.0 the following changes were made according to the AUTOSAR standard:

The InvalidValue, ScalingRef, andUnitRef properties can only be set at ApplicationDataType objects (ADTs) whose Kind property is set to **Primitive**.

DD Object	DD Property	InvalidValue	ScalingRef	UnitRef
ApplicationDataType objects whose Kind property is set to Array or Record	Forbidden	Forbidden	Forbidden	
ApplicationDataTypeComponent	Removed	Removed	Removed	

Consistency check During the upgrade of a legacy Data Dictionary to the latest data model revision, TargetLink performs a consistency check:

Array ADTs	Record ADTs
<ol style="list-style-type: none"> 1. TargetLink checks whether the values of the InvalidValue, ScalingRef, and UnitRef properties found at an array ADT match the values of these properties at the primitive ADT that is referenced at the ADT. 2. If the values match, the InvalidValue, ScalingRef, andUnitRef properties are unset at the array ADT. 3. If the values do not match, these properties are left untouched. <p>An error occurs during validation.</p>	<ol style="list-style-type: none"> 1. TargetLink checks whether the values of the InvalidValue, ScalingRef, and UnitRef properties found at each of the record ADT ApplicationDataTypeComponent objects match the values of the primitive ADT referenced at each component. 2. If the values match, the InvalidValue, ScalingRef, andUnitRef properties are removed from each component. 3. If the values do not match, these properties remain at each component as custom properties. <p>No error occurs during validation.</p>

Note

If TargetLink detects inconsistencies, resolve the conflicts manually to provide a specification basis according to the AUTOSAR standard.

AUTOSAR Export

Version-specific export

With the introduction of IncludedDataTypeSets in the AUTOSAR 4.x standard, the code representation in the Data Dictionary Subsystems area depends on the AUTOSAR version the code was generated for.

Accordingly, AUTOSAR export from the Data Dictionary complies with the following compatibility matrix:

Code Generated for Version	AR 2.x Export	AR 3.x Export	AR 4.x Export
AUTOSAR 2.x	Possible	Possible	Not possible
AUTOSAR 3.x	Possible	Possible	Not possible
AUTOSAR 4.x	Not possible	Not possible	Possible

Export of 2-D arrays

For AUTOSAR 4.x, data types for 2-D arrays are now exported as nested implementation data types.

Constraints of Array and Matrix Types

Constraints of array and matrix types

For AUTOSAR array/matrix types (i.e., types whose `AutosarArrayWidth` is > 0) that are used with AUTOSAR interfaces, you no longer have to create a `Constraints` object. During code generation, TargetLink draws on the constraints defined at the primitive data type(s).

You still have to create a `Constraints` object for array/matrix types, if:

- The type is used at a non-AUTOSAR interface.
- The type is used in Standard code generation mode.

Replaced IsQueued Property

Property replacement

For `DataElement` objects, the following change was made:

Removed Property	Replacement	Compatibility Setting
<code>IsQueued</code>	<code>ImplementationPolicy</code>	standard if <code>IsQueued</code> was set to off queued if <code>IsQueued</code> was set to on

Note

Data Dictionary files created with TargetLink versions prior to TargetLink 4.0 are automatically migrated.

Separate Installation of Container Manager

Separate installation of dSPACE's Container Manager

dSPACE's Container Manager is now separately installed and will remain on your system if you remove TargetLink.

Providing a custom workflow definition (CTW) file You can provide a custom workflow definition file to change workflow rules.

Note

It is recommended that only experienced users change the workflow rules. There is a special syntax for defining workflow rules.

1. Provide a copy of the CTW template via TargetLink's `t1CustomizationFiles` API function and save it to <MyDir>.
2. Change the CTW file's workflow rules as required.
3. In the Data Dictionary Manager, provide the path to <MyDir>\<CTW file name> as the value of the `WorkflowDefinitionFile` property contained in the `/Pool/Autosar/Config/ContainerExchange/` option set or as the value of the `t1_export_container` API function's `WorkflowDefinitionFile` property.

Related documentation [Exchanging Software Component Containers](#) ([TargetLink Interoperation and Exchange Guide](#))
[Advanced: Configuring Container Handling](#) ([Container Management Manual](#))

Changes With Access Functions

Where to go from here

Information in this section

Changes of ADDRESS_BY_PARAMETER Access Functions	348
New Access-Function-Specific Name Macro	348
New Default Macro Bodies for Macro Access Functions	349

Changes of ADDRESS_BY_PARAMETER Access Functions

Overview of changes

The type of the `_var` macro argument has changed from `scalar type` to `pointer to scalar type` for *vector variables*. This is the case in the following circumstances:

- The DD AccessFunction object has its `Kind` property set to `ADDRESS_BY_PARAMETER`.
- The DD AccessFunction object references a DD FunctionClass object whose `Macro` property is set to `on`.

TargetLink 3.5	TargetLink 4.0
<code>(GetAddressU16(vector[0]))[1] = 5;</code>	<code>(GetAddressU16(vector))[1] = 5;</code>

Note

With TargetLink 4.0 you cannot use the same macro body to access scalars and vectors but have to use a separate macro body per variable kind.

When upgrading an existing Data Dictionary, TargetLink sets *all* empty `Kind` properties to `APPLY_TO_SCALAR`. This affects *both* macro AFs and function AFs.

Tip

If you use TargetLink's predefined access function templates, you can copy the new versions from a DD workspace based on the `dsdd_master_advanced/dsdd_master_autosar3.dd` [System]/`dsdd_master_autosar4.dd` [System] system templates.

New Access-Function-Specific Name Macro

New name macros

TargetLink now has the following new name macros:

Name Macro	Description
<code>\$(Dim1Width)</code> , <code>\$(Dim2Width)</code>	<p>These name macros are allowed in access function templates to specify function names and macro bodies:</p> <ul style="list-style-type: none"> ▪ <code>\$(Dim1Width)</code> - Replaced by the number of elements in the 2-D variable's first dimension (row). ▪ <code>\$(Dim2Width)</code> - Replaced by the number of elements in the 2-D variable's second dimension (column). <p>No effect for scalar variables. For vector variables, only <code>\$(Dim1Width)</code> is applicable.</p>

Name Macro	Description
<code>\$(VarAccess)</code>	This macro is allowed in access function templates to specify macro bodies and gives correct access to the variable concerned, as and when required: For non-struct component variables, it is replaced in the same way as <code>\$v</code> , for struct component variables, it is replaced in the same way as <code>\$(Sv)\$(SPATH).\$v</code> .

Related topics**Basics**[Basics on Using Name Macros \(TargetLink Customization and Optimization Guide\)](#)[Changing Access Function Implementations Using Name Macros \(TargetLink Customization and Optimization Guide\)](#)

New Default Macro Bodies for Macro Access Functions

Default macro bodies for macro access functions

TargetLink provides default macro bodies for macro access functions. These defaults are used in the following circumstances:

- The DD AccessFunction object references a DD FunctionClass object whose Macro property is set to on.
- The DD AccessFunction object's MacroBody property is empty.

To increase MISRA C compliance, the following placeholders for macro arguments are enclosed by brackets:

- `_var`
- `_value`

The example macro bodies of the predefined access function templates contained in the dsdd_master_advanced, dsdd_master_automotive3.dd [System], and dsdd_master_automotive4.dd [System] Data Dictionary Manager system templates were adapted accordingly.

Related topics**Basics**[Basics on Access Functions \(TargetLink Customization and Optimization Guide\)](#)[Specifying Access Functions as Preprocessor Macros \(TargetLink Customization and Optimization Guide\)](#)

Code Generator Options

Migration Aspects Regarding Code Generator Options

Renamed Code Generator options

The following Code Generator options were renamed:

Old Name	New Name
NoAssignmentOfBooleanExpressions	AssignmentOfConditions (TargetLink Model Element Reference)
PolySpaceSupport	InsertComputeThroughOverflowComments (TargetLink Model Element Reference)
DisableArbitraryOptimizations	Allow64BitMultiplicationsForArbitraryScaled16BitOperands (TargetLink Model Element Reference)

Recommended compatibility settings for new Code Generator options

For the best downward compatibility to TargetLink 3.5, it is recommended to use the *Compatibility Setting* values for the new Code Generator options listed in the following table:

Code Generator Option	Compatibility Setting	Default Value
AssignmentOfConditions (TargetLink Model Element Reference)	3 - AllOutputs ¹⁾ 0 - None ²⁾ on ⁴⁾ off ⁵⁾	2 - AllBooleanOutputs on
Allow64BitMultiplicationsForArbitraryScaled16BitOperands (TargetLink Model Element Reference) ³⁾		

¹⁾ When Generate optimized code (refer to Optimization (Model element) ([TargetLink Model Element Reference](#))) is set to on.

²⁾ When Generate optimized code (refer to Optimization (Model element) ([TargetLink Model Element Reference](#))) is set to off.

³⁾ Substitutes DisableArbitraryOptimizations

⁴⁾ When DisableArbitraryOptimizations was set to off in TargetLink 3.5.

⁵⁾ When DisableArbitraryOptimizations was set to on in TargetLink 3.5.

For TargetLink versions < 3.5, refer to the versions' *New Features and Migration* document. For details, refer to [Previous release documents](#) ([New Features and Migration](#)).

Removed Code Generator option

The following Code Generator option was removed from TargetLink:

Removed Option	Replacement Option	Compatibility Setting
EfficientVectorHandling	LoopUnrollThreshold (TargetLink Model Element Reference) ¹⁾ .	INF, if EfficientVectorHandling was set to off

- ¹⁾ For details, refer to [Removal of EfficientVectorHandling](#) on page 542 and [Basics on Processing Vectors and Matrices](#) ([TargetLink Preparation and Simulation Guide](#)).

New Code Generator options

For more information on new Code Generator options, refer to [New Code Generator Options](#) on page 151.

Messages

Message Changes

Changed message type

The following messages changed their type to **Advice** in TargetLink 4.0 because they only inform you about possible code efficiency improvements, but can be safely ignored:

Old Message Number	New Message Number
W15658	A15658
N17350	A17350
W17352	A17352
W17358	A17358
W17356	A17356

Other

Where to go from here

Information in this section

Plot Channel Specification.....	352
Signal Properties Inheritance.....	353
Stricter Settings for Bus Diagnostics.....	353
Various Migration Aspects.....	355

Plot Channel Specification

New method of plot channel specification

For matrix signals, TargetLink provides two ways to specify plot channels:

- Linear indexing of matrix signal elements (in column major order)
- One-based index pair for each matrix signal element ([`row column`]).

Note

We recommend to specify the plot channels of matrix signals using the index pair approach.

Example of the index pair approach The plot channels [`1,4; 2,3; 5,6`] specify to plot the three matrix elements (1,4), (2,3), and (5,6) during simulation.

Migration consideration

Note

For matrix signals, the specification of a plot channel containing *exactly one* pair of one-based indices is ambiguous because it can stand for one of the following:

- Two matrix signal elements (linear index column-wise)
- One matrix signal element (index pair row/column).

TargetLink resolves this ambiguity by interpreting such a plot channel specification as being an index pair.

For models created with a TargetLink version ≤ 3.4 , a problem can occur if the model contains row matrices ([`1xn`]) or column matrices ([`mx1`]):

Problem	Description
One signal plotted instead of two	TargetLink blocks connected to row/column matrix signals plot one instead of two signals.

Problem	Description
Invalid plot channels	TargetLink detects invalid plot channels and displays warning W02441.

Example of invalid plot channels A [6x1] matrix signal is connected to a TargetLink block whose plot channels are defined as [3 5].

Solution

Often, the problems result from signals originating from InPort blocks or Constant blocks:

Source	Solution
InPort block	Explicitly specify the signal's dimension to make it a vector.
Constant block	Select the Interpret as 1-D checkbox.

Signal Properties Inheritance

Inheritance of different data types from preceding blocks

TargetLink's code generator no longer allows to mux or concatenate several signals of different data types if these types have different base types.

Example Block A and block B have output variables of different data types. Their output signals are muxed and fed into block C, whose signal inheritance is enabled:

TargetLink ≤ 3.5	TargetLink 4.0
Two output variables for block C, each having one of the inherited data types	E15738

Solution Use a Bus Creator block instead of a Mux block.

Stricter Settings for Bus Diagnostics

Automatic configuration adaptation

TargetLink might automatically change the values of some Simulink configuration parameters in the following circumstances:

- During model preparation
- During model upgrade
- When a TargetLink Main Dialog block is added to a model
- When a MIL Handler block is added to a model

This lets you use the full range of TargetLink MIL simulation features such as logging and overflow detection.

Simulink Configuration Parameter	Value Set By TargetLink
Element name mismatch	error ¹⁾
Mux blocks used to create bus signals	error ¹⁾
Bus signal treated as vector	error ¹⁾
Non-bus signals treated as bus signals	error ¹⁾
Signal storage reuse	off
Block reduction	off
Signal logging format	Dataset

¹⁾ If you reset this preference to a value other than 'error', TargetLink displays message E02462 at simulation start or during code generation.

Simulink Configuration Parameter	Value Set By TargetLink
Element name mismatch	error ¹⁾
Mux blocks used to create bus signals	error ¹⁾
Bus signal treated as vector	error ¹⁾
Non-bus signals treated as bus signals	error ¹⁾
Signal storage reuse	off
Block reduction	off
Signal logging format	Dataset

¹⁾ If you reset this preference to a value other than 'error' TargetLink displays message E02462 at simulation start/during code generation.

Required model parameters

Several Simulink Model Configuration parameters located in Diagnostics - Connectivity - Buses must be set according to the following table:

Parameter	Setting
Element name mismatch	error
Mux blocks used to create bus signals	error
Bus signal treated as vector	error
Non-bus signals treated as bus signals	error

TargetLink checks for these parameter settings during model initialization. If these settings do not exist, TargetLink displays error message E02462 and code generation is not possible.

The following TargetLink MIL simulation capabilities are then locked as well and message E02464 will be displayed:

- Signal logging
- Min/max logging
- Signal plotting
- Overflow detectoin

API To set these properties via the API using Simulink's `get_param`/`set_param` API functions, make the following settings:

Property	Value
StrictBusMsg	'ErrorOnBusTreatedAsVector'
NonBusSignalsTreatedAsBus	'error'
BusObjectLabelMismatch	'error'

Note

Simulink's Bus to Vector block conflicts with the bus signal treated as vector and non-bus signals treated as bus signal parameter settings. Do not use this block.

Various Migration Aspects

OpenFcn callbacks

Simulink does not allow using OpenFcn callbacks that directly or indirectly change the Commented block property. Because TargetLink 4.0 uses the Commented block property when switching the TargetLink subsystem to SIL or PIL simulation mode, custom buttons created, for example, to start code generation no longer work.

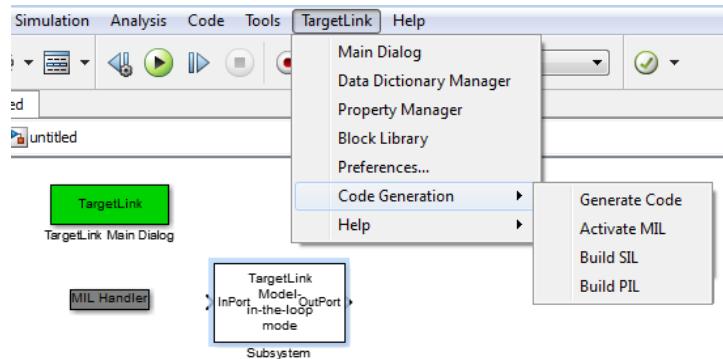
The following API functions change the Commented block property and cannot be used directly or indirectly in OpenFcn callbacks:

- `tl_generate_code`
- `tl_build_host`
- `tl_build_target`
- `tl_set_sim_mode`

Solution Several workarounds are possible:

- Use the following Utility blocks (refer to The TargetLink Simulation Blocks and Their TargetLink Properties ([TargetLink Model Element Reference](#))):
 - **MIL Mode Block** ([TargetLink Model Element Reference](#))
 - SIL mode block
 - Tool Selector block

- Use the Simulink Editor's TargetLink menu:



- Use the controls of the TargetLink Main Dialog block.
- Use annotations with ClickFcn callbacks:
 1. Double-click an empty area within the Simulink Editor to produce an annotation.
 2. Enter text as required.
 3. Select the annotation and select Properties... from its context menu. The Annotation properties dialog opens.
 4. In the ClickFcn group box, specify the command to be run by the annotation.
 5. In the Appearance group box, specify a button-like appearance for the annotation.
 6. Click OK to close the dialog.You can now run your command by single-clicking the annotation.
- Write your own GUI with controls that start the code generation process.
- Create your own Simulink menu to start your scripts as described in the Simulink Documentation

Related topics

References

- MIL Mode Block ([TargetLink Model Element Reference](#))
- SIL Mode Block ([TargetLink Model Element Reference](#))
- TargetLink Main Dialog Block ([TargetLink Model Element Reference](#))
- Tool Selector Block ([TargetLink Model Element Reference](#))

Stateflow-Related Changes

Where to go from here

Information in this section

Exported Graphical Functions.....	357
Stateflow Matrices.....	357

Exported Graphical Functions

Simulation in SIL/PIL not possible

Due to improvements in TargetLink's simulation frame, Exported Graphical Functions used outside of but defined within the TargetLink subsystem cannot be simulated in SIL or PIL simulation mode.

This is the case because the TargetLink subsystem is commented in SIL or PIL simulation mode.

Code generation

TargetLink can generate production code for Exported Graphical Functions.

Stateflow Coder cannot build Exported Graphical Functions defined in commented model parts. Accordingly, it cannot build Exported Graphical Functions defined in a TargetLink subsystem set to SIL or PIL simulation mode.

Modeling recommendation

If you solely want to use Exported Graphical Functions *outside* of TargetLink subsystems, do *not* define them *within* TargetLink subsystems. This avoids dead code.

Stateflow Matrices

Loop generation

TargetLink now uses the same mechanisms and optimizations for Stateflow and Simulink matrices. Accordingly, both dimensions can now be rolled into loops in the generated production code.

TargetLink 4.0	TargetLink 3.x
Outer dimension's iteration generated as <code>for</code> loop Rolling depends on <code>LoopUnrollThreshold</code>	Outer dimension's iteration generated as <code>do-while</code> loop Not subject to <code>LoopUnrollThreshold</code>

Possible code changes

- Merged loops
 - Other auxiliary variables for vectors/matrices
 - Former `do while` loops can now be `for` loops.
-

Related topics

Basics

[Basics on Processing Vectors and Matrices](#) ( TargetLink Preparation and Simulation Guide)

[Optimizing the Production Code](#) ( TargetLink Customization and Optimization Guide)

Migrating from TargetLink 3.4 to 3.5

Where to go from here

Information in this section

API Functions and Hook Scripts.....	359
AUTOSAR-Related Migration Aspects.....	360
Documentation.....	361
Other.....	363

API Functions and Hook Scripts

Where to go from here

Information in this section

Changes in TargetLink and TargetLink Data Dictionary API Functions.....	359
Hook Script Templates.....	359

Changes in TargetLink and TargetLink Data Dictionary API Functions

Deprecated commands

tl_find_system This command is deprecated. It is only needed with MATLAB prior to Release R2011b. With newer MATLAB releases you can use the shipped Simulink `find_system()` command for TargetLink blocks as well.

Hook Script Templates

Hook script templates

With TargetLink 3.5, hook script (and also other) templates are shipped as SAM files and no longer reside on the search path. For details on how to convert hook script templates to M files and place them on TargetLink's search path using the `tlCustomizationFiles('Create',...)` API function, refer to the following documentation:

Related documentation

- [How to Create Customization Files via the Create Customization Files Dialog](#)
([TargetLink Customization and Optimization Guide](#))
- [How to Define TargetLink's Search Path for DD-Related Customization Files](#)
([TargetLink Customization and Optimization Guide](#))

AUTOSAR-Related Migration Aspects

AUTOSAR-Related Migration Aspects

Improved AUTOSAR synchronization functionality

TargetLink's AUTOSAR synchronization functionality has been improved and the synchronization scripts used in prior versions have been removed. To ease access to the improved functions, they were added to the relevant context menus in the /Pool/Autosar subtree.

The following table lists the new M-API functions and the corresponding DD context commands:

New API Function	New DD Context Command Command	Context Menu Of
[bSuccess,processedObjectInfo] =... dsdd('SyncAutosar',<objectIdentifier>... [,<attributeName>,<attributeValue>]); ¹⁾	Synchronize Application Data Type Settings	/Pool/Autosar subtree
[bSuccess,processedObjectInfo] =... dsdd('CreateVariableObjects',<objectIdentifier>... [,<bDeleteObsoleteInterfaceElements>,<bVerbose>]); ²⁾	Synchronize Interface Settings	DD RequireCalPrmPort objects
[bSuccess,processedObjectInfo] =... dsdd('CreateRteModes',<objectIdentifier>[,<bVerbose>]); ³⁾	Synchronize Mode Settings	DD ModeDeclarationGrou p objects
[bSuccess,processedObjectInfo] =... dsdd('SyncTypes',<objectIdentifier>... [,<attributeName>,<attributeValue>]); ⁴⁾	Adjust to Typedef	Data prototype objects

¹⁾ Replaces dsdd_sync_autosar4 synchronization script.

²⁾ Replaces dsdd_sync_calprms_ports synchronization script.

³⁾ Replaces dsdd_sync_modedeclarations synchronization script.

⁴⁾ Replaces dsdd_sync_typestruct2object synchronization script.

New NVDATA synchronization TargetLink now provides a functionality to synchronize NVDATA settings:

Function	Access	Description
Synchronize NvDataInterface Settings ¹⁾	Context menu of ports belonging to DD NvDataInterface	Synchronizes NvDataInterface settings.

- ¹⁾ Corresponds to the M-API function [bSuccess,processedObjectInfo] = dsdd('CreateVariableObjects',<objectIdentifier>...[,<bDeleteObsoleteInterfaceElements>,<bVerbose>]);.

New properties in ImportExport AutosarOptionSet New properties were added to the /Pool/Autosar/Config/ImportExport AutosarOptionSet. These let you specify which synchronization commands are executed during imports of AUTOSAR data into the DD:

Property	Description
SynchronizeCalPrmInterfaceSettings	If on, synchronizes CalPrmInterface settings during import.
SynchronizeNvDataInterfaceSettings	If on, synchronizes NvDataInterface settings during import.
SynchronizeModeSettings	If on, synchronizes Mode settings during import.
SynchronizeApplicationDataTypeSettings	If on, synchronizes ApplicationDataType settings during import.

New AutosarOptionSet There is a new object called /Pool/Autosar/Config/ContainerExchange DD AutosarOptionSet. Its PerformA2LExportOnContainerExport property lets you specify to export an A2L file during container exchange.

You can copy the option set from a DD workspace based on the dsdd_master_autosar.dd template or create it via the Create AutosarOptionSet command from the context menu of the DD /Pool/Autosar/Config subtree in your existing DD files.

Deprecated hook script The t1_post_arimport_hook_sync_calprm.m hook script is deprecated and was removed from the TargetLink installation.

Documentation

Documentation Changes

User documentation renamed

To facilitate finding user documentation in the Print folder of dSPACE HelpDesk, the PDF file names have been changed to match the titles of the documents. The modifications are also applied to the related CHM files in the Online folder.

For TargetLink, the following documents are relevant:

Guides		
Title	Old Name	New Name
TargetLink Multirate Modeling Guide	TLMultirateGuide.pdf	TargetLinkMultirateModelingGuide.pdf
TargetLink AUTOSAR Modeling Guide	TLAUTOSARGuide.pdf	TargetLinkAUTOSARModelingGuide.pdf
TargetLink Blockset Guide	TLBlocksetGuide.pdf	TargetLinkBlocksetGuide.pdf
TargetLink Data Dictionary Basic Concepts Guide	DDGuide.pdf	TargetLinkDataDictionaryBasicConceptsGuide.pdf
References		
Title	Old Name	New Name
TargetLink Block and Object Reference	TLBlockRef.pdf	TargetLinkBlockAndObjectReference.pdf
TargetLink Tool and Utility Reference	TLToolRef.pdf	TargetLinkToolAndUtilityReference.pdf
TargetLink API Reference	TLAPIRef.pdf	TargetLinkAPIReference.pdf
TargetLink File Reference	TLFileRef.pdf	TargetLinkFileReference.pdf
TargetLink Demo Models	TLDemoModels.pdf	TargetLinkDemoModels.pdf
TargetLink Evaluation Board Hardware Reference	TLTargetRef.pdf	TargetLinkEvaluationBoardHardwareReference.pdf
TargetLink Data Dictionary Manager Reference	DDRef.pdf	TargetLinkDataDictionaryManagerReference.pdf
Import Export Documents		
Title	Old Name	New Name
TargetLink Data Dictionary XML Import and Export	DDImportExport1.pdf	TargetLinkDataDictionaryXMLImportAndExport.pdf
TargetLink Data Dictionary ASAM MCD-2 MC Import and Export	DDImportExport2.pdf	TargetLinkDataDictionaryA2LImportAndExport.pdf
TargetLink Data Dictionary OIL Import and Export	DDImportExport3.pdf	TargetLinkDataDictionaryOILImportAndExport.pdf
TargetLink Data Dictionary AUTOSAR File Import and Export	DDImportExport4.pdf	TargetLinkDataDictionaryAUTOSARFileImportAndExport.pdf

New user guides

With TargetLink 3.5, the structure of the user documentation has been improved. To facilitate access to the documentation, the topics are now grouped by their use case.

As a result, new use-case-specific guides are being introduced:

- The all-new  [TargetLink Orientation and Overview Guide](#)

This guide contains basic information for all people who work with TargetLink, from beginners up to expert users. It provides an introduction to TargetLink and points you to the user documentation that applies to your use case. It also lists TargetLink limitations and provides a glossary of terms used throughout the TargetLink documentation.

-  [TargetLink Preparation and Simulation Guide](#)

This guide shows you how to use TargetLink out-of-the-box for preparing Simulink models for code generation and for simulating and testing the generated production code.

-  [TargetLink Customization and Optimization Guide](#)

This guide shows you how to adapt TargetLink settings so that the generated production code meets your specific requirements.

-  [TargetLink Interoperation and Exchange Guide](#)

This guide shows you how to use TargetLink as part of a tool chain, for example, exchanging data with other tools or using TargetLink via command line.

Discontinued guides

The following guides have been discontinued. They are substituted by new use-case-oriented guides.

Previous Guides	New Guides
TargetLink Production Code Generation Guide	TargetLink Preparation and Simulation Guide
TargetLink Advanced Practices Guide	TargetLink Customization and Optimization Guide
	TargetLink Interoperation and Exchange Guide

Other

Various Migration Aspects

System preparation

The Replace blocks according to libmaps option is no longer available. As a result, if there are libmaps that define block replacements, these replacements are always performed. With TargetLink 3.5, you can no longer suppress them by clearing this option, which was only helpful for debugging libmaps. As a workaround, you can add a return at the beginning of a libmap, or remove the libmap file from the path, or rename it.

This option was used in the `t1_prepare_system` API function and in the [System Preparation Dialog](#) ( [TargetLink Tool and Utility Reference](#)).

Restoring demo models to their original state

You can restore a demo model in the selected root folder with the `t1_demos [<DemoName>] -restore` command. If you use the command without <DemoName>, all demo models are restored to their original state.

A2L export

A2L export now consists of two phases instead of three. For details, refer to [Basics on Customizing the A2L Export](#) ([TargetLink Interoperation and Exchange Guide](#))

Changes in documentation generation

The mechanism of TargetLink's documentation generation has been changed. For details, refer to [Generating Documentation on Model Characteristics](#) ([TargetLink Interoperation and Exchange Guide](#)) and [Maintaining and Documenting](#) ([TargetLink API Reference](#)).

Related topics

References

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

Migrating from TargetLink 3.3 to 3.4

Where to go from here

Information in this section

API Functions.....	365
AUTOSAR-Related Migration Aspects.....	365
Code Generator Options.....	366
Other.....	367

API Functions

Changes in TargetLink and TargetLink Data Dictionary API Functions

Obsolete DD MATLAB API functions

The following DD MATLAB API function is no longer supported:

- **`dsdd_upgrade`**

To upgrade a DD project file, use the `dsdd('Upgrade'[,<DD_Identifier>])` command (refer to [Upgrade](#)), or the DD Manager.

AUTOSAR-Related Migration Aspects

AUTOSAR-Related Migration Aspects

Validation of AUTOSAR data

The validation of AUTOSAR data was significantly improved. In particular, badly/insufficiently specified ComSpecs are now detected.

This improves data exchange with system architecture tools such as SystemDesk and third-party RTE generators.

AUTOSAR File Import/Export	<p>The following two properties are obsolete:</p> <ul style="list-style-type: none">▪ Merge▪ EnablePackageSupport <p>The value of both properties is now always set to on.</p> <p>Adapt existing scripts accordingly.</p>
RTE frame module	<p>The default name of RTE frame modules has changed from Rte_<u>\$N</u> to Rte.</p> <p>It is no longer possible to generate additional variables or functions into the RTE frame module. Contact dSPACE Support (www.dspace.com/go/supportrequest) concerning the migration of TargetLink models containing workarounds for RTE API functions unsupported in earlier TargetLink versions.</p>

Code Generator Options

Migration Aspects Regarding Code Generator Options

Basics on default changes	<p>The settings of the Code Generator options are stored with the model (model-based option storage). In addition, you can store user-defined sets of Code Generator options in OptionSets in the TargetLink Data Dictionary since TargetLink version 3.1 (DD-based option storage). You can use DD-based option settings as a central source for overwriting model-based option settings.</p> <p>If a model-based option value equals the default value of an older TargetLink version, it is changed to the new default value during upgrade. If a DD-based option value equaled the old default value, it is not changed to the new default value during upgrade but keeps the old value.</p> <p>For details on both storage kinds, refer to Basics on Configuring the Code Generator for Production Code Generation ( TargetLink Customization and Optimization Guide).</p>
----------------------------------	--

Changed default values of Code Generator options	The default values of the Code Generator options listed below were changed:		
Code Generator Option	Default Value TargetLink 3.4	≤ TargetLink 3.3	
AllowInterleavingCodeForAllSubsystems	on	off	
ExploitRangesIndependentOfErasable	off	on	

Code Generator Option	Default Value	
	TargetLink 3.4	≤ TargetLink 3.3
GlobalOptimizationIterationThreshold	30	10
OptimizationIterationThreshold	50	10
SideEffectFreeAnalysisThreshold	1000	10

Recommended compatibility settings for new Code Generator options For the best downward compatibility to earlier TargetLink versions (≤ TargetLink 3.3), it is recommended to use the *Compatibility Setting* values for the new Code Generator options listed in the following table:

Code Generator Option	Compatibility Setting	Default Value
AllowStructAssignments	off	on
AssumeOperationCallsHaveNoUnknownDataFlow	off	on
UtilizeValueEqualitySignalLineSplit	off	on
UtilizeValueEqualityStructFunctionArguments	off	on
ExploitComputeThroughOverflow	3 = Always	2 = Optimized

Related topics

References

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

Other

Various Migration Aspects

Style definition file cconfig.xml

For the %DSPACE_ROOT%\Matlab\T1\config\codegen\cconfig.xml style definition file, the <TL:generation-date ...> element was renamed.

The element now reads as follows: <TL:generation-date-comment ...>.

Note

If you use your own style definition file, you must adapt it.

The style definition file is referenced in the TargetLink Main Dialog block on the Advanced page.

Migrating from TargetLink 3.2 to 3.3

Where to go from here

Information in this section

API Functions.....	368
AUTOSAR-Related Migration Aspects.....	370
Code Generator Options.....	370
Other.....	372

API Functions

Changes in TargetLink and dSPACE Data Dictionary API Functions

Changed API functions

In TargetLink 3.3, the behavior of the following TargetLink/dSPACE Data Dictionary API functions has changed slightly.

- **SetAccessRights**

The **SetAccessRights** API function has changed. As of TargetLink 3.3, this API function no longer returns an error code but the number of objects that were changed. For details, refer to [dSPACE Data Dictionary MATLAB API Reference > DD Matlab API Functions > Generic Commands > SetAccessRights](#).

- **filename attribute**

Previous TargetLink versions could only load one DD project file to memory at a time. The name of the currently loaded file was written to the **ProjectFile** environment variable. As of TargetLink 3.3, the Data Dictionary can load multiple DD project files simultaneously, each to its own DD workspace. The Data Dictionary supports an arbitrary number of DD workspaces (DD 0 ... DD n), each of which is represented by a *DD Workspace pane*. The name of the DD file loaded to a DD workspace is now an attribute of that workspace.

As a consequence, read or write access to the project file via the `dsdd` API function has changed as shown below:

< TargetLink 3.3	>= TargetLink 3.3
<pre>projectFile = dsdd('GetEnv','projectFile'); dsdd('SetEnv','projectFile',projectFile);</pre>	<pre>projectFile = dsdd('GetDDAttribute','DD0','file Name'); dsdd('SetDDAttribute','DD0','fileNa me',projectFile);</pre>

The commands `dsdd('GetEnv', 'projectFile');` and `dsdd('SetEnv', 'projectFile', projectName);` are mapped to the new commands. in addition, a warning is displayed in the MATLAB Command Window

Accessing Stateflow charts via API

Up to TargetLink 3.3, the TargetLink API accepted both the handle and the ID of a Stateflow chart to access Stateflow chart objects. As of TargetLink 3.3, the Stateflow chart handle can no longer be used for API requests. If you have used TargetLink API functions consistently/exclusively, you do not have to adapt existing scripts. However, if you used hybrid forms, i.e., TargetLink and Simulink API functions as shown in Discontinued Access Approach, problems may occur that make it necessary to adapt existing scripts.

Discontinued Access Approach	Supported Access Approach
<pre>h = find_system('sf_demo', 'MaskType' , 'Stateflow'); tl_get(h, 'stepfunctionclass');</pre>	<pre>h = tl_get_sfobjects('sf_demo', 'SF Chart'); tl_get(h, 'stepfunctionclass');</pre>

Obsolete DD MATLAB API functions

The following DD MATLAB API functions are no longer supported:

- `dsdd_online_help`
Has been replaced by `tl_online_help`
- `GetDDInfo`
Has been replaced by `GetDDAttributes`
- `GetCallback`
- `SetCallback`

Callbacks, i.e., delete callbacks, no longer exist.

AUTOSAR-Related Migration Aspects

AUTOSAR-Related Migration Aspects

Changed DD objects for mode management This version of TargetLink supports the RTE API for mode management. DD objects that relate to mode management have been added or changed.

This affects the following DD objects:

- Mode disabling dependencies (changed)
- Mode switch interfaces (new)
- Mode sender/mode receiver ports (new)
- Mode elements (changed)
- Sender-receiver interfaces (changed)

Suggested migration for DD API functions Adapt MATLAB API functions that create or change DD objects that are related to mode management.

Suggested migration of the Data Dictionary The DD upgrade process changes the existing elements that are related to mode management. However, the upgrade process can lead to an invalid Data Dictionary due to insufficient specification of mode management in the old Data Dictionary.

To migrate Data Dictionaries with mode management, perform the following steps:

1. Validate the Data Dictionary after upgrading to check whether insufficient specification of mode communication has led to an invalid Data Dictionary.
2. Add required specifications if necessary.

Code Generator Options

Migration Aspects Regarding Code Generator Options

Obsolete Code Generator option The following Code Generator option is no longer used in TargetLink 3.3:

- OptimizationLevel

This option was replaced by the new Optimization option.

Old optimization levels 0, 1, 2 have been replaced with Optimization=on (formerly 1,2) and Optimization=off (formerly 0). Basically, you do not have to do anything unless you use a *pre_codegen_hook* file to set an optimization level directly. If so, replace the old values (0, 1, 2) with the new ones (*off*, *on*). For details, refer to [Optimization \(Model element\)](#) ( [TargetLink Model Element Reference](#)).

If you generate code while the obsolete options are still in the model, a warning occurs.

Changed Code Generator option The Code Generator option listed below was changed. The default value has not changed and the new values are italicized.

	Description	Explanation	Default Value
ShiftMode			
	Controls the shift mode	<p>TargetLink tries to replace multiplications or divisions with shift operations. This can be controlled by the ShiftMode option. Shift operations explicitly defined in Stateflow or in an Arithmetic Shift block can also be controlled with this option.</p> <ul style="list-style-type: none"> ▪ 0 = Automatic: Use shift operations where possible (default). ▪ 1 = Don't shift right: Divisions are not replaced with right shifts and explicitly defined right shifts are replaced with divisions. ▪ 2 = Don't shift left: Multiplications are not replaced with left shifts and explicitly defined left shifts are replaced with multiplications. ▪ 3 = Don't shift at all: Suppress all shift operations. ▪ 17 = <i>Don't shift right, keep explicit shift operations:</i> <i>Divisions are not replaced with right shifts and explicitly defined right shifts are kept.</i> ▪ 18 = <i>Don't shift left, keep explicit shift operations:</i> <i>Multiplications are not replaced with left shifts and explicitly defined left shifts are kept.</i> ▪ 19 = <i>Don't shift at all, keep explicit shift operations:</i> <i>Suppress all shift operations but keep explicitly defined shifts.</i> 	0

Recommended compatibility settings for new Code Generator options For the best backward compatibility to earlier TargetLink versions, it is recommended to use the *Compatibility Setting* values for the new Code Generator options listed in the following table:

Code Generator Option	Compatibility Setting	Default Value
ExtendedLifeTimeOptimization	off	on
GenerateFrameVariablesForAccessFunctions	on	off

Related topics

References

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

Other

Various Migration Aspects

Changed behavior when specifying module ownership

Up to now, you could specify the module ownership (List of modules owned by this function (without extension)) for subsystems configured for incremental code generation via the TargetLink Function block. If you did so and if no corresponding `ModuleOwnership` object existed in the Data Dictionary, only a warning (W15712) was displayed that the module ownership specification would not be used in all cases.

As of TargetLink 3.3, module ownership specifications made for subsystems configured for incremental code generation via the TargetLink Function block are ignored. This is a behavioral change, and the following error message is generated instead of a warning.

E15723 The module ownership is only specified at the Function block and not via a `ModuleOwnership` object in the Data Dictionary, and is therefore ignored. Specify the module ownership via a `ModuleOwnership` object in the Data Dictionary instead.

You can use the following methods to specify the module ownership in the dSPACE Data Dictionary:

- Precondition: The associated DD is loaded.
 1. Open the Function block and select the Incremental page.
 2. Right-click the List of modules owned by this function (without extension) edit field.

Select Create ModuleOwnership in DD from the context menu.

The specified ModuleOwnerShip object is created.

- Precondition: The model is open and the associated DD is loaded.
- 1. In the MATLAB Command Window, use the `tl_check_module_ownership` API function with its parameters set accordingly. For example: `tl_check_module_ownership('Model', 'poscontrol', 'CreateModuleOwnerShip', 'on', 'CreateModules', 'on', 'SearchInRefModels', 'on')`.

Look-Up Tables

In analogy to MATLAB R2011a, the Look-Up Table 1D and Look-Up Table 2D blocks in TargetLink 3.3 provide a reduced set of look-up methods. The following table shows the mapping of the old and the new look-up methods with their Simulink counterparts.

TargetLink 3.2 and Older	TargetLink 3.3	Simulink
Interpolation – Extrapolation	Interpolation – Extrapolation	Interpolation method = Linear Extrapolation method = Linear
Interpolation – Use End Values	Interpolation – Use End Values	Interpolation method = Linear Extrapolation method = Clip
Use Input Below	Use Input Below	Interpolation method = Flat
Use Input Nearest		
Use Input Above		

Note

During migration from TargetLink 3.2 to TargetLink 3.3, the old Use Input Nearest and Use Input Above look-up methods are substituted by the Use Input Below look-up method. You therefore have to adjust the axis (input) values.

Workaround for Use Input Nearest and Use Input Above You can mimic the old Use Input Nearest and Use Input Above look-up methods by using the Use Input Below interpolation method and applying a constant shift to the axis (input) values, which must be equidistant.

Requirements information in the documentation

If requirement information is set at the import block of a subsystem configured for incremental code generation, requirement information generation in the documentation has changed. The section Requirements Referenced in Generated Code, which is generated for the system surrounding the incremental subsystem, now states the path for this requirement information as a Simulink/Stateflow path relative to the incremental code generation unit, instead of relative to the surrounding code generation unit as previously.

Refined code generation	<p>You can generate production code via the TargetLink – Code Generation – Generate Code menu command. This now generates context-based production code, i.e., only for the system or subsystem that code generation was started from.</p> <p>Suppose your model consists of two subsystems (<i>SS1</i> and <i>SS2</i>), and <i>SS2</i> itself contains a subsystem (<i>SS3</i>) that is configured for incremental code generation. If you generate production code for <i>SS2</i> via TargetLink – Code Generation – Generate Code, no production code is generated for <i>SS3</i> because it is configured for incremental code generation. To generate production code for <i>SS3</i>, you have to execute the menu command on it directly. This contrasts with previous TargetLink versions, which generate code for all subsystems.</p>
Scalar expansion and Same scaling renamed	<p>The previous terms Scalar expansion and Same scaling have been changed to Uniform elements.</p> <p>Uniform elements indicates whether the same LSB, offset, Min, and Max values are applied to all of the vector elements. The width of a vector must therefore no longer be specified explicitly. If different scaling data exists and you select this checkbox, the scaling data currently displayed is applied to all of the vector elements.</p>
Scope reduction for implicit variable	<p>If you use a read access function for a TargetLink Import block that references a function class whose Optimization property is not set to SIDE_EFFECT_FREE, then TargetLink tries to reduce the number of function calls to one by introducing a global implicit variable that can be used by all the functions in the generated code. Whenever possible, TargetLink tries to reduce the scope of this variable appropriately, i.e., according to the context the variable is used in. For example, if the Code Generator can reduce the Scope to local, it does so. As a result, optimal scope reduction is now performed for such variables.</p>
Error caused by an incorrect sample time	<p>If there is a block in your TargetLink model whose sample rate is set to zero, as of now an error message is displayed instead of a warning. This change is caused by the <i>multirate, single task</i> support (also referred to as generic multirate code generation) in the Standard and AUTOSAR code generation modes, which cannot generate code for blocks with a sample time of zero. If an error message is displayed, correct the block settings accordingly (<i>Sample time (-1 for inherited)</i>).</p>
New error message E03156: BUS PORT WITH EXPLICIT WIDTH	<p>If you set an explicit (fixed) width on the Output page of a Bus Import or Bus Outport block, the following error message is displayed during code generation: <i>E03156: BUS PORT WITH EXPLICIT WIDTH: Port block passes a bus signal, and its port dimensions are explicitly set (not -1)</i>. Previous TargetLink versions do not generate this error message.</p>

Microsoft Visual Studio compiler version	<ul style="list-style-type: none">▪ Microsoft Visual Studio compiler version 6 used as a MEX compiler is not supported for 64-bit versions of TargetLink.▪ Microsoft Visual Studio compiler version 8 used as a MEX compiler is not supported for 64-bit versions of TargetLink.
No A2L file import in a 64-bit version of TargetLink	It is not possible to import an A2L file into a 64-bit version of TargetLink.
Changed mechanism for deriving bus signal names	<p>When migrating to Simulink 7.6 from earlier Simulink versions, a problem can occur with the names of bus signals:</p> <ul style="list-style-type: none">▪ Before version 7.6, Simulink primarily derived bus signal names from bus objects and TargetLink used these names accordingly.▪ From version 7.6 on, Simulink exclusively derives bus signal names from the labels assigned to each signal. TargetLink basically extracts the signal labels from the value of the BusHierarchy property. If no label exists, a generic label (signal1, signal2, ..., signal<n>) is generated. <p>The problem only occurs if the names of labels and the names in the bus object are different, which most likely is the case if no labels were assigned during creation of the model.</p> <p>Note</p> <p>To avoid a naming conflict, you have to ensure that the labels and the names contained in the Busobject match, for example, by using Simulink's Element name mismatch check.</p>

Related topics

References

[tl_check_module_ownership](#) ( TargetLink API Reference)

Migrating from TargetLink 3.1 to 3.2

Where to go from here

Information in this section

API Functions.....	376
AUTOSAR-Related Migration Aspects.....	378
Code Generator Options.....	380
Other.....	383

API Functions

Changes in TargetLink API Functions

Changed API functions

In TargetLink 3.2, the behavior of the following TargetLink API function has slightly changed.

- **tl_get_blocks**

This command now enables you to make *case-sensitive* searches for port blocks. In addition, it now returns the block type which is useful if the search criteria involves several block types, for example, **TLSim**.

```
[hBlocks, blockTypes] = tl_get_blocks('pipt1', 'TLSim')
```

You can search for TargetLink blocks and supported (enhanced) Simulink blocks. The search criteria can be the block name (or a list of block names) used for the block in the TargetLink block library (e.g. **Gain**), or the block type (e.g. **TL_Gain**).

The command returns a cell array containing associated block type(s), a vector with handles of found blocks, or an empty matrix if no blocks were found. If you are interested in TargetLink blocks only, use the block type as the search criteria. **Inport** returns unenhanced inports, and **InPort** or preferably **TL_Inport** returns the TargetLink port blocks.

Example:

```
tl_get_blocks('pipt1', 'TL_Inport')
```

```
ans =  
15.0032  
16.0032
```

- **tl_get_sfobjects**

As of TargetLink 3.2, you cannot call the **tl_get_sfobjects** command with the **Stateflow chart** parameter. Stateflow charts are now regarded as supported Simulink blocks. If you want to search for Stateflow charts, use the **tl_get_blocks** command instead.

```
[y,x] = tl_get_blocks('sf_demo', 'Statechart')
y =
  14.0011
x =
  'Stateflow'
```

- **tl_pref**

As of TargetLink 3.2, you cannot call the **tl_pref** command with the **list** parameter. If you want to list all preferences, use the **get** parameter instead.

```
tl_pref('get')

ans =

  CodeCovProgressBar: 'on'
  SyncSLScaling: 'off'
  SyncOutputScalingData: 'on'
  SyncSignalScalingData: 'on'
  SyncSaturationFlags: 'on'
  SyncConstrainedLimits: 'on'
  SyncParameterScalingData: 'on'
  SyncSFObjectScalingData: 'on'
  SyncSFObjectCompiledScalingData: 'on'
  SyncRTWData: 'on'
  ProjectFile: 'default.dd'
  ProjectFileAutosave: 'off'
  Editor: 'MATLAB Editor'
  BlockLibMode: 1
  DialogProvider: 'TargetLink'
```

- **tl_generate_code**

The **GenerateAll** property is obsolete. For backward compatibility it is still supported but you should use **IncludeSubItems** instead.

```
tl_generate_code('IncludeSubItems', 'on')
```

- **tl_generate_swc_model**

The **AutosarVersion** and **ModelClientServerPorts** properties are obsolete. The **AutosarVersion** property is obsolete because the version is recognized by the AUTOSAR import automatically. The **ModelClientServerPorts** property is obsolete because no SWC port blocks are added to the model for the client and the server ports.

AUTOSAR-Related Migration Aspects

AUTOSAR-Related Migration Aspects

Changed runnable signature

Due to the AUTOSAR compiler abstraction, runnable code generated with this version of TargetLink can differ from the runnable code of prior TargetLink versions.

TargetLink now supports the FUNC, P2CONST, and P2VAR macros for functions, variables, and pointers.

Runnable code of prior TargetLink versions

```
void Run(sint16 ScalarIn,
         sint16 ArrayIn[10],
         StructType* StructIn,
         sint16* ScalarOut,
         sint16 ArrayOut[10],
         StructType* StructOut)
```

Possible runnable code of this TargetLink version

```
FUNC(void, RTE_APPL_CODE) Run(sint16 ScalarIn,
                               P2CONST(sint16, AUTOMATIC, RTE_APPL_DATA) ArrayIn,
                               P2CONST(StructType, AUTOMATIC, RTE_APPL_DATA) StructIn,   P2VAR(sint16,
AUTOMATIC, RTE_APPL_DATA) ScalarOut,
                               P2VAR(sint16, AUTOMATIC, RTE_APPL_DATA) ArrayOut,
                               P2VAR(StructType, AUTOMATIC, RTE_APPL_DATA) StructOut)
```

The `Rte_Type.h` header file contains the compiler abstraction macro definitions. It is included in the software component header file (`<SWC/Runnable>.h`) if you generate code supporting compiler abstraction. Otherwise the `Rte_Type.h` header file is included in the software component C-file (`<SWC/Runnable>.c`).

Suggested migration TargetLink generates runnable code including the AUTOSAR compiler abstraction macros only if the `FunctionClass` of a runnable is unset or `AUTOSAR/RUNNABLE`.

If you want to generate unchanged runnable code for TargetLink models of versions prior to 3.2, you have to specify a runnable `FunctionClass` different from `AUTOSAR/RUNNABLE` such as `GLOBAL_FCN`. However, the default `FunctionClass` for runnables of TargetLink versions prior to 3.2 is `GLOBAL_FCN`.

Support of MATLAB versions after 2009b

Previously unsupported MATLAB versions, i.e., MATLAB versions after 2009b might result in an unrecoverable MATLAB crash if you want to simulate models that have been built with TargetLink versions prior to 3.2.

ReceiverComSpec and SenderComSpec blocks Since version 3.1 the TargetLink AUTOSAR Block Library contains the ReceiverComSpec and SenderComSpec blocks. S-functions for models with these blocks cannot be run with new MATLAB versions unless built with TargetLink 3.2.

Suggested migration Make a new build if you want to work with new MATLAB versions such as 2010b and unchanged TargetLink 3.1 models.

Improved exchange of data types with an architecture modeling tool

Changed Data Dictionary Master template for AUTOSAR

For improved exchange of software components with an architecture modeling tool the following changes have been made to the `dsdd_master_autosar.dd` template:

- The `TLDataTypes` and the `DataTypes` typedef groups have been added.
- The optimization options of the AUTOSAR variable classes have been changed.
- The `CodeGenerationBasis` property of the `AUTOSAR/Rte_Type` module object has been changed to `ModelAndDDBased`.

Note

Do not use or import packages and subpackages named `DataTypes` such as `/DataTypes` because the `/AUTOSAR/DataTypes` package is predefined by AUTOSAR.

During import the Data Dictionary would import elements of the standard AUTOSAR data types (`/AUTOSAR/DataTypes`) and the imported package or subpackage to the same DD group object named `/Pool/Typedefs/DataTypes`.

Suggested migration To use the improved exchange of software components with an architecture modeling tool with models prior to TargetLink 3.2, you have to perform the following steps:

1. Type `dsddman` in MATLAB's Command Window to open the DD Manager.
2. Open the Data Dictionary of your model in the DD Manager.
3. From the menu of the DD Manager, select View – Show DD Merge Explorer.

The DD Manager opens a file selection dialog.

4. In the file selection dialog, select the `dsdd_master_autosar.dd` file that is located in the `./Dsdd/Config` folder of your installation.

The DD Manager opens the template file in the DD Merge Explorer pane for you to copy specific items to the Data Dictionary of your model.

5. In the DD Merge Explorer, right-click the `Pool/Typedefs` node to open its context menu.
6. From the context menu, select `Copy left (merge-overwrite)`.
The DD Manager merges the `typedefs` node to your Data Dictionary.
7. In the DD Merge Explorer, right-click the `/Pool/VariableClasses/AUTOSAR` node to open its context menu.
8. From the context menu, select `Copy left (merge-overwrite)`.
The DD Manager merges the variable classes to your Data Dictionary.

9. In the DD Merge Explorer, right-click the `Modules/TLPdefinedModules/AUTOSAR/Rte_Type` node to open its context menu.
10. From the context menu, select `Copy left (merge-overwrite)`.
The DD Manager merges the module to your Data Dictionary.

Code Generator Options

Migration Aspects Regarding Code Generator Options

Obsolete Code Generator options

The following Code Generator options are no longer used in TargetLink 3.2:

- `InvalidateCodeOnError`
- `TreatSpecificErrorsAsWarnings`
- `OmitZeroInitializationsInStartupFunction`

If you generate code while the obsolete options are still in the model, a warning occurs.

Changed Code Generator options

The Code Generator options listed below were changed. Except for the `RequirementInfoAsCodeComment` option, whose default value was changed from `on` to `off`, all other changes are additional texts referring to Embedded MATLAB support.

Because of the changed `RequirementInfoAsCodeComment` option, comments containing requirement information are no longer included in the generated code automatically. If you want to have requirement information as comments in the generated code, set the `RequirementInfoAsCodeComment` to `on`.

Description	Explanation	Default Value	Min	Max
ConsiderStateflowAuxiliariesForVariableSharing				
For variable sharing, try to include internal auxiliary variables created for use in Stateflow charts and Embedded MATLAB function blocks.	Extend the pool of variables eligible for variable sharing by internal auxiliary variables created for use in Stateflow Charts and Embedded MATLAB function blocks.	on	-	-

Description	Explanation	Default Value	Min	Max
HandleUnscaledStateflowExpressionsWithTIType				
If set, TargetLink applies its code generation rules to Stateflow and EML expressions that contain variables with different TargetLink and Stateflow/EML data types.	Variables and macros used in Stateflow and Embedded MATLAB functions have a Stateflow/EML data type and a TargetLink data type. If the Stateflow/EML type of a variable/macro is different from its TargetLink type and this variable is used in a Stateflow/EML expression, the usual TargetLink code generation rules regarding scalings and types will be applied for this expression if the option is set to 'on'. Otherwise if no <i>LSB</i> != 1.0 and <i>Offsets</i> != 0.0 are involved in the expression, the expression will be copied to the generated code unchanged.	on	-	-
RequirementInfoAsCodeComment				
Show requirements associated with code fragments as comments in the generated code.	Requirements associated with TargetLink blocks and Stateflow objects that were associated with code fragments in the generated code are emitted as comments in the generated code, if the option is set to on.	off	-	-

Description	Explanation	Default Value	Min	Max
TreatAllStateflowFunctionsAsWeakAtomic				
Assume weak atomic semantics (i.e. allow interleaving code if it does not interfere with the generated code's behavior) for all Stateflow charts and Embedded MATLAB function blocks whose functions are inlined.	<p>A Stateflow chart and an Embedded MATLAB function block are treated as if implemented by an (inlined) function, and weak atomic determines whether TargetLink performs optimizations such that nontrivial computations from outside the chart/EML block are introduced into the code area belonging to this function, i.e. there is no interleaving code between different atomic units. Example:</p> <pre>b = a + 5; c = a; /* Start execution of chart */ d = f(c); e = 7 * b; /* End execution of chart */ can always be optimized to b = a + 5; /* Start execution of chart */ d = f(a); e = 7 * b; /* End execution of chart */ but will be optimized further to /* Start execution of chart */ d = f(c); e = 7 * (a + 5); /* End execution of chart */ only if the chart/EML block is "weak atomic". This option makes all Stateflow charts and EML function blocks "weak atomic". If the option is off, it can be overridden by switching on the option "AllowInterleavingCodeForAllSubsystems".</pre>	on	-	-

New Code Generator options For information on new Code Generator options, refer to [Code Generator Options](#) on page 244.

Related topics

References

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

Other

Various Migration Aspects

Sqrt block instead of function

As of TargetLink 3.2 and MATLAB R2010a, the `sqrt` function no longer appears in the Math block. If you open a model built with TargetLink < 3.1 in combination with MATLAB R2010a for the first time and an update is performed by the system, a Sqrt block is inserted for each Math block that computes the square root function. For details, refer to [Sqrt Block \(TargetLink Model Element Reference\)](#).

Loop variables in Stateflow

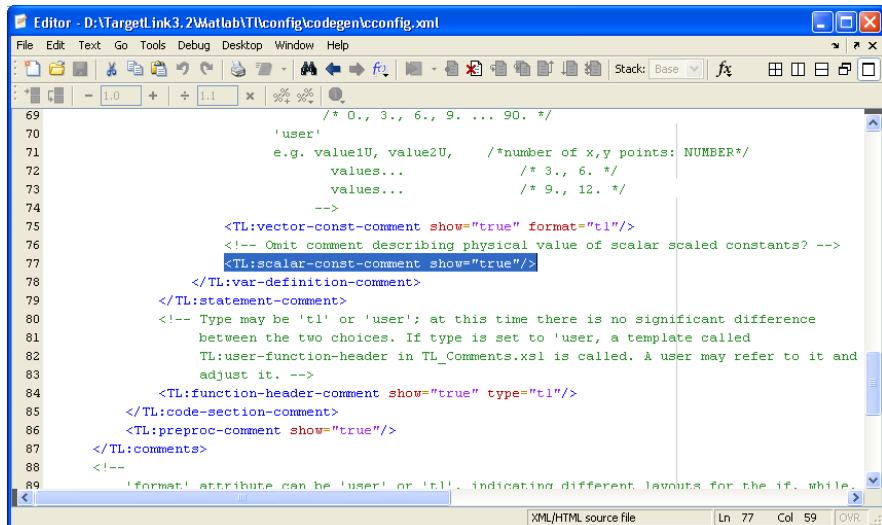
If you model a loop variable in Stateflow (always do .. while, not based on the mechanisms of *LoopsForVectorSignals*), other variable names are now used for the loop variable (TargetLink <= 3.1: `idx`, `dx`; TargetLink 3.2: `idx1`, `idx2`).

Optimization improvements regarding constants

TargetLink 3.2 provides various optimization improvements and simplifications regarding constants. These improvements influence the generated code, for example, with regard to constant folding ($a + b$; if both operands are constant, they are replaced by their result), elimination of cast operations and utilization of constant initialization values. Depending on the type of optimization, cast operations are eliminated, additional comments are inserted in the generated code or fewer computations are performed on constants.

Comments for constants

Comments for constants that describe physical values can now be switched off via the `cconfig.xml` file. To do so, set the default shown below to false:



```

Editor - D:\TargetLink3.2\MatlabATL\config\codegen\cconfig.xml
File Edit Text Go Tools Debug Desktop Window Help
File Edit Text Go Tools Debug Desktop Window Help
1.0 1.1 %& %&
69      /* 0., 3., 6., 9. ... 90. */
70      'user'
71      e.g. value1U, value2U,    /*number of x,y points: NUMBER*/
72      values...                /* 3., 6. */
73      values...                /* 9., 12. */
74      -->
75      <TL:vector-const-comment show="true" format="t1"/>
76      <!-- Omit comment describing physical value of scalar scaled constants? -->
77      <TL:scalar-const-comment show="true"/>
78      </TL:var-definition-comment>
79      </TL:statement-comment>
80      <!-- Type may be 't1' or 'user'; at this time there is no significant difference
81      between the two choices. If type is set to 'user', a template called
82      TL:user-function-header in TL_Comments.xsl is called. A user may refer to it and
83      adjust it. -->
84      <TL:function-header-comment show="true" type="t1"/>
85      </TL:code-section-comment>
86      <TL:preproc-comment show="true"/>
87      </TL:comments>
88      <!--
89      'format' attribute can be 'user' or 't1', indicating different layouts for the if, while,
      -->

```

Once you have set `<TL:scalar-const-comment show="false"/>`, constant comments no longer appear in the generated code.

Parenthesizing of variables used as operands (address and dereference operators)

Simple variables used as operands of address operators and dereference operators are no longer parenthesized.

TargetLink 3.2	TargetLink < 3.2
<code>pVar = &var;</code>	<code>pVar = &(var);</code>
<code>var = *pVar;</code>	<code>var = *(pVar);</code>
For struct components or array elements parentheses are still used.	
<code>pVar = &(Vector[1]);</code>	<code>pVar = &(Vector[1]);</code>
<code>var = *(Struct.pPointer);</code>	<code>var = *(Struct.pPointer);</code>

Discontinued boards

There is a new evaluation board (Freescale MPC5604BEVB) and evaluation boards that are no longer supported and/or distributed.

No longer supported, no longer distributed The following boards are no longer supported by TargetLink and no longer distributed by dSPACE:

- NEC Drivelt Evaluation Board
- Renesas Evaluation Board MSA2114
- FS Forth-Systeme STart276 Development Board
- Texas Instruments TMS470R1x Evaluation Board

No longer supported, still distributed The following board is no longer supported by TargetLink but still distributed by dSPACE:

- Infineon TriBoard TC1775 Evaluation Board

Tip

To use the unsupported evaluation boards with TargetLink 3.1, please contact dSPACE.

Still supported, no longer distributed The following boards are still supported by TargetLink but no longer distributed by dSPACE:

- Renesas EVB7055F Evaluation Board
- Renesas EVB7058 Evaluation Board
- Axiom CMD-0565 Evaluation Board
- MCT HCS12 T-Board (DP256) and Freescale M68EVB912DP256 Evaluation Boards

Discontinued tool

Previous versions of TargetLink and the dSPACE Data Dictionary provided you with the `t1_export2dd` tool that let you import TargetLink model data into the dSPACE Data Dictionary. As of TargetLink 3.2, this tool is not part of the TargetLink Base Suite anymore.

If you want to use **tl_export2dd**, you can download it via the *TargetLink Product Support Center* (www.dspace.com/goto?TargetLinkProductSupportCenter).

Rate Limiter block

As of TargetLink 3.2, the code generated for the Rate Limiter block reflects the initial value specified at the Simulink block (under the block mask) if the block is run with a discrete sample time. There are two scenarios where the code differs from previous TargetLink versions:

Migrating from TargetLink 2.x In TargetLink 2.x, the block library contained the Rate Limiter block with the sample time set to **continuous**. The initial value could not be specified.

After the upgrade to TargetLink 3.2, the sample time is set to **inherited** and the initial value is set to **0**. Thus, the Rate Limiter block's simulation behavior is different from TargetLink 2.x in all simulation modes.

Migrating from TargetLink 3.x As of TargetLink 3.2, the block library contains the Rate Limiter block with the sample time set to **inherited** and the initial value is set to **0**.

As the code generated for the Rate Limiter block now reflects the initial value if the block is run with a discrete sample time, the Rate Limiter block's simulation behavior is different from TargetLink 3.x in SIL and PIL simulation modes.

Cast operations with unary minus and bitwise operations

As of TargetLink 3.2, additional cast operations are applied to unary minus and bitwise operations.

Unary minus The operand of a unary minus operation is always cast to the result type.

Additionally, if the operand is unsigned, it is cast to a signed type before the minus operator is applied. This is done to comply with MISRA rule 12.9, which does not allow the unary minus operator to be applied to an expression whose underlying type is unsigned.

If the width of the result type is larger than the width of the operand, the operand is cast to the output width before the minus operator is applied.

TargetLink < 3.2

```
b = -a;
```

TargetLink 3.2

```
b = (Int16)-a;
```

Bitwise operations A bitwise operation is always cast to the result type.

If the original width is larger than the width of the operand, the operand is cast to the original width before the `~` operator is applied.

With binary bitwise operations, the operands are cast to the result type if both are less wide than the output and are signed differently.

TargetLink < 3.2

```
d = ~c&4;
```

TargetLink 3.2

```
d = (UInt16)((UInt16)(~c)&((UInt16)4));
```

Comparisons with Float32 constants

To ensure that comparisons (<, <=, >, >=) are calculated correctly, Float32 constants that cannot be represented in Float32 are rounded to the next higher or lower number that can be represented in Float32. Whether the constant is rounded up or down depends on the operator and the position of the constant (left- or right-hand side for the comparison expression).

TargetLink < 3.2

```
F32Var > 2097151.95F
```

TargetLink 3.2

```
F32Var > 2097151.875F
```

Abs operation

As of TargetLink 3.2, the code generated for Abs operations is different, if all of the following conditions are fulfilled:

- There are only integer operands in the Abs operation.
- The Abs operation is saturated.
- The Abs argument has a signed data type.
- The Abs argument is scaled with an offset = 0.

Sequence of variable definitions and declarations

The sequence of variable definitions and declarations may change if the model that the code is generated for references variables that are specified in the Data Dictionary and assigned to a DD module object with the property **CodeGenerationBasis** set to **ModelAndDDBased**.

Sequence generated with TargetLink 3.1

```
A2L_None Int16 Axis[3] =
{
  /*[0..2]*/ 1, 2, 3
  /* 1., 2., 3. */
};
A2L_None UInt16 NumAxisPoints = 3 /* LSB: 2^0 OFF: 0 MIN/MAX: 0 ...
65535 */;
```

Sequence generated with TargetLink 3.2

```
A2L_None UInt16 NumAxisPoints = 3 /* LSB: 2^0 OFF: 0 MIN/MAX: 0 ...
65535 */;
A2L_None Int16 Axis[3] =
{
  /*[0..2]*/ 1, 2, 3
  /* 1., 2., 3. */
};
```

Migrating from TargetLink 3.0 to 3.1

Where to go from here

Information in this section

API Functions.....	387
AUTOSAR-Related Migration Aspects.....	388
Code Generator Options.....	391
General Migration Aspects.....	392
Other.....	395

API Functions

Where to go from here

Information in this section

Modified DD MATLAB API Functions.....	387
Changes in TargetLink API Functions.....	388

Modified DD MATLAB API Functions

Changes to the DD MATLAB API

There is a new feature for code generation from the dSPACE Data Dictionary providing new Module objects in the data model of the dSPACE Data Dictionary.

The upgrade of the Data Dictionary replaces all the ModuleName properties in the Data Dictionary with references to Module objects.

You have to remember this when using the DD MATLAB API, i. e., in scripts.

Changed DD MATLAB API functions: SetWidth / GetWidth

The Data Dictionary MATLAB API provides the **GetWidth** and **SetWidth** commands to read and set the **Width** property of DD variable objects. The behavior of these commands has been changed.

If a DD variable object references a typedef object, the variable object's **Width** property is set via the typedef object's **Width** property. In this case **GetWidth** now returns the typedef object's width, and **SetWidth** returns an error because

the variable object's width can be specified only either via the local **Width** property or via the **Width** property of the referenced Typedef object.

Changes in TargetLink API Functions

Changed API function In TargetLink 3.1, the following TargetLink API function name has changed.

TargetLink 3.0	TargetLink 3.1	Comment
<code>get_sfobjects</code>	<code>tl_get_sfobjects</code>	The command <code>get_sfobjects</code> no longer exists in TargetLink 3.1 and has been replaced by <code>tl_get_sfobjects</code> .

AUTOSAR-Related Migration Aspects

AUTOSAR-Related Migration Aspects

Migration of Data Dictionaries prior to Data Dictionary 2.0

Automatic Data Dictionary upgrade When you access Data Dictionaries prior to Data Dictionary 2.0 they are upgraded automatically.

Note

AUTOSAR DD upgrade is not supported for Data Dictionaries prior to Data Dictionary 1.5. If you want to upgrade an older Data Dictionary, contact dSPACE Support (www.dspace.com/go/supportrequest).

The following steps are performed during DD upgrade:

- Synchronization of the Data Dictionary with the revised AUTOSAR DD template. The DD upgrade adapts DD objects such as type definitions and variables. The DD upgrade also moves the `/Pool/Interfaces` DD node to `/Pool/Autosar/Interfaces`.

Note

You have to customize self-written scripts as required.

- Initial values that are specified in communication specifications have been replaced by variables. The upgrade creates variable objects that are referenced via the `InitRef` property in the communication specifications.

- InitFunction and TermFunction DD objects have been replaced by runnables. Runnables for initialization and termination are specified via their Kind property.

Migrating Data Dictionaries If you are also working with an AUTOSAR-compliant architecture tool, you should reimport the AUTOSAR file from the architecture tool to the Data Dictionary after upgrading. The Data Dictionary 2.0 supports the following items for improved exchange of AUTOSAR files.

- Constant specifications
- Array types
- Access points and read/write accesses

Migration of Simulink/TargetLink models prior to TargetLink 3.1

Automatic model upgrade When you access Simulink/TargetLink models prior to TargetLink 3.1, they are upgraded automatically.

Note

AUTOSAR model upgrade is not supported for Simulink/TargetLink models prior to TargetLink 2.3. If you want to upgrade an older model, contact dSPACE Support (www.dspace.com/go/supportrequest).

The following steps are performed during model upgrade:

- Replacement of Runnable blocks by Function blocks. The AUTOSAR settings for code generation are now on the AUTOSAR page of the Function block.
- Replacement of Runnable Inport blocks by TargetLink InPort/Bus Inport blocks as required. The AUTOSAR settings for incoming signals are now on the AUTOSAR page of the InPort/Bus Inport block. If the Status port of a Runnable Inport block is enabled, a ReceiverComSpec block is added.
- Replacement of Runnable Outport blocks by TargetLink OutPort/Bus Outport blocks as required. The AUTOSAR settings for outgoing signals are now on the AUTOSAR page of the OutPort/Bus Outport block. If the Status port of a Runnable Outport block is enabled, a SenderComSpec block with enabled Status port is additionally added.
- Replacement of ClientPort blocks by atomic subsystems with a Function block that is configured for implementing an operation call.

Note

The model upgrade does not support replacing ClientPort blocks that are connected with Bus Inport/Bus Outport blocks for modeling operations with several IN and/or OUT arguments. You have to migrate those ClientPort blocks by hand. Refer to [Migrating ClientPort Blocks](#) on page 395.

Migrating SWC SenderPort and SWC ReceiverPort blocks If you migrate a model where TargetLink blocks other than Mux/Demux, Bus Creator/Bus Selector, or Selector are used together with SWC SenderPort/SWC ReceiverPort

blocks outside runnables, code generation might fail. Perform the following migration steps:

1. Place a Simulink Inport block before/after the SWC ReceiverPort/SWC SenderPort block.
2. Transform it into a TargetLink InPort block via the TargetLink - Enhance block context menu command.
3. If the combined signal represents a struct, transform the TargetLink InPort by opening the block dialog and clicking **Click here to transform the port block to a busport block**.

Migrating buses for struct-based data elements If you migrate a model with struct-based data elements, model initialization might fail due to non-matching signals. For migration, you have to name the bus signals of a bus that represents a struct-based data element according to the components of the data element.

In previous TargetLink versions, the components of a structured data element were simply mapped onto the signals within a Simulink bus based on their order of appearance in the struct and the Simulink bus. With TargetLink 3.1, component names and signal names in the Simulink bus must match.

Changed modeling practices

The following modeling practices have been changed:

- The Runnable block has been replaced by the AUTOSAR dialog page of the Function block. You have to select the **Implement as AUTOSAR runnable** checkbox on the dialog page to implement a subsystem as an AUTOSAR runnable. The runnable subsystem now has to be triggered in the same way as in Simulink, i.e., by triggering or enabling the runnable subsystem as required. RTE event properties have to be edited in the Data Dictionary. You have to adapt user-written scripts that have run on Runnable blocks.
- The Runnable Inport/Runnable Outport blocks have been replaced by AUTOSAR dialog pages of the TargetLink InPort/Outport or Bus Inport/Bus Outport blocks. You have to specify AUTOSAR settings for runnable subsystem interfaces via the **Use AUTOSAR communication** checkbox on the dialog pages. You have to adapt user-written scripts that have used Runnable Inport/Runnable Ouport blocks for reading runnable interface data.
- The ClientPort block has been removed from the TargetLink AUTOSAR Block Library. You can now model operation calls using a subsystem with a Function block. Select the **Implement the subsystem as operation call** checkbox on the AUTOSAR dialog page for this purpose. You can also implement the same subsystem for simulation in SIL/PIL mode or even implement the same subsystem as the runnable for the operation. You have to adapt user-written scripts that have run on ClientPort blocks.
- The mapping of struct type data element components to bus signal elements was performed by *sequence* with TargetLink versions prior to TargetLink 3.1. Now the mapping is performed by *name*. You have to name the bus signals according to the components of a data element.

Discontinued support for AUTOSAR Release 2.0

With Version 2.0, the Data Dictionary no longer supports the import/export of AUTOSAR files according to AUTOSAR Release 2.0.

Code Generator Options

Migration Aspects Regarding Code Generator Options

Obsolete options for the Code Generator

The following Code Generator options are no longer used in TargetLink 3.1:

EnableMultirate The EnableMultirate option is obsolete. Use the CodeGenerationMode = RTOS setting instead.

ScopeReductionPreserveLinkerSectionEntriesUserClass The ScopeReductionPreserveLinkerSectionEntriesUserClass option is no longer used.

VariantFileDefaultID The VariantFileDefaultID option is obsolete. As of TargetLink 3.1 you can set the ModuleRef property directly in the DataVariant object specifying a data variant.

If your model contains obsolete Code Generator options that are set to values other than the default, the model upgrade does not delete these options automatically. If you generate code while the obsolete options are still stored at the model, a warning occurs.

Changed options for the Code Generator

The interpretation of the numerical values (0 to 4) of the Code Generator option Logging was changed. If you use the `pre_codegen_hook` function to generate clean code or to configure global logging options, this will now create wrong results. For details, refer to [tl_pre_codegen_hook](#) ( [TargetLink File Reference](#)).

New Code Generator options

For further information on new Code Generator options, refer to [Code Generator Options](#) on page 272.

General Migration Aspects

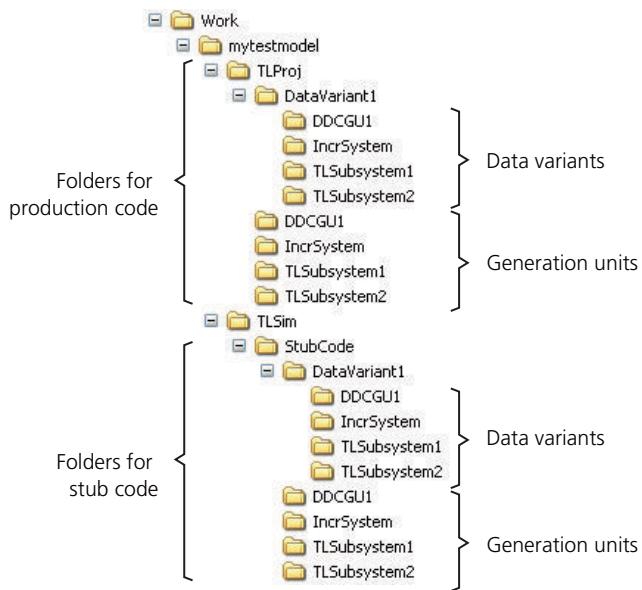
General Migration Aspects

Migration aspects There are various aspects that you have to note when migrating to TargetLink 3.1.

New file deposition structure TargetLink 3.1 allows you to generate code for code generation units (CGUs), which can be TargetLink subsystems, subsystems configured for incremental code generation, referenced models, or CodeGenerationUnit objects in the Data Dictionary. In contrast to older versions, TargetLink 3.1 does not place the generated production code directly in the working directory, but in a specific file deposition structure. .

For the production code, TargetLink generates a `/TLProj` folder which contains one subfolder for each CGU in your model. The names of the subfolders are the names of the CGUs. For the stub code (code generated for model components using objects from foreign modules i.e., variables are declared but not defined), TargetLink generates a `/StubCode` folder in the `/TLSim` folder. Like the `/TLProj` folder, the `/StubCode` folder contains one subfolder for each CGU. For data variants, specific folders are generated with the same structure as the `/TLProj` and `/StubCode` folders.

When an application is built, the files to be built are copied to a build folder under the `/TLSim` folder. If there are still generated files in the working directory, a warning occurs. If there are files to be compiled in the working directory when an application is built, an error message is issued and the build process is stopped. Generated documentation is not stored under `/doc` but under `/TLProj/doc`. TargetLinks generates ASAM MCD 2MC files in the `/TLProj` folder.



The `ModuleOwnership` object in the DataDictionary specifies which code generation unit (CGU) generates the production code version of a module. For further information, refer to [Mapping Code Generation Units to Code \(Module Ownership\)](#) ([TargetLink Customization and Optimization Guide](#)) and to [File Deposition Structure of the Files Generated by the TargetLink Code Generator](#) contained in the [TargetLink File Reference](#) shipped prior to TargetLink 4.3 (Release 2017-B).

Module objects in the pool area

As of TargetLink 3.1, module objects are created in the Pool area of the Data Dictionary. They represent modules created and/or emitted in the file system during production code generation.

The upgrade of TargetLink models and DD project files creates module objects for TargetLink internal files such as `t1_basetypes.h` or `t1Defines.h` in `/Pool/Modules/TLPRedefinedModules/` and for each module name specified at variables, types, variable classes, etc. Hence TargetLink allows you to modify internal modules via the module objects. The module templates are applied, and all modules originating from the module templates are created. In TargetLink 3.1, module templates must be used only to modify sets of modules. Templates specifying one specific module must be replaced by a module object. Remember that all module name properties at various DD objects have been replaced by references to a specific module object. Keep in mind that if a DD module object already exists, no module template is applied to it.

During code generation, module templates are not applied to modules that are specified via module objects in the dSPACE Data Dictionary.

ModuleOwnership

The `ModuleOwnership` object lets you define the owner of a module precisely. It specifies which code generation unit (CGU) generates the production code version of the module.

If you used module templates to implement the module ownership in a model in TargetLink < 3.1, you have to create corresponding module ownership objects and use them instead. For further information, refer to [Basics on Modules and Module Ownership](#) ( [TargetLink Customization and Optimization Guide](#)).

Data types not assigned to a specific module

In contrast to TargetLink < 3.1, all the data types that are not assigned to a specific module are generated into the user-defined data type header file (UDT.h).

Prevent emitting files

With TargetLink < 3.1, you could specify subsystems under the <Application> node of the Data Dictionary and assign a module to them to prevent them from being emitted. As of TargetLink 3.1, this specification is obsolete and will be ignored. If you used this mechanism, you have to create module objects and set the EmitFile property or specify an ownership with ModuleOwnership objects instead.

The upgrade does not provide module objects based on this specification, and deletes the Subsystem and <Application> areas from the Data Dictionary.

To prevent such subsystems from being emitted, you can explicitly create module objects in the Data Dictionary and prevent them from being emitted by setting the EmitFile property to off. This applies especially to `t1_basetypes.h`, which exists as a DD module object and which is not emitted.

Data type with buses in referenced models

If the input of a referenced model is a bus, the data type of the bus must be explicitly assigned to a module. Otherwise, an error message tells you that the data type is defined twice. With TargetLink < 3.1, this was the case for models that are connected to the same bus. If you want to distribute the data types in the previous way, you can set the compatibility option StrictTypedefPlacement to false.

Changed behavior of Bus Import and Outport blocks

TargetLink no longer determines changes in the bus hierarchy automatically when the block dialog is opened. As of TargetLink 3.1 you have to click the Rescan bus hierarchy button in the Bus Import or Bus Outport dialogs to update the bus hierarchy. The model must be initializable for this. The update of the bus hierarchy may take some time.

Other

Where to go from here	Information in this section
	Migrating ClientPort Blocks..... 395 Various Migration Aspects..... 396

Migrating ClientPort Blocks

Objective To migrate TargetLink models that contain ClientPort blocks that are connected to bus signals.

Note

The model upgrade replaces ClientPort blocks that are connected to non-bus signals without the need for further migration steps. You can generate code for those models after the model upgrade has been performed.

If a ClientPort block has been connected to bus signals before model upgrade, the ClientPort block is replaced by a subsystem that implements an operation call. To fully migrate the model, you have to replace the bus signal and connect its contained signals via TargetLink InPort/OutPort blocks.

Preconditions TargetLink must have upgraded the model.

- Method**
- To migrate ClientPort blocks**
- 1 In the subsystem that replaces the ClientPort block, add TargetLink InPort/OutPort blocks to match the individual signals of the incoming/outgoing bus signals.
 - 2 Remove Demux/Bus Creator blocks that handle the bus signals in the subsystem.
 - 3 Properly connect the signals in the subsystem and specify the AUTOSAR settings for client-server communication in the InPort/OutPort AUTOSAR dialog pages. For information on modeling client-server communication, refer to [Modeling Client-Server Communication](#) ([TargetLink Classic AUTOSAR Modeling Guide](#)).
 - 4 Properly connect the subsystem in your model, i.e., replace the ingoing/outcoming bus signals by individual signals using Demux/Bus Creator blocks as required.

Result	You have migrated TargetLink models with ClientPort blocks that had been connected to bus signals and you are ready to generate code.
---------------	---

Various Migration Aspects

No automatic replacement of bit operation blocks	The model upgrade does not replace Bit operations blocks from the TargetLink Sample Blocks (t1samples/Bit Operations) with the new bit operation blocks introduced with TargetLink 3.1. If you want to use the new blocks, you have to replace the old ones manually.
Obsolete EmitCode property	As of TargetLink 3.1, the <code>EmitCode = {on,off}</code> property of FileSpecification objects (child objects to ModuleTemplate objects) no longer exists. It is replaced by the <code>EmitFile={on,off,auto}</code> property. <code>EmitFile=Auto</code> is equivalent to <code>EmitCode=on</code> .
Fewer parentheses for unary operations	<p>Fewer parentheses are used for unary operations such as incrementing a loop variable.</p> <ul style="list-style-type: none">▪ In TargetLink < 3.1, unary operations are generated as follows: <code>for (i = 0; i < 10; (i)++)</code>▪ In TargetLink 3.1, unary operations are generated as follows: <code>for (i = 0; i < 10; i++)</code>
Modified code pattern for loops for vector signals	<p>The code pattern representing For loops for vector signals has been changed. In TargetLink 2.3 and 3.0, For loops were generated with a <code><=</code> relation in the condition, as shown in the following example:</p> <pre>for (i = 0; i <= 9; i++) {...}</pre> <p>In TargetLink 3.1, For loops are generated with a <code><</code> relation, as shown in the following example:</p> <pre>for (i = 0; i < 10; i++) {...}</pre>
Modified code pattern for relational operations	<p>In TargetLink 3.1, the scaling in which a relational operation is performed can change. That is, if an operand of a relational operation was part of an scale adaption operation in a previous version of TargetLink, now the other operand of the relational operation might be part of the invers scale adaption operation, as shown in the following example:</p> <pre>Var1 << 1 > Var2</pre>

might become

```
Var1 > Var2 >> 1
```

No empty return value during DD upgrade

The upgrade of a DD project file generates warnings and information that are displayed in the Message Browser. The return value of the upgrade function is the message number of the latest message. This also applies to the `dsdd('Open',<DDFile>)` command.

Scripts that check for an empty return value will indicate errors even if no errors occurred. After the upgrade, the messages are not displayed.

Input signal dependent variable structure

If a block supporting signal properties inheritance has a bus signal at its input, the variables of the block are created according to the structure of the bus signal.

This can cause the following kinds of code changes in comparison with prior TargetLink versions:

- A single vector variable is replaced by multiple variables.
- A single loop is replaced by multiple single statements and / or multiple loops.
- Multirate code generation: A single message is replaced by multiple messages.
- Multirate code generation: A message is no longer protected (because the criterion 'atomic access' is fulfilled more often).

Merging of variables

In previous TargetLink versions, two or more variables with the same name but different variable classes were optimized in one variable provided that the MERGEABLE optimization property was set in all variable classes, even if the optimization properties were not compatible with regard to MOVABLE, ERASABLE, SCOPE_REDUCIBLE. As of TargetLink 3.1, the relevant properties must match. The recommended procedure when using MERGEABLE is to create exactly one variable in the dSPACE Data Dictionary, which is then referenced in all the appropriate places.

Subsystem-to-subsystem ID assignment without model name

In the comment in the file header of the C code file, among others, the assignment of the subsystem ID to the subsystem that code is generated for (TargetLink subsystem or subsystem configured for incremental code generation) is indicated. As of TargetLink 3.1, the name of the model is no longer output before the name of the subsystem.

Example:

- TargetLink < 3.1

```
Sa1      MyMode1/MyTLSubsystem
```

- TargetLink 3.1

```
Sa1      MyTLSubsystem
```

```

/*
*** Simulink model      :pigt1
*** TargetLink subsystem :pigt1/picontroller
*** Codefile             :picontroller.c
***
*** Generated by TargetLink, the dSPACE production quality Code Generator
*** Generation date: 2009
***
*** CODE GENERATOR OPTIONS:
*** Compiler              :<unknown>
*** Target                :Generic
*** ANSI-C compatible code :yes
*** Optimization level    :2
*** Constant style        :decimal
*** Clean code option     :disabled
*** Logging mode          :According to block-specific data
*** |
*** Add model checksum    :disabled
***
*** SUBSYS               CORRESPONDING SIMULINK SUBSYSTEM
*** Sa1                  picontroller
***
*** SUBSYS               CORRESPONDING MODEL BLOCK (REFERENCED MODEL)
*** Copyright (c) 2009 dSPACE GmbH
*/

```

Default name of modules generated from referenced models

As of TargetLink 3.1, the default name of a module generated from a referenced model is not `TL_<ModelName>` but `<ModelName>`. This complies with the general rule that a module's default name is the name of the system (TargetLink subsystem or subsystem configured for incremental code generation) that code is generated for.

Changed table names

The default names for the axes of look-up tables (1-D) and (2-D) have changed as shown below. If you do not want to use the new default names, you can also specify your own names.

The following changes apply to Look-up table (1D):

Look-up table (1D)	TargetLink 3.0	TargetLink 3.1
X axis	<code>\$S_\${B}_x_table</code>	<code>\$S_\${B}_axis</code>
Table	<code>\$S_\${B}_z_table</code>	<code>\$S_\${B}_table</code>

The following changes apply to Look-up table (2D):

Look-up table (2D)	TargetLink 3.0	TargetLink 3.1
Row axis	<code>\$S_\${B}_x_table</code>	<code>\$S_\${B}_axis_1</code>
Column axis	<code>\$S_\${B}_y_table</code>	<code>\$S_\${B}_axis_2</code>
Table	<code>\$S_\${B}_z_table</code>	<code>\$S_\${B}_table</code>

Changes to user-defined header files

Up to TargetLink < 3.1, the name of the user-defined header file was built from the base name of the TargetLink root file and the extension `_udt`. As of TargetLink 3.1, the name of the user-defined header file is by default `udt_${I}.h`. This also necessitates a change in the include statements.

You can use the `UseRootFileNameAsUserDefinedTypesHeaderBasename` Code Generator option to achieve the behavior of the previous TargetLink versions.

Additionally, you can change the name of the user-defined header file via the `NameTemplate` property of the `/Pool/Modules/TLPPredefinedModules/UserDataTypes` module object.

Avoid merging of Bus and Mux signals

The merging of bus and muxed signals must always be avoided, as otherwise the overflow detection for Bus Import blocks does not work properly. If you split a signal to separate signal parts using Demux blocks and subsequently merge them in a bus signal via Bus Creator blocks, the used Simulink interface provides wrong data for the overflow detection and min/max logging. This results in overflow warnings not being properly assigned to the separate bus elements.

Name macro `$(FunctionName)` in `LookupFunctionTemplate` objects

With TargetLink < 3.1, the `$(FunctionName)` name macro could be used for the module name, function name and type name in DD `LookupFunctionTemplate` objects. With the introduction of the module objects in the dSPACE Data Dictionary, the `$(FunctionName)` name macro can no longer be used for the module name.

There are two alternatives:

- Set the `ModuleRef` property to `<n.a.>` and activate the `Share functions between subsystems` option. A separate file is then generated for each created LookUp function. The name of each file is identical to the default LookUp function name, which in turn is identical to the name that would have resulted from the `$(FunctionName)` name macro.
 - If you want to extend the default LookUp table function name with a prefix or postfix, you can specify the module name directly. You need a template and a module object for each LookUp table function.
-

Tool Selector block configured for model referencing

As of TargetLink 3.1, the functionality of the Model Referencing Control Center is integrated into the TargetLink Main Dialog. The Tool Selector block configured for the Model Referencing Control Center is therefore no longer needed in TargetLink models with references to other models and is automatically removed from the model by the upgrade.

Listing files used for building the simulation application

After code generation has finished, the MATLAB Command Window displays a link named `Click here`, which outputs a list of the generated files that were used for building the simulation application.

Changed code pattern for the Multiport-Switch block

In TargetLink 3.1, the code pattern for the Multiport-Switch block has changed. The last input of the block is not calculated in the conditionally executed (case) branch, but in the default branch.

No `ModuleRef` entry for variable object of scope `ref_param`

A variable object with its scope set to `ref_param` means call-by-reference semantics. The `ref_param` scope is the default in the variable classes `FCN_REF_ARG`, `OPT_FCN_REF_ARG` and `FCN_REF_PARAM`. In the resulting C code, such a variable becomes a formal parameter in a function declaration, as follows:

```
f(int16 x) { ... }
```

Consequently, the variable is bound to the module the function is generated into. The ModuleRef property of the variable object must therefore not be set. Usually, a function will not only be declared but also called in the generated code. In order to call the function, you need a variable to pass to the function, the actual parameter, as follows:

```
g()
{
...
int16 z = 1
f(&z)
...
}
```

The module into which the definition of the actual parameter will be generated can be specified by setting the ArgClass property of the variable class of the variable object which will become the formal parameter. The property value of the ArgClass is a variable class itself, and the ModuleRef property of the variable class object specifies the module that will contain the definition of the variable that becomes the actual parameter.

Discontinued boards

No longer supported, no longer distributed The following boards are no longer supported by TargetLink and no longer distributed by dSPACE:

- NEC Drivelt Evaluation Board
- Renesas Evaluation Board MSA2114
- FS Forth-Systeme STart276 Development Board
- Texas Instruments TMS470R1x Evaluation Board

No longer supported, still distributed The following board is no longer supported by TargetLink but still distributed by dSPACE:

- Infineon TriBoard TC1775 Evaluation Board

Tip

To use the unsupported evaluation boards with TargetLink 3.1, please contact dSPACE.

Still supported, no longer distributed The following boards are still supported by TargetLink but no longer distributed by dSPACE:

- Renesas EVB7055F Evaluation Board
- Renesas EVB7058 Evaluation Board
- Axiom CMD-0565 Evaluation Board
- MCT HCS12 T-Board (DP256) and Freescale M68EVB912DP256 Evaluation Boards

Discontinuations

Where to go from here

Information in this section

Discontinuations as of TargetLink 5.1.....	402
Discontinuations as of TargetLink 5.0.....	403
Discontinuations as of TargetLink 4.4.....	406
Discontinuations as of TargetLink 4.3.....	409
Discontinuations as of TargetLink 4.2.....	411
Discontinuations as of TargetLink 4.1.....	415
Discontinuations as of TargetLink 4.0.....	418
Discontinuations as of TargetLink 3.5.....	422
Discontinuations as of TargetLink 3.4.....	424
Discontinuations as of TargetLink 3.3.....	426
Discontinuations as of TargetLink 3.2.....	429
Discontinuations as of TargetLink 3.1.....	430

Discontinuations as of TargetLink 5.1

Where to go from here

Information in this section

Discontinued TargetLink Features.....	402
Obsolete Limitations.....	402

Discontinued TargetLink Features

DD ModuleTemplate

With TargetLink 5.1, DD ModuleTemplate objects have been removed. Existing DD ModuleTemplate objects are removed from the Data Dictionary without replacement.

The removal of DD ModuleTemplate objects was announced with TargetLink 5.0.

Demo models

The following demo models are discontinued with TargetLink 5.1:

- SF_USER_FUNCTION
- BLACKBOX

Obsolete Limitations

Array-of-struct

Signal injection is not supported for array-of-struct variables.

Signal tunneling is not supported for array-of-struct variables.

Compatibility of SICs with different platforms

SICs built for Linux 64-bit (Generic_x86_64_Linux/GCC) are not compatible with dSPACE platforms..

TargetLink does not inform at SIC generation time if the generated SIC is compatible with at least one of the dSPACE simulation platforms consuming SIC files.

Discontinuations as of TargetLink 5.0

Where to go from here

Information in this section

Discontinued TargetLink Features	403
Obsolete API Functions.....	404
Obsolete Limitations.....	405

Discontinued TargetLink Features

Code generation for special OSEK versions

The code generation for special OSEK versions, such as OsCan, is no longer supported with TargetLink 5.0 :

- The RTOS selection on the RTOS page of the [TargetLink Main Dialog Block](#) ([TargetLink Model Element Reference](#)) is limited to generic RTOS and generic OSEK.
- Customizing which real-time operating systems are displayed in any user interface via the TargetLink Preferences Editor is no longer possible.

Additionally, the following restrictions apply to the CounterAlarm block:

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

The removal of the support of special OSEK versions was announced with TargetLink 4.1.

Related documentation

- [Connecting Imports and Outports of the CounterAlarm Block](#) ([TargetLink Multirate Modeling Guide](#))

User state flags in Stateflow

With TargetLink 5.0, the support for the TargetLink-specific user state flags feature in Stateflow has been removed

As a replacement, you can use active state data with child activity mode. Refer to [Basics on Working with Active State Data](#) ([TargetLink Preparation and Simulation Guide](#)).

The removal of the support of user state flags was announced with TargetLink 4.3.

Support of AUTOSAR 2.x and 3.x	<p>With TargetLink 5.0, the support of AUTOSAR 2.x and AUTOSAR 3.x has been removed.</p> <p>The removal of the support for AUTOSAR 2.x and AUTOSAR 3.x was announced with TargetLink 4.3.</p>
TargetLink Blockset (stand-alone)	<p>With TargetLink 5.0, the setup routine of the TargetLink Blockset (stand-alone) has been discontinued.</p> <p>If you do not require the features of a fully-featured TargetLink installation, you can install TargetLink via the dSPACE Setup and adjust the operation mode to Modeling Only. You can adjust the operation mode via the TargetLink Preferences Editor or via the API command <code>tlOperationMode</code>.</p> <p>The removal of the Blockset (stand-alone) setup was announced with TargetLink 4.4.</p> <p>Related documentation</p> <ul style="list-style-type: none"> ▪ tlOperationMode (TargetLink API Reference) ▪ Overview of the TargetLink Operation Modes (TargetLink Blockset Guide)

Obsolete API Functions

Obsolete API functions	Function	Status	Replacement
	<code>tl_adapt_dd_references</code>	Error ¹⁾	<code>tlMoveDDObject</code>
	<code>tl_extract_subsystem</code>	Error ¹⁾	<code>tlExtractSubsystem</code>
	<code>tl_find_dd_references</code>	Error ¹⁾	<code>tlFindDDReferences</code>
	<code>tl_get_blockset_mode</code>	Error ¹⁾	<code>tlOperationMode</code>
	<code>tl_switch_blockset</code>	Error ¹⁾	<code>tlOperationMode</code>
	<code>tl_sim_interface</code>	Error ¹⁾	<code>tlSimInterface</code>
	<code>tl_upgrade</code>	Error ¹⁾	<code>tlUpgrade</code>
	<code>generate_ASAP2</code>	Error ¹⁾	<code>dsdd_export_a2l_file</code>
	<code>tl_upgrade_libmapfile²⁾</code>	Error ¹⁾	-

¹⁾ The function was removed from TargetLink.

²⁾ The support for libmaps in the TargetLink Version 2.x format was removed.

Note

See the help contents on the new API functions to adjust your user scripts accordingly.

**Obsolete Data Dictionary
MATLAB API functions**

Function	Status	Replacement
CreateFunctionDocumentation ¹⁾	Error ²⁾	CreateDocumentation
GetFunctionDocumentation ¹⁾	Error ²⁾	GetDocumentation
GetFunctionDocumentationBlockComment ¹⁾	Error ²⁾	GetBlockComment
GetFunctionDocumentationCodeComment ¹⁾	Error ²⁾	GetCodeComment
SetFunctionDocumentationBlockComment ¹⁾	Error ²⁾	SetBlockComment
SetFunctionDocumentationCodeComment ¹⁾	Error ²⁾	SetCodeComment

¹⁾ With TargetLink 5.0, DD Documentation objects changed. For more information, refer to [DD Documentation objects](#).

²⁾ The function was removed from TargetLink.

Obsolete Limitations

Obsolete with TargetLink 5.1

With TargetLink 5.1, the following limitations of previous TargetLink versions were removed:

Simulink.Bus objects

The use of `Simulink.Bus` objects in Stateflow diagrams is not supported. For Stateflow data that are `Simulink.Bus` objects, the following error message will be displayed:

E20950 The data type Bus Object is currently unsupported for the variable <x> in state charts.

No export of CodeDecoration objects

For AUTOSAR version 2.1.4, TargetLink does not support the export of CodeDecoration objects.

DD API does not detect data model compliance of partial DD files

The DD API does not detect if a partial DD file complies with the current data model. For example, if you load a partial DD file created with a different TargetLink version, the active DD workspace might become invalid.

Discontinuations as of TargetLink 4.4

Where to go from here

Information in this section

Discontinued TargetLink Features	406
Obsolete API Functions.....	406
Obsolete Code Generator Options.....	406
Obsolete Limitations.....	407

Discontinued TargetLink Features

No discontinued TargetLink features

No features have been discontinued with TargetLink 4.4.

Obsolete API Functions

No obsolete API functions

No API functions have been discontinued with TargetLink 4.4.

Obsolete Code Generator Options

Obsolete Code Generator options as of TargetLink 4.4

The following Code Generator options were removed from TargetLink:

Removed Option	Additional Information
CreateRestartFunctions	As a replacement, you can use the new InitializeVariablesViaRestartFunction option.

More migration aspects regarding Code Generator options

For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 314.

Obsolete Limitations

Obsolete with TargetLink 4.4 With TargetLink 4.4, the following limitations of previous TargetLink versions were removed:

General limitations	Fast restart behavior Before a simulation run starts, TargetLink checks whether Simulink's Fast Restart mode is active. If yes, an error occurs in TargetLink and the simulation features are disabled, i.e., logging and overflow detection. The following limitation applies: <ul style="list-style-type: none">▪ Simulink automatically disables the Fast Restart mode when starting a simulation if at least one TargetLink subsystem is in SIL or PIL mode.
	TargetLink subsystem IDs Each TargetLink subsystem must have a unique ID string with up to 10 characters. You can enter the ID on the Code Generation page of the TargetLink Main Dialog.
	ModelDataLogs logging format If the Simulink ModelDataLogs logging format is active, signals in referenced models that are instantiated multiple times cannot be logged. Use the DataSet logging format instead.
Block-specific limitations	Constant block Constant blocks whose data type is specified by using a Simulink.Bus object are not supported by TargetLink.
Code generation limitations	Structured variables TargetLink does not support arrays of structured variables.
The following AUTOSAR limitations no longer exist:	Import of ARRAY-SPECIFICATION elements modeled as arrays of struct When importing value specifications that are specified as 1-D or 2-D arrays of struct, TargetLink only imports the first array element that contains a RECORD-SPECIFICATION into the Data Dictionary as a struct. No NvData communication For the following access points, only sender-receiver communication is supported but not NvData communication: <ul style="list-style-type: none">▪ DataSendPoint▪ DataReceivePointByArgument

The following Data Dictionary limitation no longer exists:

DD objects with write protection

If a DD object is write-protected, i.e., Access="rwr-", "rw--", "r-r-", or "r---", its properties cannot be modified and its children cannot be deleted. However, it is possible to add new children to a write-protected DD object.

Discontinuations as of TargetLink 4.3

Where to go from here

Information in this section

Discontinued TargetLink Features	409
Obsolete API Functions.....	409
Obsolete Limitations.....	410

Discontinued TargetLink Features

MISRA C:2004 Compliance Documentation document

As announced previously, the MISRA C:2004 Compliance Documentation document was discontinued. Use the [MISRA C:2012 Compliance Documentation](#) document together with [MISRA C:2012 Addendum 1 - Rule mapping](#) instead.

Obsolete API Functions

List of obsolete API functions

Function	Status	Replacement
<code>tl_distribute_refmodel_files</code>	Error ¹⁾	_ ²⁾
<code>tl_integrate_refmodel_files</code>	Error ¹⁾	_ ²⁾
<code>tl_propman</code>	Warning ³⁾	<code>tlPropman</code>

¹⁾ The function was removed from TargetLink.

²⁾ Specify artifact locations in the Data Dictionary instead. Refer to [Specifying the Location of Artifacts Generated or Used by TargetLink](#) ([TargetLink Customization and Optimization Guide](#)).

³⁾ The function is obsolete and will be removed in a future version of TargetLink.

Compatibility consideration Adapt your user scripts and tool chain accordingly.

Obsolete Limitations

Obsolete with TargetLink 4.3

With TargetLink 4.3, the following limitations of previous TargetLink versions were removed:

AUTOSAR limitations

Unsupported calls to RTE API functions

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

- Rte_IsUpdated

Block-specific limitations

Inheritance of block properties

The Code Generator does not support any Min/Max constraints at the input and output signals of the following blocks:

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

Logical Operator block

TargetLink does not support the NXOR operation.

Stateflow Limitations

State reset behavior of function-called charts

TargetLink supports only the setting inherit for the state reset behavior of function-called charts.

Subsystem Creation

Limitations

States in function-call-triggered subsystems

MATLAB/Simulink provides the States when enabling property for the Trigger block when the function-call trigger type is selected. It lets you influence the way states are dealt with in function-call-triggered subsystems. For function-call-triggered subsystems, TargetLink does not support the held and reset settings of the States when enabling property (this limitation does not apply to the root level of referenced models). To avoid problems with function-call-triggered subsystems, select inherit.

Discontinuations as of TargetLink 4.2

Where to go from here

Information in this section

Discontinued TargetLink Features	411
Obsolete API Functions.....	412
Obsolete Code Generator Options.....	412
Obsolete Limitations.....	413

Discontinued TargetLink Features

Fully built V-ECUs as OSA

TargetLink no longer builds ECUs as OSA files. TargetLink only exports the generated production code as V-ECU implementations (CTLGZ files) and leaves the platform-specific build process to VEOS for offline simulation and ConfigurationDesk for real-time simulation.

Generation of RTF documents

TargetLink can no longer generate documentation in Rich Text Format (RTF). If you need RTF files, use the HTML generation functionality in combination with third party tools such as Microsoft Word to produce RTF files.

EPS graphics for documentation generation

EPS files are no longer supported for documentation generation (`ImageFormatType t1doc`). You can now use SVGs in HTML and PDF documentation to obtain high-resolution screenshots. Refer to [General Enhancements and Changes](#) on page 104.

A2L import

TargetLink's Data Dictionary no longer provides a default import of A2L files into the Subsystem area.

Obsolete API Functions

Overview of obsolete and removed API functions

Function	Status	Replacement
<code>get_tloptions</code>	Error ¹⁾	<code>tl_global_options</code>
<code>tl_build_vecu</code>	Error ¹⁾	-
<code>tl_compile_vecu</code>	Error ¹⁾	-
<code>tl_vecu_compiler</code>	Error ¹⁾	-
<code>dsdd('AddCodegenOptions' ...)</code>	Error ¹⁾	<code>dsdd('CreateTLPDefinedOptionSet' ...)</code>
<code>dsdd('CreateCodegenOptionSetsTLPDefinedOptionset' ...)</code>	Error ¹⁾	<code>dsdd('AddCodegenOptionSet' ...)</code>

¹⁾ The function was removed from TargetLink.

Compatibility consideration Adapt your user scripts and tool chain accordingly.

Related topics

References

[tl_global_options](#) ( TargetLink API Reference)

Obsolete Code Generator Options

Obsolete Code Generator options as of TargetLink 4.2

The following Code Generator options were removed from TargetLink:

Removed Option	Additional Information
<code>ConsiderFunctionClassesForBlockDiagramBasedSwitchOptimization</code>	In TargetLink 4.1 this option was set to <code>on</code> by default. TargetLink now always behaves as if this option was set to <code>on</code> .
<code>ConsiderStateflowAuxiliariesForVariableSharing</code>	In TargetLink 4.1 this option was set to <code>on</code> by default. TargetLink now always behaves as if this option was set to <code>on</code> .
<code>EnableVariableVectorWidths</code>	In TargetLink 4.1 this option was set to <code>on</code> by default. TargetLink now always behaves as if this option was set to <code>on</code> . This has only an effect if you use <code>ExchangeableWidth</code> objects in the TargetLink DataDictionary and your model.

Removed Option	Additional Information
TreatAllStateflowFunctionsAsWeakAtomic	In TargetLink 4.1 this option was set to on by default. TargetLink now always behaves as if this option was set to on .

More migration aspects regarding Code Generator options For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 333.

Related topics

References

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

Obsolete Limitations

Obsolete limitations as of TargetLink 4.2 With TargetLink 4.2, the following limitations of previous TargetLink versions were removed:

General limitations

Starting MIL simulation via sim command

For MATLAB R2015b (64-bit), starting a MIL simulation by using the Simulink `sim` command from within a function leads to an error. Use TargetLink's `tl_sim` command instead, which is generally recommended.

Non-ASCII characters in Windows user name

With the 64-bit version of MATLAB R2014a, it can happen that the TargetLink demos start page is not displayed correctly if your Windows user name contains non-ASCII characters, for example, ü in Müller. In such a case, the right-hand pane is empty.

Block-specific limitations

Matrix Concatenate block

TargetLink supports this block only if the Mode block parameter is set to Multidimensional array.

Data Store blocks

For Data Store Memory/Data Store Read/Data Store Write blocks the following limitation applies:

- The number of input and output ports is limited to one.
- *Partial* reading from and writing to data stores is not supported.

Data Store Memory block

Data Store Memory blocks whose data type specifies a bus are not supported by TargetLink.

Code generation limitations

No CTC code coverage measurements with the LCC compiler

The LCC compiler cannot be used with the CTC Testwell third party tool to perform code coverage measurements. However, you can use TargetLink's own code coverage measurement instead.

AUTOSAR limitations

Data type of interruptable variables

Interruptable variables specified in TargetLink must be scalar.

Stateflow

Graphical function with more than one data output

It is not possible to define and call graphical functions with more than one output data.

Discontinuations as of TargetLink 4.1

Where to go from here

Information in this section

Obsolete Code Generator Options.....	415
Obsolete Limitations.....	416

Obsolete Code Generator Options

Obsolete Code Generator option as of TargetLink 4.1

The MPC5xx-specific (TOM-specific) EnableLogicalOperationOptimisation Code Generator option was removed from TargetLink:

Removed Option	Replacement Option	Compatibility Setting
EnableLogicalOperationOptimisation	None	None

With TargetLink 4.1, this option was removed together with MPC5xx-support.

More migration aspects regarding Code Generator options

For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 339.

Related topics

References

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

Obsolete Limitations

Obsolete limitations as of TargetLink 4.1

With TargetLink 4.1, the following limitations of previous TargetLink versions were removed:

General limitations

Initialization of buses¹⁾

Bus signals can be initialized only if one of the following conditions is met:

- All the bus signals are enumerations of the same type.
- All the bus signals are non-enumerations.

Basically, the same scalar initial value has to be applied to all the bus signals.

¹⁾ This limitation became obsolete because TargetLink 4.1 supports Simulink's simplified mode.

Block-specific limitations

Look-up tables (vectorized)

Table look-up functions are replaced by user-provided implementations if a matching custom look-up script is available. This applies to look-up table blocks with a scalar output. The custom look-up script mechanism does not support vector/matrix input signals or uniform elements.

Merge block

It is not possible to specify the initial value of bus signal elements by setting the initial output parameter of blocks that pass the bus signal to a Simulink Initial Condition Structure.

OutPort block

It is not possible to specify the initial value of bus signal elements by setting the initial output parameter of blocks that pass the bus signal to a Simulink Initial Condition Structure.

Rate Transition block

It is not possible to specify the initial value of bus signal elements by setting the initial output parameter of blocks that pass the bus signal to a Simulink Initial Condition Structure.

Component-based development limitations

Function reuse

TargetLink does not support multiple references to the same model inside one model hierarchy. Therefore, code generated for referenced models cannot be reused.

Simulink model arguments

TargetLink does not support Simulink model arguments for referenced models.

Code generation limitations**Incremental code generation**

Incrementally generated subsystems or referenced models cannot be subject to function reuse and cannot reside in reused functions.

Discontinuations as of TargetLink 4.0

Where to go from here	Information in this section
	Discontinued TargetLink Features 418
	Discontinued Data Dictionary Features 418
	Obsolete API Functions..... 419
	Obsolete Code Generator Options..... 420
	Obsolete Limitations..... 420

Discontinued TargetLink Features

Discontinued TOMs	As of the TargetLink version 4.0, the following TOMs are no longer available: <ul style="list-style-type: none">▪ HCS12: HCS12/Cosmic 4.7 HCS12/CodeWarrior 3.1▪ MPC5xx: MPC5xx/Green Hills 5.1 MPC5xx/WindRiver Diab 5.7▪ MPC55xx: MPC55xx/WindRiver Diab 5.7▪ M32R: M32R/GAIO 9
--------------------------	--

Discontinued Data Dictionary Features

Use Case filter in the Data Dictionary Manager	The Use Case filter function is no longer available in the Data Dictionary Manager. You can now use and specify your own XML-based filter rule sets. For further information on how to generate filter rule sets, refer to Basics on Filter Rule Sets for the Data Model (TargetLink Data Dictionary Basic Concepts Guide) and How to Create Filter Rule Sets via DD Files (TargetLink Data Dictionary Basic Concepts Guide).
---	---

Obsolete API Functions

Obsolete API functions as of TargetLink 4.0

Function	Status	Replacement
<code>build_customcode_sfcn</code>	Error ¹⁾	<code>tl_build_customcode_sfcn</code>
<code>get_mdltdata</code>	Error ¹⁾	-
<code>get_sfobjects</code>	Error ¹⁾	<code>tl_get_sfobjects</code>
<code>get_tlblocks</code>	Error ¹⁾	<code>tl_get_blocks</code>
<code>tl_adapt_dd_references</code>	Warning ²⁾	<code>tlMoveDDObject</code>
<code>tl_extract_subsystem</code>	Warning ²⁾	<code>tlExtractSubsystem</code>
<code>tl_find_system</code>	Error ¹⁾	<code>find_system</code>
<code>tl_get_blockset_mode</code>	Warning ²⁾	<code>tlOperationMode</code>
	Error ¹⁾	<code>dsdd_manage_project('GetProjectFile', simulinkSystem)</code>
<code>tl_get_project</code>	Error ¹⁾	<code>dsdd_manage_project('SetProjectFile', projectName, simulinkSystem)</code>
<code>tl_set_project</code>		
<code>tl_sim_interface</code>	Warning ²⁾	<code>tlSimInterface</code>
<code>tl_switch_blockset</code>	Warning ²⁾	<code>tlOperationMode</code>

¹⁾ The function was removed from TargetLink.

²⁾ The function is obsolete and will be removed in a future version of TargetLink.

Compatibility consideration Adapt your user scripts and toolchain accordingly.

Related topics

References

[dsdd_manage_project\('GetProjectFile', simulinkSystem\)](#) (TargetLink API Reference)
[dsdd_manage_project\('SetProjectFile', projectName, simulinkSystem\)](#) (TargetLink API Reference)
[tl_build_customcode_sfcn](#) (TargetLink API Reference)
[tl_get_blocks](#) (TargetLink API Reference)
[tl_get_sfobjects](#) (TargetLink API Reference)
[tlExtractSubsystem](#) (TargetLink API Reference)
[tlMoveDDObject](#) (TargetLink API Reference)
[tlOperationMode](#) (TargetLink API Reference)
[tlSimInterface](#) (TargetLink API Reference)

Obsolete Code Generator Options

Obsolete Code Generator option as of TargetLink 4.0

The following Code Generator option was removed from TargetLink:

Removed Option	Replacement Option	Compatibility Setting
EfficientVectorHandling	LoopUnrollThreshold () ¹⁾ .	INF, if EfficientVectorHandling was set to off

- ¹⁾ For details, refer to [Removal of EfficientVectorHandling](#) on page 542 and [Basics on Processing Vectors and Matrices](#) (.

More migration aspects regarding Code Generator options

For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 350.

Related topics

References

[Code Generator Options](#) (

Obsolete Limitations

Obsolete limitations as of TargetLink 4.0

With TargetLink 4.0, the following limitations of previous TargetLink versions were removed:

General limitations

Multidimensional signals

TargetLink does not support multidimensional signals (matrix signals) at block outputs. Matrix parameters are supported only for Look-Up Table (2-D), Interpolation Using Prelookup, Direct Look-Up Table (n-D), Discrete State-Space, and Custom Code blocks. Parameters with more than two dimensions are not possible.

In Stateflow, the interface cannot contain multidimensional signals, but local variables can be matrices. For details on Stateflow limitations, refer to [Stateflow Limitations](#) (.

Note

This TargetLink version now supports 2-D matrix signals. However, 3-D signals (or higher) are not supported.

Block-specific limitations**Gain block**

These blocks are only supported by TargetLink when the Multiplication parameter in the Simulink block dialog is set to Element-wise($K.*u$) (default).

TargetLink AUTOSAR Module**Limitations****Model-linked code view**

TargetLink does not support the model-linked code view for blocks with activated AUTOSAR communication.

Updating frame model / Name modifications

When updating a generated frame model from AUTOSAR data, TargetLink treats renamed AUTOSAR elements as new. TargetLink prompts you to remove the blocks carrying the old names from the model and adds the blocks with the new names.

Discontinuations as of TargetLink 3.5

Where to go from here

Information in this section

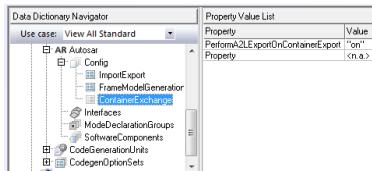
Discontinued Data Dictionary Features	422
Obsolete Limitations.....	423

Discontinued Data Dictionary Features

Discontinuation of EnableDefaultContainerExpor tHook property

The DD EnableDefaultContainerExportHook property, was removed from the /Pool/Autosar/Config/ImportExport subtree, due to a change in how hook script templates are shipped (refer to [Hook Script Templates](#) on page 359). TargetLink displays a warning if your DD file contains the discontinued property.

With TargetLink 3.5, AUTOSAR files and A2L files are gathered by the **tl_export_container** API function. To enable/disable A2L export during container export, a new AUTOSAR option set is provided in the dsdd_master_autosar.dd template.



You can merge it from a new DD Workspace, based on the dsdd_master_autosar.dd template template, or create it yourself.

AUTOSAR Option Set	Custom Property
/Pool/Autosar/Config/ContainerExchange	PerformA2LExportOnContainerExport

Related topics

References

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

Obsolete Limitations

Obsolete limitations as of TargetLink 3.5

With TargetLink 3.5, the following limitations existing in previous TargetLink versions have been removed. They are listed below.

TargetLink simulations

Simulation Rewind

Rewinding (stepping backward) in simulations is not possible in TargetLink MIL/SIL/PIL simulation modes.

Note

Rewinding is now possible in TargetLink MIL simulation mode. However, the limitation has not completely been solved, refer to [General Limitations](#) ([TargetLink Limitation Reference](#)).

Stateflow support

Execute (enter) Chart at Initialization

TargetLink does not support the Execute (enter) Chart At Initialization chart property. Any setting of this chart property is ignored and TargetLink generates a warning message.

Note

TargetLink now supports this chart property, refer to [Basics on Specifying Stateflow Models](#) ([TargetLink Preparation and Simulation Guide](#)).

Discontinuations as of TargetLink 3.4

Where to go from here

Information in this section

Discontinued Data Dictionary Features	424
Obsolete Limitations.....	424

Discontinued Data Dictionary Features

`dsdd_upgrade`

The `dsdd_upgrade` command is no longer available.

To upgrade a DD project file, use the `dsdd('Upgrade'[,<DD_Identifier>])` command (refer to [Upgrade](#)), or the DD Manager.

Obsolete Limitations

Obsolete limitations as of TargetLink 3.4

With TargetLink 3.4, a couple of limitations existing in previous TargetLink versions have been removed. They are listed below.

Component-Based Development Limitations

AUTOSAR

AUTOSAR blocks are not supported in referenced models or subsystems configured for incremental code generation.¹⁾

¹⁾ AUTOSAR blocks still are not allowed in model-based code generation units containing functional sub-parts of runnables.

Code generation limitation

Struct data types cannot be accessed by access functions

Access functions are not supported for structures as such. Only access to the components of a structure which are not themselves of struct type is supported.

Scope of structured variables

TargetLink reduces the scope of structures only to static local at most but not to local, even if this is technically possible. If the structure has the local scope to start with, it keeps the local scope.

	<p>Access function kind ADDRESS</p> <p>Implementing the ADDRESS access function kind in the form of macros would require different implementations for scalar and vector variables. Currently, only one implementation can be specified, and it is therefore necessary to use separate variable classes for scalar and vector variables that reference different AccessFunction templates.</p>
Stateflow limitations	<p>Shift by variable</p> <p>TargetLink does not support shift operations with a shift value that is a variable. For example:</p> <ul style="list-style-type: none">▪ <code>a >> 5; // ok</code>▪ <code>a >> ShiftValue; // not supported</code>
AUTOSAR limitations	<p>RTE API calls in referenced models</p> <p>TargetLink does not support the use of AUTOSAR communication in referenced models. If AUTOSAR data or parameters shall be used in referenced models, the data and parameters have to be passed to the referenced models via port blocks with deactivated AUTOSAR communication.</p>

Discontinuations as of TargetLink 3.3

Where to go from here

Information in this section

Discontinued Data Dictionary Features	426
Obsolete Code Generator Options.....	426
Obsolete Limitations.....	427

Discontinued Data Dictionary Features

DD features no longer available

With this version of the dSPACE Data Dictionary, the following features are no longer available:

- Model Browser
- AUTOSAR SWC-D Merge Explorer
- DD Merge Explorer

You can use the Property Manager and the new DD Compare Tool instead.

Obsolete Code Generator Options

Obsolete Code Generator option as of TargetLink 3.3

The following Code Generator option is no longer used in TargetLink 3.3:

- OptimizationLevel

This option was replaced by the new Optimization option.

Old optimization levels 0,1,2 have been replaced with Optimization=on (formerly 1,2) and Optimization=off (formerly 0). Basically, you do not have to do anything unless you use a `pre_codegen_hook` file to set an optimization level directly. If so, replace the old values (0,1,2) with the new ones (`off`, `on`).

For details, refer to [Optimization \(Model element\)](#) ( [TargetLink Model Element Reference](#)).

If you generate code while the obsolete options are still in the model, a warning occurs.

More migration aspects regarding Code Generator options

For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 370.

Related topics**References**

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

Obsolete Limitations

Obsolete limitations as of TargetLink 3.3

With TargetLink 3.3, a couple of limitations existing in previous TargetLink versions have become obsolete. They are listed below.

General limitations

Wrong Overflow Detection for bus Signals

Overflow detection identifies wrong bus signal element if the bus signal consists of demuxed signals.

Logging a simulation

If you use MATLAB Release 2008a (R2008a) or earlier, blocks cannot be logged in MIL simulation mode if their input is fed by a MUX output signal and several MUX block inputs are connected to the same input signal.

Referenced Models

Starting a MIL simulation via the `tl_sim` command or the Start Simulation Tool Selector button fails if the referenced model file is write-protected. Allow write access to thereferenced model, or start the MIL simulation via the Simulink start simulation button.

Block-specific limitations

Direct Look-Up Table block

It is not possible to model a Direct Look-Up Table block capable of uniform elements at the block output. The block output always has the same width as the index input.

Product block

For MIL simulations, the multiplication of two 32-bit integer signals is not supported (only MATLAB Release R2008a and older).

Limitation in creating a TargetLink subsystem

InPort width

TargetLink InPort blocks at the boundary of a TargetLink subsystem should have their TargetLink Width property set to the width of the incoming signal. Otherwise, initialization in SIL/PIL mode might fail.

Code generation limitation	\$E name macro not supported for reuse structures Variables with the \$E name macro are not supported, because it can happen that variables passed to reuse structures (states, parameters, global variables, etc.) are given different names in different instances.
AUTOSAR limitation	Struct types at AUTOSAR interfaces TargetLink does not support importing/exporting nested struct types at AUTOSAR interfaces. The data types supported for struct components that are used at AUTOSAR interfaces are scalar and array types.
Stateflow limitation	Sharing global data between Simulink and Stateflow MATLAB provides an interface that gives Stateflow charts access to global variables in Simulink models. Simulink implements global variables as data stores, created either as data store memory blocks or as instances of Simulink. Signal objects. Stateflow charts can share global data with Simulink by reading and writing data store memory symbolically using the Stateflow action language. This feature is not supported by TargetLink. If you try to access global data, for example specified in Data Store Memory blocks, using the Stateflow action language from within TargetLink, the following error message will be displayed: E20941 Stateflow data with scope data store memory is currently not supported.
ASAM MCD-2 MC file generation limitation	State Space blocks For calibratable matrices, the Keep vector structure option must be enabled.

Discontinuations as of TargetLink 3.2

Obsolete Code Generator Options

Obsolete Code Generator options as of TargetLink 3.2

The following Code Generator options are no longer used in TargetLink 3.2:

- InvalidateCodeOnError
- TreatSpecificErrorsAsWarnings
- OmitZeroInitializationsInStartupFunction

If you generate code while the obsolete options are still in the model, a warning occurs.

More migration aspects regarding Code Generator options

For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 380.

Related topics

References

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

Discontinuations as of TargetLink 3.1

Obsolete Code Generator Options

Obsolete Code Generator options as of TargetLink 3.1

The following Code Generator options are no longer used in TargetLink 3.1:

EnableMultirate The EnableMultirate option is obsolete. Use the CodeGenerationMode = RTOS setting instead.

ScopeReductionPreserveLinkerSectionEntriesUserClass The ScopeReductionPreserveLinkerSectionEntriesUserClass option is no longer used.

VariantFileDefaultID The VariantFileDefaultID option is obsolete. As of TargetLink 3.1 you can set the ModuleRef property directly in the DataVariant object specifying a data variant.

If your model contains obsolete Code Generator options that are set to values other than the default, the model upgrade does not delete these options automatically. If you generate code while the obsolete options are still stored at the model, a warning occurs.

More migration aspects regarding Code Generator options

For more information on migration aspects regarding Code Generator options, refer to [Migration Aspects Regarding Code Generator Options](#) on page 391.

Related topics

References

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

Code Changes

Where to go from here

Information in this section

Code Changes Between TargetLink 5.0 and TargetLink 5.1.....	432
Code Changes Between TargetLink 4.4 and TargetLink 5.0.....	456
Code Changes Between TargetLink 4.3 and TargetLink 4.4.....	470
Code Changes between TargetLink 4.2 and TargetLink 4.3.....	487
Code Changes between TargetLink 4.1 and TargetLink 4.2.....	510
Code Changes between TargetLink 4.0 and TargetLink 4.1.....	521
Code Changes between TargetLink 3.5 and TargetLink 4.0.....	535
Code Changes between TargetLink 3.4 and TargetLink 3.5.....	547
Code Changes between TargetLink 3.3 and TargetLink 3.4.....	551

Code Changes Between TargetLink 5.0 and TargetLink 5.1

Where to go from here	Information in this section
	AUTOSAR.....432
	Auxiliary Variables.....433
	Basetype-Related Code Changes.....434
	Comment-Related Code Changes.....435
	Control Flows.....437
	Custom Code.....438
	Function Inlining.....439
	Interprocedural Analysis.....439
	Initialization.....446
	Logical Expressions.....448
	Range Propagation.....449
	Shift Operations.....450
	Stateflow and MATLAB Code.....450
	Variable Optimization.....452
	Variable Scope.....453
	Other.....453

AUTOSAR

Optimizing comparisons of RTE API calls TargetLink now optimizes not equal comparisons of identical RTE API calls.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>if(Rte_IRead_Run_NvSRP_Deep_01()->e.b != Rte_IRead_Run_NvSRP_Deep_01()->e.b) { ... }</pre>	<pre>if (0) { ... } (and subsequent removal)</pre>

Reason Code efficiency

Auxiliary Variables

Auxiliary variables in sqrt function calls

In calls of the `sqrt` function, TargetLink now uses an auxiliary variable if the operand of the function is an expression that is more complex than a simple variable access. This prevents that the expression is calculated twice:

TargetLink ≤ 5.0

```
if (((Ca6_F64Index1 + ((Float64) Ca6_F32Index1)) - Ca6_F64Index1) >= 0.) {
    AUX_var_d = (Int32) sqrt((Ca6_F64Index1 + ((Float64) Ca6_F32Index1)) - Ca6_F64Index1);
} else {
    AUX_var_d = (Int32) (-sqrt(-((Ca6_F64Index1 + ((Float64) Ca6_F32Index1)) - Ca6_F64Index1)));
}
```

TargetLink 5.1

```
Aux_ = (Ca6_F64Index1 + ((Float64) Ca6_F32Index1)) - Ca6_F64Index1;
if (Aux_ >= 0.) {
    AUX_var_d = (Int32) sqrt(Aux_);
} else {
    AUX_var_d = (Int32) (-sqrt(-Aux_));
}
```

This change occurs mainly in Stateflow or MATLAB code.

Additionally, when generating unoptimized code, the names of the auxiliary variables might change.

Reason Code efficiency

Auxiliary variables for min, max and unary minus

In `min`, `max`, and `unary minus` operations, TargetLink now uses an auxiliary variable if the operand is an expression that is more complex than a simple variable access. This prevents that the expression is calculated twice:

TargetLink ≤ 5.0	TargetLink 5.1
<pre><code>if (((Int32) Ca1_I16In1) + 5) < (((Int32) Ca1_I16In2) - 12)) { Ca1_I32Out = ((Int32) Ca1_I16In1) + 5; } else { Ca1_I32Out = ((Int32) Ca1_I16In2) - 12; }</code></pre>	<pre><code>Aux_ = ((Int32) Ca1_I16In1) + 5; Aux__a = ((Int32) Ca1_I16In2) - 12; if (Aux_ < Aux__a) { Ca1_I32Out = Aux_; } else { Ca1_I32Out = Aux__a; }</code></pre>

Reason Code efficiency

Dimension of auxiliary matrix variables TargetLink now might downgrade the dimension of 64-Bit auxiliary matrix variables.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>for(i = 0; ...) { F_I64MUL..(Const, &Aux_Hi[0][i], &Aux_LO[0][i]); ... }</pre>	<pre>for(i = 0; ...) { F_I64MUL..(Const, &Aux_a, &Aux_b); ... }</pre>

Reason Code efficiency

Basetype-Related Code Changes

Macros for minimum and maximum values for floating-point types

With TargetLink 5.1, you can use predefined DD Variable objects to model the use of floating-point limits. This will result in the use of macros in the production code. Refer to [Basics on Floating-Point Limits in TargetLink \(TargetLink Preparation and Simulation Guide\)](#). The macros are named as follows:

- tl_FLOAT32MAX
- tl_FLOAT32NMIN
- tl_FLOAT64MAX
- tl_FLOAT64NMIN

In addition to user-specified uses of the macros, the macros occur in the exception handling of the following operations:

- division with floating-point result
- RSqrt (= 1/sqrt)
- atanh
- log, log₁₀

Example for the division:

TargetLink ≤ 5.0	TargetLink 5.1
<pre> if (Sa1_InPort1 != 0) { Sa1_NoMinNoMax = ((Float32) Sa1_InPort) / ((Float32) Sa1_InPort1); } else { if (((Float32) Sa1_InPort) < 0.F) { Sa1_NoMinNoMax = -3.402823466e+38F; } else { Sa1_NoMinNoMax = 3.402823466e+38F; } } </pre>	<pre> if (Sa1_InPort1 != 0) { Sa1_NoMinNoMax = ((Float32) Sa1_InPort) / ((Float32) Sa1_InPort1); } else { if (((Float32) Sa1_InPort) < 0.F) { Sa1_NoMinNoMax = tl_FLOAT32NMIN; } else { Sa1_NoMinNoMax = tl_FLOAT32MAX; } } </pre>

Reason Improved platform independence of the production code

Migration issue With this change, the names of the macros and the corresponding macros for simulation frame and Fixed-Point library are treated as identifiers reserved for use by TargetLink. Refer to [Migration Aspects regarding reserved identifiers](#) on page 301.

Comment-Related Code Changes

Omitting identical comments TargetLink now omits identical comment parts for each statement comment.

TargetLink ≤ 5.0	TargetLink 5.1
<pre> /* # combined # BusOutport: Subsystem/Runnable/Bus Outport # combined # Gain: Subsystem/Runnable/Gain # combined # BusOutport: Subsystem/Runnable/Bus Outport */ Rte_Call_ClientPort_Setter(Scalar * 5, (const sint16 *) Vector, (const RootType *) &Struct_a); </pre>	<pre> /* # combined # BusOutport: Subsystem/Runnable/Bus Outport # combined # Gain: Subsystem/Runnable/Gain */ Rte_Call_ClientPort_Setter(Scalar * 5, (const sint16 *) Vector, (const RootType *) &Struct_a); </pre>

Reason

- Bug fix
- Readability
- Matches user expectations

Sorting operation comment parts TargetLink now transfers the comment part of the outermost operation to the first position.

TargetLink ≤ 5.0	TargetLink 5.1
<pre> /* Abs: subsystem/ABS # combined # TargetLink outport: subsystem/OutPort */ Sa1_OutPort = (Int8) __satb(((__sat) __abs((Int16) (((Int16) Sa1_InPort) - 158))) - ((__sat) 12)); </pre>	<pre> /* TargetLink outport: subsystem/OutPort # combined # Abs: subsystem/ABS */ Sa1_OutPort = (Int8) __satb(((__sat) __abs((Int16) (((Int16) Sa1_InPort) - 158))) - ((__sat) 12)); </pre>

Reason

- Bug fix
- Easier matching of model and code

Unifying block type, block path and code comment TargetLink now adds the block type and block path for the Merge and Rate Limiter block.

TargetLink ≤ 5.0	TargetLink 5.1
<code>/* Reference of merge block: Merge subsystem/Merge */</code>	<code>/* Merge: subsystem/Merge */</code>
<code>/* subsystem/Rate Limiter: output */</code>	<code>/* Rate Limiter: subsystem/Rate Limiter: output */</code>

For the Assignment, Custom Code, and the dynamic Selector block, TargetLink now adds the separator : between the block path and the code comment.

TargetLink ≤ 5.0	TargetLink 5.1
<code>/* Assignment: subsystem/Assignment - output initialization */</code>	<code>/* Assignment: subsystem/Assignment: output initialization */</code>
<code>/* Custom code: subsystem/CustomCode << output code >> */</code>	<code>/* Custom code: subsystem/CustomCode: << output code >> */</code>
<code>/* Selector: subsystem/Selector - init phase */</code>	<code>/* Selector: subsystem/Selector: init phase */</code>

Reason

- Bug fix
- Easier matching of model and code
- Matches user expectations

Slashes in include statements TargetLink now uses slashes (/) instead of backslashes (\) in include statements.

TargetLink ≤ 5.0	TargetLink 5.1
<code>#include "builtin_functions\standard_mathematics.h"</code>	<code>#include "builtin_functions/standard_mathematics.h"</code>

Reason MISRA C compliance

List of charts and statemachines in header comments

The header comments of generated files changed regarding the list of Stateflow statemachines and charts. With TargetLink 5.1 this list contains only those elements that are accessible by the [code generation unit \(CGU\)](#) for that the file was generated.

TargetLink ≤ 5.0		
<code>*** SF-NODE</code>	<code>CORRESPONDING STATEFLOW NODE</code>	<code>DESCRIPTION</code>
<code>*** Cd0</code>	<code>MyLibNoExternGraphFcn</code>	
<code>*** Cd1</code>	<code>MyLibWithExternGraphFcn</code>	
<code>*** Cd2</code>	<code>SomeModel</code>	
<code>*** Cd3</code>	<code>SomeSubsystem/Chart</code>	

TargetLink 5.1

<i>*** SF-NODE</i>	<i>CORRESPONDING STATEFLOW NODE</i>	<i>DESCRIPTION</i>
<i>*** Cd1</i>	<i>MyLibWithExternGraphFcn</i>	
<i>*** Cd2</i>	<i>SomeModel</i>	
<i>*** Cd3</i>	<i>SomeSubsystem/Chart</i>	

Reason

- Easier matching of model and code
- Matches user expectations

Control Flows

Merging control flows

TargetLink now merges the control flows in opposite, mirrored, and `#if` conditions.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>if (a >= b) { //1 } else { //2 } if (a < b) { //3 } if (a >= 0) { //1 } if (0 <= a) { //2 } if (a < b) { //1 if (b > a)) { //2 } } else { //3 if (b <= a) { //4 } } #if defined a && a >= 199 //1 #endif #if defined a && a >= 199 //2 #else //3 #endif</pre>	<pre>if (a >= b) { //1 } else { //2 //3 } if (a >= 0) { //1 //2 } if (a < b) { //1 //2 } else { //3 //4 } #if defined a && a >= 199 //1 //2 #else //3 #endif</pre>

TargetLink ≤ 5.0	TargetLink 5.1
<pre>#if MODE == 3 //1 #if MODE != 3 //2 #else //3 #endif #endif</pre>	<pre>#if MODE == 3 //1 //3 #endif</pre>

Reason Code efficiency

Migration issue You can control this optimization via the CombineControlFlowStatements Code Generator option.

Custom Code

Custom code sections and control flow

TargetLink sorts custom code sections in the production code in the following order:

- Common before not common
- Top before unqualified or bottom
- Output before update

TargetLink now also considers data flow on sorting custom code sections which can lead to the following consequences:

- Custom code output variables are not removed if a bottom output section is present, because the bottom output section potentially changes the value of the output variable after the value is read.
- Variables that are custom code input variables and for whose input direct feedthrough is specified, may not be removed when using a top output section or may not be reduced in scope to automatic storage duration if the calculation of these variables is performed after the top output section as the top output section potentially accesses the input variable.
- Variables that are custom code input variables and for whose input direct feedthrough is not specified are described earlier when using a top update section in order to prevent uninitialized read accesses.
- Update sections that are not qualified as bottom section are not placed after the other state sections. Instead these update sections are placed at the position where the state update of another block would be placed.

In some situations, the position of custom code sections in the production code can change if the data flow requires it.

Reason Bug fix

Function Inlining

Changed variable suffixes

A function might contain several calls to the same function:

```
void func() {
    Int16 Sum;
    Sum = expr;
}

void main() {
    func();
    ...
    func();
}
```

These function calls are now always inlined in the same order in which they are located in the code. This can lead to changed variable suffixes:

TargetLink ≤ 5.0	TargetLink 5.1
<code>main() { ... Sum_a = expr; ...; Sum = expr; }</code>	<code>main() { ... Sum = expr; ...; Sum_a = expr; }</code>

Reason

- Bug fix
- Matches user expectations

Interprocedural Analysis

Different statement order

In rare cases, TargetLink now might move some statements differently due to improved interprocedural and alias data flow analysis.

Reason Bug fix

AccessFunction patterns

TargetLink now uses different AccessFunction code patterns for the following situations:

- Multiple definitions before one use.
- Multiple uses before one definition.
- One use before multiple definitions.

TargetLink ≤ 5.0	TargetLink 5.1
Multiple Definitions Before One Use	
<pre>if (cond) { Aux_S16 = ...; } else { Aux_S16 = ...; } = ... Aux_S16... SetA(Aux_S16);</pre>	<pre>if (cond) { Aux_S16 = ...; } else { Aux_S16 = ...; } SetA(Aux_S16); = ... Aux_S16...;</pre>
Multiple Uses Before One Definition	
<pre>Aux_S16 = GetA(); ... = ... Aux_S16... = ... Aux_S16... ... Aux_S16 = ...; SetA(Aux_S16);</pre>	<pre>Aux_S16 = GetA(); ... = ... Aux_S16... = ... Aux_S16... ... SetA(...);</pre>
One Use Before Multiple Definitions	
<pre>Aux_S16 = GetA(); ... = ... Aux_S16... ... if (cond) { Aux_S16 = ...; } else { Aux_S16 = ...; } SetA(Aux_S16);</pre>	<pre>... = ... GetA() if (cond) { Aux_S16 = ...; } else { Aux_S16 = ...; } SetA(Aux_S16);</pre>

Reason

- Bug fix
- Code efficiency

Adding auxiliary variables when inlining for global CallByValue function parameters

Based on TargetLink's internal data flow analysis during code generation, TargetLink now adds auxiliary variables when inlining for global function arguments if the global variable might have write accesses via this function call. If a variable is passed into a function both by value and as a pointer an auxiliary variable is used when inlining, too. In many situations the auxiliary variables are removed during the optimization process.

Reason Bug fix

Write accesses of global variables in conditionally executed control flow branches

TargetLink now keeps the write accesses of global variables out of conditionally executed control flow branches in any of the following cases:

- If there may be additional accesses to the variable to be propagated outside the current CGU or in external functions.

- If the global variable has a DD VariableClass whose Optimization property does not contain MOVABLE.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>foo() { if (cond) { Out = <...> } } bar() { foo(); ... = Out }</pre>	<pre>foo () { static int x; if (cond) { x = <...> } Out = x; } bar() { foo(); ... = Out }</pre>

Reason Bug fix

Condition assignment for Boolean variables

TargetLink now optimizes Boolean variables that are specified with the predefined variable class FCN_REF_ARG.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>if (Cond == 1) { *PtrVar = 1; } else { *PtrVar = 0; }</pre>	<pre>*PtrVar = Cond == 1;</pre>

Reason Bug fix

Migration issue You can avoid this optimization via the AssignmentOfConditions Code Generator option, at the cost of losing optimization in other contexts that are targeted by this option.

Consecutive assignments to dereferenced pointers

Some TargetLink code patterns, e.g., for Stateflow graphical functions, require potentially unnecessary assignments to variables to make sure that values are assigned on all execution paths. TargetLink removes any unnecessary previous assignments. This optimization is now also possible if the value assignment takes place via a dereferenced pointer.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>pA = &a; *pA = 5; *pA = 4;</pre>	<pre>pA = &a; *pA = 4;</pre>

Reason Bug fix

Correct data flow relationships for static local variables in cyclic function calls

TargetLink now correctly considers interprocedural data flow via cyclic function calls for variables with `local` scope and `static` storage duration that have write accesses, because they might be a state of the complete cycle:

- Seemingly unnecessary write accesses can be removed only if there are no intervening calls to a function belonging to the cycle.
- Reduction of the storage duration to `auto` is only possible if no cyclic function calls occur between write and read accesses.

The following table shows a case where the seemingly unnecessary assignment `Ca1_x = 1.;` is retained:

TargetLink ≤ 5.0	TargetLink 5.1
<pre>if (Ca1_x == 1.) { Ca1_a = 10.; } else { Ca1_a = -10.; /* call of function: Subsystem/Chart2 */ Ca3_Chart2(); Ca1_x = 2.; Ca1_out = Ca1_a; }</pre>	<pre>if (Ca1_x == 1.) { Ca1_a = 10.; } else { Ca1_a = -10.; Ca1_x = 1.; /* call of function: Subsystem/Chart2 */ Ca3_Chart2(); Ca1_x = 2.; Ca1_out = Ca1_a; }</pre>

Reason Bug fix

Static local variables prevent expression movement

TargetLink no longer moves function calls that directly or indirectly contain accesses to the same static local variable past each other.

Reason Bug fix

Interfaces of global functions

In TargetLink 5.1, global functions with function reuse and Stateflow® exported graphical functions can be called outside the current [code generation unit \(CGU\)](#) and effectively become part of the CGU's interface. Consequently, TargetLink does not remove seemingly unused output variables because calls outside of the current CGU might consume them. Moreover, for exported graphical functions unnecessarily modeled input variables are also preserved in order to prevent compiler or linker errors. If some of the inputs or outputs are truly unnecessary and the functions are not to be called outside the current CGU, consider removing them from the respective function interface via changing the model.

Reason Matches user expectations

Optimizing movable variables

TargetLink now interprets the presence of `MOVABLE` in the Optimization property of a DD VariableClass object in the same way as `ERASABLE` and `SCOPE_REDUCIBLE` with respect to assumptions about data flow:

If a variable is **MOVABLE** then the modeled accesses in the current CGU are all accesses that take place, i.e., there are neither accesses in other CGUs nor in external functionality called by TargetLink unless they have been specified as part of the respective function or Custom Code interface. This affects the moving of accesses to global variables as follows:

- Past calls to functions with unknown content.
- Into conditionally executed control flow branches.

This usually leads to generated code that is more efficient.

In addition to the previously mentioned effects, TargetLink sometimes does the following:

- Eliminates more or different intermediate variables.
- Performs more control flow optimizations.

Refer to [Basics on Optimizing Variables](#) ([TargetLink Customization and Optimization Guide](#)).

Reason Code efficiency

Migration issue Refer to [Migration Aspects Regarding Optimization](#) on page 303.

Optimization in guaranteed write before read situations

For function calls or custom code, variables are now optimized as follows: Variables whose Write before read user assertion is set are treated the same way as variables whose Write only user assertion is set.

For information on the different user assertions, refer to [Basics on Using TargetLink Data Stores with Custom Code \(Type II\) Blocks](#) ([TargetLink Preparation and Simulation Guide](#)).

Example

```
f(&b /* Write Before Read */);
c = b;
```

The variable **b** can be reduced to automatic storage duration and eliminated in favour of variable **c**. If variable **b** has an assignment before the call, the assignment can be removed.

Reason Code efficiency

Saturated Abs operations

Previously, if the **Abs** operation had an unnecessary saturation and this saturation was implemented in different branches of a control flow, TargetLink generated a comment in each branch. Now, TargetLink tests earlier for unnecessary saturations of **Abs** operations. This results in less comments:

TargetLink ≤ 5.0	TargetLink 5.1
<pre> if(I16In >= 0) { /* <Block>: Omitted upper saturation * <Block>: Omitted Lower saturation */ U16Out = I16In; } else { /* <Block>: Omitted upper saturation */ U16Out = C_U16FITI16_SATL(I16In, -32768); } </pre>	<pre> /* <Block>: Omitted upper saturation */ if(I16In >= 0) { /* <Block>: Omitted lower saturation */ U16Out = I16In; } else { U16Out = C_U16FITI16_SATL(I16In, -32768); } </pre>

Additionally, the unnecessary saturation might be removed from individual branches. In the following example, the saturation of the lower bound in the `then` branch is omitted:

TargetLink ≤ 5.0
<pre> /* Abs: Inp_Int32_Outp_Int8/Abs_14 */ if (Sh1_InPort_14 >= 0) { Sh1_OutPort_14 = C_I8SHRI32C6_LT24_SATb(Sh1_InPort_14, 23, 1065353216, -1073741824); } else { /* Inp_Int32_Outp_Int8/Abs_14: Omitted lower saturation */ C_I64NEGI32(Sh1_InPort_14, Aux_S32, Aux_U32); Sh1_OutPort_14 = C_I8SHRI64C6_LT32_SATu(Aux_S32, Aux_U32, 23, 9, 0, 1065353216); } </pre>
TargetLink 5.1
<pre> /* Abs: Inp_Int32_Outp_Int8/Abs_14 Inp_Int32_Outp_Int8/Abs_14: Omitted lower saturation */ if (Sh1_InPort_14 >= 0) { Sh1_OutPort_14 = C_I8SHRI32C6_LT24_SATu(Sh1_InPort_14, 23, 1065353216); } else { C_I64NEGI32(Sh1_InPort_14, Aux_S32, Aux_U32); Sh1_OutPort_14 = C_I8SHRI64C6_LT32_SATu(Aux_S32, Aux_U32, 23, 9, 0, 1065353216); } </pre>

Reason Code efficiency

Reducing variable scopes In previous TargetLink versions, variables whose addresses were used in the initialization of other variables were not reduced in scope.

In TargetLink 5.1, the Code Generator can reduce variables whose addresses are used in the initialization of other variables down to static local, if the initialization allows for it. For example:

- Reuse substructures
- Axes and map in a look-up table map structure

If referenced and referencing variables are created on the same level, i. e., in the same file, function, or compound statement in the function body, the variables

are defined in relation to each other. This means, that the variables might be moved further upwards or change their relative order if necessary.

Reason MISRA C compliance, Matches user expectations.

Migration issue To avoid the scope reduction of variables whose addresses are used in the initialization of other variables, you can use one of the following:

- Disable the AvoidStaticLocalScope Code Generator option.
- Reference a DD VariableClass object whose Optimization property does not contain SCOPE_REDUCIBLE.

More inlining in conditionally compiled code

TargetLink now estimates the costs of functions that are encapsulated by preprocessor directives such as #if in a better way. This results in more inlined function in the context of conditional compilation. In other contexts, less inlining might occur in rare cases.

Reason Code efficiency

Optimizing read accesses of non erasable symbols

TargetLink can now optimize ineffectual read access of symbols with a DD VariableClass object whose Optimization property is not set to ERASABLE. While the symbols cannot be removed, some of their read access may be removed. *Ineffectual* read access do not affect the result of a computation, usually due to constant operands that determine the value of an expression.

Additionally, read access of macros with a DD VariableClass object whose Optimization property is not set to ERASABLE can now be removed only if they are constant. Otherwise they are considered to have possible side-effects. This can prevent optimizations that were performed in previous versions of TargetLink.

Examples Consider the following examples:

```
x = a * 0;
y = b || 1;
z = c && 0;
```

a, b, and c do not contribute to the value of x, y, or z, respectively, and can be removed. Note that for logical AND and OR operations, the operand order determines whether an access is ineffectual or unreachable.

```
Aux = MY_EXTERNAL_MACRO;

if (Value > 1)
{
    Output = Aux * 2.;
} else {
    Output = Aux * 0.5;
}
```

Given the externally defined macro MY_EXTERNAL_MACRO, the accesses to the variable Aux in the if and else branches will not be replaced by MY_EXTERNAL_MACRO, because MY_EXTERNAL_MACRO is neither constant nor has a variable class with the ERASABLE property set.

Reason

- Matches user expectations
- Code efficiency

Migration issue To prevent the optimized read access, you have to enable the `AssumePossibleSideEffectsForReadAccessToNonErasableVariables` Code Generator option.

No Stateflow machine data variables optimization

TargetLink no longer optimizes Stateflow® machine data variables because they can be accessed anywhere in the respective model and are not limited to one CGU.

Reason Bug fix

Initialization

Optimization of initial and constant values

When optimizing initial values or constant values, TargetLink now preserves data type information by generating a cast that precedes the propagated value. This ensures that optimized and unoptimized code behaves in the same way.

Take this code as an example:

```
UInt32 var = 1;
if ( var << 31 >= 2 ) {
//...
}
```

The following table shows the optimized code:

TargetLink ≤ 5.0	TargetLink 5.1
<pre>if (1 << 31 >= 2) { //... }</pre>	<pre>if (((UInt32) 1) << 31 >= 2) { //... }</pre>

No cast is generated in the following contexts:

- Direct assignments
- Function parameters
- Function return values
- Macros (`OmitCastOfMacroValue` property set to `true`)
- The replaced variable has a floating point type.

The following side effects might rarely occur:

- Casts are omitted for constants that have special semantics.
- Additional comments for constant folding might be generated.

Reason Bug fix

Initial value for floating point variables specified at block and in Data Dictionary

The code changed, if all of the following conditions apply:

- The block references a a DD Variable object whose Type property references a floating point data type.
- An initial value is specified both at the block and at the DD Variable object.
- The initial values specified at the block and at the DD Variable differ within allowed tolerance (epsilon comparison).

TargetLink ≤ 5.0	TargetLink 5.1
The initial value was chosen from Simulink or from the Data Dictionary.	The initial value of the DD Variable object is used.

Reason

- Bug fix
- Increased consistency

Initialization code for Assignment blocks in iterated subsystems

The TargetLink code pattern for the initialization code of Assignment blocks in iterated subsystems changed. In rare circumstances, the old cold pattern could result in false code. With TargetLink 5.1 the initialization is now performed within a conditional control flow that tests an explicit variable (`FirstRun`).

This can result in the following changes:

- The suffixes in the names of local variables might change during optimization.
- The `FirstRun` variable remains in production code, if any of the following conditions holds:
 - The Assignment block resides in a conditionally executed subsystem.
 - The code generated for the Assignment block lies in a branch of a conditionally executed control flow.
- In while-iterator subsystems, the statement order changes: the initialization of the assignment is now placed between the initialization of the loop variable and the begin of the `while` or `do` loop.

The following code example shows the `FirstRun` variable that now remains in the code:

TargetLink ≤ 5.0

```
for (Sa3_For_Iterator_it = 1; Sa3_For_Iterator_it <= Aux_; Sa3_For_Iterator_it++)
{
  if (Sa3_For_Iterator_it == 1) {
    for (Aux_a = 0; Aux_a < VvwY0PropOn; Aux_a++)
    {
      /* Assignment: Subsystem/For/Y: output initialization */
      Sa3_Y[Aux_a] = DD_Sa3_Y0[Aux_a];
    }
  }
  //...
}
```

TargetLink 5.1

```

Sa3_Y_FirstRun = 1;

for (Sa3_For_Iterator_it = 1; Sa3_For_Iterator_it <= Aux_; Sa3_For_Iterator_it++)
{
    if (Sa3_Y_FirstRun > 0) {
        for (Aux_a = 0; Aux_a < VvwY0PropOn; Aux_a++)
        {
            /* Assignment: Subsystem/For/Y: output initialization */
            Sa3_Y[Aux_a] = DD_Sa3_Y0[Aux_a];
        }
        Sa3_Y_FirstRun = 0;
    }

    //...
}

```

Reason Bug fix

Logical Expressions

Optimizing logical expressions

TargetLink now optimizes logical expressions with the type AND or OR that contain operand relations which express identical or inverse relations in arbitrary order. Prior to this, only identically structured expressions were considered as identical relations.

TargetLink ≤ 5.0	TargetLink 5.1
c = x < 5 5 > x	c = x < 5
c = x < 5 x >= 5	c = 1
c = x < 5 && x >= 5	c = 0

Reason Code efficiency

Range Propagation

Improved UInt64 code

The TargetLink worst-case range calculation for unsigned 64-Bit integer code was improved. This can result in better code:

TargetLink ≤ 5.0	TargetLink 5.1
<pre>F_U64MULI32I32(1999999999, 1999999999, &Aux_U32_b, &Aux_U32_c); C_U64ADDU64U64(Aux_U32, Aux_U32_a, Aux_U32_b, Aux_U32_c, Aux_U32, Aux_U32_a); Sa1_OutUpLowSat = (Int16) C_I32FITU64_SATu(Aux_U32, Aux_U32_a, 2147483647);</pre>	<pre>F_U64MULI32I32(1999999999, 1999999999, &Aux_U32_b, &Aux_U32_c); C_U64ADDU64U64(Aux_U32, Aux_U32_a, Aux_U32_b, Aux_U32_c, Aux_U32, Aux_U32_a); Sa1_OutUpLowSat = (Int16) 2147483647;</pre>

Reason Bug fix

Worst-case range propagation

In prior versions, TargetLink sometimes propagated worst-case ranges for variables of global scope or static storage duration. Because, in rare cases, this might have led to false code, propagation of worst-case ranges is now restricted to variables of local scope that do not have static storage duration.

This can result in the following code changes:

- Unnecessary saturations
- Unnecessary comparisons
- Operations calculated in a width greater than actually needed

The following table shows an example:

TargetLink ≤ 5.0	TargetLink 5.1
<pre>b = C_I16SATI16_SATb(Sa1_InPort, 15, -1); c = C_I16SATI16_SATu(b, 1);</pre>	<pre>b = C_I16SATI16_SATb(Sa1_InPort, 15, -1); c = C_I16SATI16_SATb(b, 1, -15);</pre>

In some cases, the resulting code might improve as shown in the following table:

TargetLink ≤ 5.0	TargetLink 5.1
<pre>Sa1_OutPort4 = Sa1_InPort2; Sa1_OutPort4 = Sa1_OutPort4 * 5; Sa1_OutPort4 = (Int16) (((UInt16) Sa1_OutPort4) + 2);</pre>	<pre>Sa1_OutPort4 = Sa1_InPort2; Sa1_OutPort4 = Sa1_OutPort4 * 5; Sa1_OutPort4 = (Int16) (Sa1_OutPort4 + 2);</pre>

Reason Bug fix

Shift Operations

Code pattern for the Shift Arithmetic block TargetLink now uses the shift operation code pattern for the Shift Arithmetic block.

TargetLink ≤ 5.0	TargetLink 5.1
<code>Sa1_U16LShift = Sa1_U16In << 2;</code>	<code>Sa1_U16LShift = (UInt16) (Sa1_U16In << 2);</code>
Reason Improved platform independence of the production code	

Code patterns for shifting numerical values TargetLink now creates special code patterns for shifting numerical values. Closing casts are also inserted if the shift value is a variable.

TargetLink ≤ 5.0	TargetLink 5.1
<code>Ca1_UnscaledI32Tmp = (Int32) (10 << 1);</code>	<code>Ca1_UnscaledI32Tmp = (Int32) (((UInt32) 10) << 1);</code>
Reason	
<ul style="list-style-type: none"> ▪ Improved platform independence of the production code ▪ MISRA C compliance 	

Stateflow and MATLAB Code

Different order of extracting parts from complex expressions

In Stateflow and MATLAB code, TargetLink extracts different parts from complex expressions, e.g., function calls, matrix multiplications, or casts.

Consider the following code as example:

```
out = 4 + f() + 5 + g();
```

TargetLink ≤ 5.0	TargetLink 5.1
Extracting from left to right: <code>aux = g();</code> <code>aux_a = f();</code> <code>out = 4 + aux_a + 5 + aux;</code>	Extracting from right to left: <code>aux = f();</code> <code>aux_a = g();</code> <code>out = 4 + aux + 5 + aux_a;</code>
Reason	

- Matches user expectations
- Increased consistency
- Readability

Improved detection of constant values for Stateflow/MATLAB code

During code generation, TargetLink now detects more constant elements in Stateflow or MATLAB code. This can result in code changes like the following:

- Different or simpler code for MATLAB `for` loops.
- Concatenations might now be implemented as a single vector/matrix assignment using an auxiliary constant instead of several individual assignments.
- More frequent folding of local MATLAB variables, e.g.:

```
localMATLABVar = 5;
out = 4 + localMATLABVar;
```

becomes:

```
out = 9;
```

Reason Code efficiency

Counter variables for Stateflow temp logic more efficient

The counter variable generated for Stateflow constructs such as `after`, `at`, `before`, and `any` is generated differently:

TargetLink ≤ 5.0	TargetLink 5.1
TargetLink always uses <code>Float64</code> (constrained to <code>UInt32</code>) as data type: <pre>if (Ca1_Chart_ctr < 4294967295U) { Ca1_Chart_ctr++; }</pre>	TargetLink determines the data type from the Data Dictionary, e.g., <code>Int16</code> : <pre>if (Ca1_Chart_ctr < 32767) { Ca1_Chart_ctr++; }</pre>

Reason

- Code efficiency
- Precision

For loops in Stateflow

With TargetLink 5.1, the conditions for a For loop generation in Stateflow are changed.

TargetLink ≤ 5.0
<pre>for (Ca1_i = 0; ((Int32) Ca1_i) < (((Int32) Ca1_LoopSize) * 5); Ca1_i = Ca1_i + 1) { Ca1_F64Out = Ca1_F64Out + (Sa1_InPort1 + 42); }</pre>

TargetLink 5.1
<pre>Ca1_i = 0; while (((Int32) Ca1_i) < (((Int32) Ca1_LoopSize) * 5)) { Ca1_F64Out = Ca1_F64Out + (Sa1_InPort1 + 42); Ca1_i = (Int16) (Ca1_i + 1); }</pre>

Reason Bug fix

Migration issue For the changed conditions, refer to [For Loop \(TargetLink Preparation and Simulation Guide\)](#).

Binary bitwise operations in Stateflow

TargetLink now considers the TargetLink code generation target on binary bitwise operations in Stateflow®, i. e., TargetLink changes **U16Var** and **I16Var** to **Int32** if the code generation target is 32 bits.

TargetLink ≤ 5.0	TargetLink 5.1
<pre>if ((UInt16) (Ca1_t1I16Var Ca1_t1U16Var) > 0) { ...</pre>	<pre>if ((Int32) (((Int32) Ca1_t1I16Var) ((Int32) Ca1_t1U16Var)) > 0) { ...</pre>

Reason Bug fix

Migration issue When performing MIL-SIL comparisons, make sure you specify the same properties for **int** datatypes in Simulinks model configuration parameters on the **Hardware Implementation** page. This results in target-specific code. If generic code is required, avoid binary bitwise on 16 bit operands with mixed signedness. Remember that bitwise operations with signed operands are a MISRA-violation anyhow.

Variable Optimization

Optimized vector and matrix assignments

TargetLink now optimizes iterated vector and matrix access where the assigning loop copies a part of a larger vector or matrix variable to the intermediate variable.

TargetLink ≤ 5.0	TargetLink 5.1
<pre> for (Aux_ = 0; Aux_ < 5; Aux_++) { /* Selector: Sample/Selector */ Sa1_Selector[Aux_] = Sa1_In1[Aux_ + 0]; } /* Rescaler: Sample/Data Type Conversion */ Sa1_Data_Type_Conversion = Sa1_In2 != 0; for (Aux_a = 0; Aux_a < 5; Aux_a++) { /* Logical: Sample/Logical Operator */ Sa1_Logical_Operator[Aux_a] = (Sa1_Selector[Aux_a]) && Sa1_Data_Type_Conversion; } for (Aux_b = 0; Aux_b < 5; Aux_b++) { /* TargetLink outport: Sample/Out1 */ Sa1_Out1[Aux_b] = Sa1_Logical_Operator[Aux_b]; } </pre>	<pre> Sa1_Data_Type_Conversion = Sa1_In2 != 0; for (Aux_S32 = 0; Aux_S32 < 5; Aux_S32++) { /* TargetLink outport: Sample/Out1 # combined # Logical: Sample/Logical Operator # combined # Sum: Sample/Sum # combined # Selector: Sample/Selector */ Sa1_Out1[Aux_S32] = (Sa1_In1[Aux_S32]) && Sa1_Data_Type_Conversion; } </pre>

Reason Bug fix

Variable Scope

Changed initial scope

TargetLink now sets the initial scope to **global** for the following variables:

- Custom Code parameter variables
- Custom Code work variables

Subsequent optimization usually reduces the scope to the same or even lower scope as in previous TargetLink versions.

Reason Bug fix

Other

Removing includes from tlDefines_a.h

TargetLink now removes includes of **tlDefines_a.h** if the header file only defines variable class macros that are not used in production code.

Reason Bug fix

Dynamically assigning new values to data store variables

TargetLink now can dynamically assign new values to data store variables.

In these situations, the code patterns are changed as follows:

- Without code optimization

TargetLink ≤ 5.0	TargetLink 5.1
<pre> for (Aux_a = 0; Aux_a < 10; Aux_a++) { /* Assignment: Subsystem/Run/Subsystem/Subsystem/Assignment: output initialization */ Sa3_Assignment[Aux_a] = Sa3_Data_Store_Read[Aux_a]; } /* Assignment: Subsystem/Run/Subsystem/Subsystem/Assignment: output calculation */ Sa3_Assignment[Sa3_Rescaler1 - 1] = Sa3_Rescaler; for (Aux_b = 0; Aux_b < 10; Aux_b++) { /* Data store write: Subsystem/Run/Subsystem/Data Store Write */ Sa2_Data_Store_Memory[Aux_b] = Sa5_Assignment[Aux_b]; } </pre>	<pre> /* Assignment: Subsystem/Run/Subsystem/Assignment: output calculation */ Sa3_Assignment[Sa3_Rescaler1 - 1] = Sa3_Rescaler; /* Data store write: Subsystem/Run/Subsystem/Data Store Write */ Sa2_Data_Store_Memory[Sa3_Rescaler1 - 1] = Sa3_Assignment[Sa3_Rescaler1 - 1]; </pre>

- With code optimization

TargetLink ≤ 5.0	TargetLink 5.1
<pre> /* SLStaticLocal: Default storage class for static Local variables Width: 16 */ static SInt16 Sa4_Assignment[10] = { /* [0..9] */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 /* 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. */ }; for (Sa3_For_Iterator_it = 1; Sa3_For_Iterator_it <= 3; Sa3_For_Iterator_it++) { /* Assignment: Subsystem/Run/Subsystem/Subsystem/Assignment: output calculation # combined # Rescaler: Subsystem/Run/Subsystem/Subsystem/Rescaler */ Sa4_Assignment[Sa3_For_Iterator_it - 1] = 5; for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { /* Data store write: Subsystem/Run/Subsystem/Data Store Write */ Sa2_Data_Store_Memory[Aux_S32] = Sa4_Assignment[Aux_S32]; } } </pre>	<pre> for (Sa3_For_Iterator_it = 1; Sa3_For_Iterator_it <= 3; Sa3_For_Iterator_it++) { /* Data store write: Subsystem/Run/Subsystem/Data Store Write # combined # Assignment: Subsystem/Run/Subsystem/Assignment: output calculation # combined # Rescaler: Subsystem/Run/Subsystem/Subsystem/Rescaler */ Sa2_Data_Store_Memory[Sa3_For_Iterator_it - 1] = 5; } </pre>

Reason

- Code efficiency
- Code size

Migration issue To dynamically assign new values to data store variables, refer to [Basics on Dynamically Assigning New Values to Data Store Variables](#) ([TargetLink Preparation and Simulation Guide](#)).

Changed scope reduction of structure components

TargetLink now does not reduce the scope of structure components if any access to one of the parent structures of the component is located in the same function.

TargetLink ≤ 5.0	TargetLink 5.1
Aux_S16 = 3; S4 = S3;	S3.b1i = 3; S4 = S3;

Reason Bug fix

Code Changes Between TargetLink 4.4 and TargetLink 5.0

Where to go from here	Information in this section
	Shift Operations..... 456
	Extern C Encapsulation..... 459
	Access Functions..... 459
	AUTOSAR..... 460
	Global Bitfields..... 461
	MATLAB Code..... 461
	Name Template of DD VariableTemplate Objects..... 462
	Look-Up Tables..... 462
	Variable Vector Width..... 463
	Stateflow..... 464
	Other..... 466

Shift Operations

Suppression of implicit shift operations	For shift operations that are not specified explicitly, the behavior of TargetLink changed. The associated ShiftMode Code Generator option was replaced by the UtilizeBitwiseShiftOperations Code Generator option that has different semantics and a different default behavior. Signed operands are no longer left-shifted by default. This leads to changes to the production code as multiplications are used more often. These changes are described below. Reason Compliance with C99. Migration issue For recommended compatibility settings for the new Code Generator option, refer to Migration Aspects Regarding Code Generator Options on page 304.
Multiplications with power-of-two values	Multiplications with power-of-two values are often used for rescaling operations. For signed operands, multiplications with power-of-two values are now performed as multiplications instead of left shifts by default. This is shown in the following example:

Operation:

```
Int16 I16Out; Lsb = 2^-1
Int16 I16In; Lsb = 2^0
I16Out = I16In;
```

Result:

TargetLink ≤ 4.4	TargetLink 5.0
I16Out = I16In << 1;	I16Out = I16In * 2;

Note

Most compilers implement multiplications with power-of-two values as efficient as shift operations.

Saturation

The saturation of shift operations is implemented with macros. The saturation of multiplications with a numerical value is implemented as control flows. With the new default behavior, the replacement of shift operations with multiplications therefore leads to the replacement of macro calls with control flow expressions. This is shown in the following example:

TargetLink ≤ 4.4	TargetLink 5.0
HOLD_VALUE = C_I16SHLI16C6_SATb(Sb15_Switch3, 5, 1023, -1024);	if (Sb15_Switch3 > 1023) { HOLD_VALUE = 32767; } else { if (Sb15_Switch3 < -1024) { HOLD_VALUE = -32768; } else { HOLD_VALUE = Sb15_Switch3 * 32; } }

Casts

Casts for operations with left shifts are different from casts for multiplications. This leads to changes in the production code as left shifts of signed operands are now avoided by default:

TargetLink ≤ 4.4	TargetLink 5.0
I16Out = (Int16) (I16Var1 << 1);	I16Out = I16Var1 * 2;

The missing final cast does not interfere with the MISRA C-compliance of the production code.

Signed operands and unsigned result

Expressions with a signed operand and unsigned results, e.g., code patterns for the Abs block, can change because of the different default behavior:

TargetLink 5.0
if (In >= 0) { OutAbs = (UInt8) (((UInt16) In) * 2); } else { OutAbs = (UInt8) (((UInt16) (-In)) << 1);}

In case of the `then` branch, the multiplication pattern casts the signed operand to unsigned. By default, TargetLink does not implement a shift in this expression because the signed operand is detected.

In case of the `else` branch, shifting is possible because the result of the unary minus operation already has an unsigned type.

Additions and subtractions with the same shift factor

Under certain conditions, additions and subtractions with the same shift factor were implemented by first performing the addition/subtraction and then shifting. This is different for multiplications that by default now partly replace shift operations:

Operation:

```
out = in1 << 1 + in2 << 1
```

Result:

TargetLink ≤ 4.4	TargetLink 5.0
out = (in1 + in2) << 1	out = in1 * 2 + in2 * 2

Constant folding

The change in default behavior can lead to more constant foldings because multiplications can be folded by TargetLink. Under certain conditions, overflows will be computed directly:

Operation	Result
I16Var = (Int16)(32767 << 1);	I16Var = -2;
I16Var = (Int16)(-32768 << 1);	I16Var = 0;

Operand eliminations

Shift operations that eliminate operands can now be replaced by multiplications that do not eliminate operands as shown in the following example:

Operation:

```
U16Var = I16Var * 65536
```

Result:

TargetLink ≤ 4.4	TargetLink 5.0
U16Var = 0;	U16Var = I16Var * 65536

Extern C Encapsulation

Production code now contains encapsulation of extern C:

TargetLink ≤ 4.4	TargetLink 5.0
-	<pre>#ifdef __cplusplus extern "C" { #endif ... // TargetLink-generated-C-Code #ifndef __cplusplus } #endif</pre>

Reason Lets you use TargetLink-generated production code in C++ contexts.

Migration issue You can control this behavior via the EmitEncapsulationByExternC Code Generator option.

Access Functions

Access functions containing only read accesses

The code pattern changed for access functions that contain only read accesses:

- Scalar variables: TargetLink now generates auxiliary variables to buffer the value more often if the number of accesses is greater than two
- Vector and matrix variables: TargetLink now generates an auxiliary variable to buffer the value only if the whole variable is accessed at least once
- If you force TargetLink to use auxiliary variables by setting the CreateLocalValueCopy property to **always**, the following pattern is used:
Preceding each read access of an element of the variable, TargetLink now uses an access function call to write the respective element.

Reason Code efficiency, Increased consistency

Migration issue None

Access functions specified as ADDRESS or AsAddressParameter

Access functions whose PassVariable property is set to **AsAddressParameter** now use the TargetLink base type.

TargetLink ≤ 4.4	TargetLink 5.0
MyInt16 * GetMyAfArray3Var3(MyArrayV_3 * varptr)	Int16 * GetMyAfArray3Var3(Int16 * varptr)

In calls of RTE API functions of classic AUTOSAR that are generated via access function specifications, the application data type is used:

TargetLink ≤ 4.4	TargetLink 5.0
<code>sint32 * Rte_Pim_SHZ_A4_Matrix(void)</code>	<code>MyInt32 * Rte_Pim_SHZ_A4_Matrix(void)</code>
Reason	Increased consistency
Migration issue	None

AUTOSAR

AUTOSAR 2.x/3.x no longer supported

TargetLink no longer supports code generation for AUTOSAR 2.x/3.x. After you changed the value of the DD /Pool/Autosar/Config property from 2.x/3.x to >= 4.0, the following code changes occur:

Evaluation of \$(Prm)

TargetLink ≤ 4.4	TargetLink 5.0
<code>Rte_Calprm...()</code>	<code>Rte_Prm...()</code>

Changed default for function class of runnables For DD Runnable objects that do not reference a DD FunctionClass object, the default function class has changed. This can result in a different order of declarations or definitions.

Additionally, there are new instructions for memory mapping:

TargetLink ≤ 4.4	TargetLink 5.0
-	<pre>#define Swc_Start_Sec_Code #include Swc_MemMap.h ... #define Swc_Stop_Sec_Code #include Swc_MemMap.h</pre>

And changes in compiler abstraction:

TargetLink ≤ 4.4	TargetLink 5.0
<code>FUNC(void, RTE_APPL_CODE) Run(void)</code>	<code>FUNC(void, Swc_CODE) Run(void)</code>

Removed Ptr2ArrayBasetypePassing macro

TargetLink ≤ 4.4	TargetLink 5.0
<code>#ifndef RTE_PTR2ARRAYBASETYPE_PASSING #define RTE_PTR2ARRAYBASETYPE_PASSING 1 #endif</code>	-

Reason Removal of code generation for AUTOSAR 2.x/3.x

Migration issue Refer to [AUTOSAR 2.x/3.x no Longer Supported](#) on page 307.

Global Bitfields

Boolean instead of UInt8 Boolean variables (`UseGlobalBitfields = on`) or Stateflow states and flags (`StateflowUseBitfields = on`) that are collected in global bitfields are now optimized differently:

TargetLink ≤ 4.4	TargetLink 5.0
Function local variables are removed from the global bitfield as <code>UInt8</code>	Function local variables are removed from the global bitfield as <code>Boolean</code>

This can result in a different number of casts, depending on the context.

Reason Bug fix

Migration issue None

Inheritance of data type The data type of block variables might change if all of the following conditions are fulfilled:

- The `UseGlobalBitfields` Code Generator option is set to `on`
- The block's `Inherit` properties checkbox is selected
- The block has a variable class other than `default`
- The block output is scalar
- The block is driven by a block whose variable class is set to `default` and whose data type is `Bool`

TargetLink ≤ 4.4	TargetLink 5.0
The block variable inherits the data type <code>Bitfield</code>	The block variable inherits the data type <code>Bool</code>

This can also affect auxiliary variables that are generated in this context as well as their casts.

Reason Increased consistency

Migration issue None

MATLAB Code

Mapping of Simulink data types When generating code for unspecified local MATLAB variables, TargetLink now determines the data type automatically. This can lead to type changes.

Migration issue You can control data type mapping via the following Code generator options:

- [AutoSelectIntegerTypesForLocalMATLABVariables](#) ([TargetLink Model Element Reference](#))
- [AutoSelectBoolTypesForLocalMATLABVariables](#) ([TargetLink Model Element Reference](#))

- [AutoSelectFloatTypesForLocalMATLABVariables](#) (TargetLink Model Element Reference)

Name Template of DD VariableTemplate Objects

The default value of the NameTemplate property of DD VariableTemplate objects changed for global interface variables that are specified as follows:

- DD VariableKind property set to `GlobalInterfaceVar` and DD VariableClassSpec property set to `SLGlobal`.
- DD VariableKind property set to `GlobalInterfaceVar` and DD VariableClassSpec property set to `SLGlobalInit`.

TargetLink ≤ 4.4	TargetLink 5.0
<code>IF_\${\$}_\$L</code>	<code>IF_\${\$}_\$B</code>

This also affects implicitly generated struct components of global bitfield structures and reuse structures.

Additionally, in this context, `$L` is now always evaluated as `$B`.

Reason Increased consistency

Migration issue None

Look-Up Tables

Look-up table functions and table maps Table functions and table maps now always use the TargetLink base type.

Reason Increased consistency

Migration issue None

Overflows of Direct Look-Up Table blocks If the Action for out-of-range input property of Direct Look-Up Table (n-D) blocks is set to `Warning` or `Error`, TargetLink instruments the generated code to report overflows in SIL and PIL simulations. This could lead to compiler errors in the following circumstances:

- When compiling the production code outside of TargetLink
- The `Global logging option` option was set to `Do not log anything` and the `Clean code` checkbox was cleared

With TargetLink 5.0, this has been fixed, which leads to code changes.

The generated macros were renamed:

TargetLink ≤ 4.4	TargetLink 5.0
ERR_LUT_INDEX_OVERFLOW	ERR_LUT_IDX_BEYOND_UPPER_LIMIT
ERR_LUT_INDEX_UNDERFLOW	ERR_LUT_IDX_BEYOND_LOWER_LIMIT
WARN_LUT_INDEX_OVERFLOW	WARN_LUT_IDX_BEYOND_UPPER_LIMIT
WARN_LUT_INDEX_UNDERFLOW	WARN_LUT_IDX_BEYOND_LOWER_LIMIT

Additionally, this results in changes in `tlDefines.h`:

Change	Example
<code>tlDefines.h</code> can now contain empty macro definitions that are encapsulated by preprocessor directives.	<pre>#ifndef TL_FRAME #ifndef WARN_LUT_IDX_BEYOND_LOWER_LIMIT #define WARN_LUT_IDX_BEYOND_LOWER_LIMIT(param_0) #endif /* WARN_LUT_IDX_BEYOND_LOWER_LIMIT */ #endif /* TL_FRAME */</pre> <pre>#ifndef TL_FRAME #ifndef WARN_LUT_IDX_BEYOND_UPPER_LIMIT #define WARN_LUT_IDX_BEYOND_UPPER_LIMIT(param_0) #endif /* WARN_LUT_IDX_BEYOND_UPPER_LIMIT */ #endif /* TL_FRAME */</pre>
<code>tlDefines.h</code> can now contain an encapsulated include of the simulation frame header file.	<pre>#ifdef TL_FRAME #include "TL_Controller_frm.h" #endif</pre>

Reason Bug fix

Migration issue None

Variable Vector Width

Names and order of auxiliary variables

The name and order of auxiliary variables can change in the context of variable vector width (VVW). This can be the case if these auxiliary variables are used in restart functions that are used to initialize VVV variables.

TargetLink ≤ 4.4	TargetLink 5.0
<pre>q_BasicColorVWUDState[Aux_S32] = Temp_[Aux_S32]; r_BasicColorVWUDState[Aux_S32] = Temp__a[Aux_S32]; m_LightColorVWUDState[Aux_S32] = Temp__b[Aux_S32];</pre>	<pre>q_BasicColorVWUDState[Aux_S32] = Temp__b[Aux_S32]; r_BasicColorVWUDState[Aux_S32] = Temp__c[Aux_S32]; m_LightColorVWUDState[Aux_S32] = Temp__h[Aux_S32];</pre>

Reason Bug fix

Migration issue None

Stateflow

Removal of UseSfDataAsStateFlags Code Generator option

With prior TargetLink versions you were able to use user-defined Stateflow data variables as state activity variables, if the UseSfDataAsStateFlags Code Generator option was enabled. These user defined state activity variables were called *user state flags*. With the removal of the UseSfDataAsStateFlags Code Generator option, user state flags are no longer supported. Instead, TargetLink generates an implicit state activity variable, as shown in the following example:

TargetLink ≤ 4.4	TargetLink 5.0
<pre>/* Begin execution of state Subsystem/Chart.StateA */ switch (MyUSF_StateA) { case 1: { ... } case 2: { ... } default: { break; } }</pre>	<pre>/* Begin execution of state Subsystem/Chart.StateA */ switch (SIBFS_Chart_a.Ca1_Chart_ns) { case StateA1_id: { ... } case StateA2_id: { ... } default: { break; } }</pre>

If your model contains a Stateflow data variable that was used as a user state flag, TargetLink displays message W21039.

Reason Resolves differences in MIL/SIL/PIL simulation modes

Replaced by support of Stateflow active state data with the `Child activity` activity type.

Migration issue As a replacement, you can use active state data with child activity mode. Refer to [Basics on Working with Active State Data](#) ( [TargetLink Preparation and Simulation Guide](#)).

State activity output and signal merging

In Stateflow you can output an active state data to Simulink via a port. If this port is connected to a Merge block, this might lead to false code. With TargetLink 5.0, in this context, the variable generated for the active state data is no longer used as a state activity variable:

TargetLink ≤ 4.4	TargetLink 5.0
<pre> ... switch (Ca1_AMode) { case AModeType_A1: { /* Begin execution of state TL_Root/Chart.A.A1 */ if (...) { /* State transition from TL_Root/Chart.A.A1 to TL_Root/Chart.A.A2 */ Ca1_AMode = AModeType_A2; Sa1_Merge_A = Ca1_AMode; } else { ... } /* End execution of state TL_Root/Chart.A.A1 */ break; } case AModeType_A2: { /* Begin execution of state TL_Root/Chart.A.A2 */ if (...) { /* State transition from TL_Root/Chart.A.A2 to TL_Root/Chart.A.A1 */ Ca1_AMode = AModeType_A1; Sa1_Merge_A = Ca1_AMode; } else { ... } /* End execution of state TL_Root/Chart.A.A2 */ break; } default: { break; } } ... </pre>	<pre> ... switch (SIBFS_Chart_a.Ca2_A_ns) { case Ca3_A1_id: { /* Begin execution of state TL_Root/Chart.A.A1 */ if (...) { /* State transition from TL_Root/Chart.A.A1 to TL_Root/Chart.A.A2 */ SIBFS_Chart_a.Ca2_A_ns = Ca4_A2_id; Ca1_AMode = AModeType_A2; Sa1_Merge_A = Ca1_AMode; } else { ... } /* End execution of state TL_Root/Chart.A.A1 */ break; } case Ca4_A2_id: { /* Begin execution of state TL_Root/Chart.A.A2 */ if (...) { /* State transition from TL_Root/Chart.A.A2 to TL_Root/Chart.A.A1 */ SIBFS_Chart_a.Ca2_A_ns = Ca3_A1_id; Ca1_AMode = AModeType_A1; Sa1_Merge_A = Ca1_AMode; } else { ... } /* End execution of state TL_Root/Chart.A.A2 */ break; } default: { break; } } ...</pre>

Reason Bug fix**Migration issue** None

Removal of superfluous consecutive assignments	Unoptimized Stateflow code could contain superfluous consecutive assignments to variables that are internally used by TargetLink. With TargetLink 5.0, only the last assignment is generated.
---	---

TargetLink ≤ 4.4	TargetLink 5.0
SIBFS_Chart_a.Aux_Ca1_sflag0 = 0; SIBFS_Chart_a.Aux_Ca1_sflag0 = 1;	SIBFS_Chart_a.Aux_Ca1_sflag0 = 1;

Reason Suppress superfluous code**Migration issue** None

Other

Changed default value of UseName property

The default value of the DD UseName property of DD VariableClass objects is now off. This affects only the following:

- DD VariableClass objects that you create in existing Data Dictionaries
- All the DD VariableClass objects in the preconfigured DD templates

TargetLink ≤ 4.4	TargetLink 5.0
<pre>/***\n CAL: global calibratable parameters (ROM) Width: 16\n*****/\nCAL Int16 Sa1_Constant = 1;</pre>	<pre>/***\n CAL: global calibratable parameters (ROM) Width: 16\n*****/\nconst volatile Int16 Sa1_Constant = 1;</pre>

Additionally, includes might be placed in C files that were previously placed in H files.

Reason The macros generated for VariableClass objects are placed in the CGU-specific tlDefines header. Because variable classes are often used across several CGUs, this leads to multiple macro definitions. This includes the risk that the same variable class name is associated with different properties, leading to different macro definitions which is a C language constraint violation.

Migration issue None

Initialization of subsystem in ports and out ports

If a InPort or OutPort block of a subsystem references a DD Variable object that provides an initial value and whose Scope property is set to `value_param`, `ref_param`, or `fcn_return`, the initialization of the actual parameter in the production code changed:

TargetLink ≤ 4.4	TargetLink 5.0
<p>The actual parameter was initialized only in the following cases:</p> <ul style="list-style-type: none"> ▪ The code was generated for the OutPort block of conditionally executed subsystem. ▪ The code was generated for an InPort block that had one of the following Simulink checkboxes selected: <ul style="list-style-type: none"> ▪ Latch input by delaying the outside signal ▪ Latch input for feedback signals of function-call subsystem outputs 	<p>The actual parameter is now always initialized with the initialization value that is specified in the Data Dictionary.</p>

Reason Bug fix

Migration issue To revert to the old behavior, remove the initial value that is specified in the Data Dictionary.

Scope reduction of matrix variables

During optimization, the scope and storage duration of matrix variables is reduced earlier. This can result in a different order of optimizations.

Additionally, the following changes are possible:

- Fewer casts
- More inlining
- Auxiliary variables lose their suffix (Aux_ instead of Aux_b)

Reason Code efficiency

Migration issue None

Function inlining

During code generation, TargetLink now uses a different strategy to inline functions. This is done to generate code that maximizes the number of functions of the same size:

- Earlier inlining of Stateflow functions.
- Inlining of functions that contribute to indirect recursion (Stateflow and MATLAB code)

This can lead to the following code changes:

- Additional casts in compute-through-overflow-calculations.
- Functions that were inlined by previous TargetLink versions are no longer inlined.
- Functions that were not inlined by previous TargetLink versions now are inlined.
- Variables are eliminated or reduced to function-local-scope. Assignments are moved into conditionally executed control flows.
- Variables that were reduced to function-local-scope in previous TargetLink versions are no longer reduced in scope.

Reason Increased consistency, Code efficiency

Migration issue None

Suffixes of auxiliary variables

In the context of inlined functions, the suffixes of auxiliary variables might change.

Reason Increased consistency

Migration issue None

Active variant config in file header comment

In previous TargetLink versions, the file header comment did not contain the active variant config if the `cconfig.xml` file contained the following lines:

```
<TL:header-comment show="true" show-in-non-root-files="true">
  ...
  <TL:active-variant show="true"/>
  ...
  <TL:option-list-comment show="false"/>
  ...
</TL:header-comment>
```

With TargetLink 5.0 this works as expected:

TargetLink ≤ 4.4

```
/************************************************************************/
***  

*** Simulink model      : online_parameter_modification  

*** TargetLink subsystem : online_parameter_modification/LissajousFigure  

*** Codefile            : LissajousFigure.c  

***  

*** Generated by TargetLink, the dSPACE production quality code generator  

***  

*** SUBSYS              CORRESPONDING SIMULINK SUBSYSTEM  

*** Sa1                 LissajousFigure  

***  

*** SUBSYS              CORRESPONDING MODEL BLOCK (REFERENCED MODEL)  

***  

*** SF-NODE             CORRESPONDING STATEFLOW NODE           DESCRIPTION  

***  

\*****
```

TargetLink 5.0

```
/************************************************************************/
***  

*** Simulink model      : online_parameter_modification  

*** TargetLink subsystem : online_parameter_modification/LissajousFigure  

*** Codefile            : LissajousFigure.c  

***  

*** Generated by TargetLink, the dSPACE production quality code generator  

***  

*** Active Variant Config : VC1  

***  

*** SUBSYS              CORRESPONDING SIMULINK SUBSYSTEM  

*** Sa1                 LissajousFigure  

***  

*** SUBSYS              CORRESPONDING MODEL BLOCK (REFERENCED MODEL)  

***  

*** SF-NODE             CORRESPONDING STATEFLOW NODE           DESCRIPTION  

***  

\*****
```

Reason Matches user expectations

Migration issue None

Different order of function declarations in <cgus>_fri.h

In the simulation code of previous TargetLink versions, the <cgus>_fri.h file could contain double declarations of the following kind of functions:

- AUTOSAR restart functions
- AUTOSAR terminate functions
- Access functions

With TargetLink 5.0 this was fixed. This can result in a different order of function declarations in the `<cgus>_fri.h` file (simulation code):

TargetLink ≤ 4.4	TargetLink 5.0
<pre>extern void TerminateRun(void); extern void RESTART_SWC1_Run1(void); extern void Subsystem(void); ... extern void Task_RestartRun(void); extern void RestartRun(void); extern void TerminateRun(void); extern void GetSa2_In1(sint16 * valueptr); extern void SetSa2_Out1(const sint16 * valueptr); #endif /* SUBSYSTEM_FRI_H */</pre>	<pre>extern void GetSa2_In1(sint16 * valueptr); extern void RestartRun(void); extern void SetSa2_Out1(const sint16 * valueptr); extern void TerminateRun(void); extern void RESTART_SWC1_Run1(void); extern void Subsystem(void); ... extern void Task_RestartRun(void); #endif /* SUBSYSTEM_FRI_H */</pre>

Additionally, for the same kind of functions, function class comments or block declaration statements could be missing in production code. This also has been fixed:

TargetLink ≤ 4.4
<pre>FUNC(void, FuelsysController_CODE) FuelsysControllerInit(void);</pre>
TargetLink 5.0
<pre>/* start of memory section 'CODE' */ #define FuelsysController_START_SEC_CODE #include "FuelsysController_MemMap.h" /*** ARRunnable4: Default function class for AUTOSAR runnables ***/ FUNC(void, FuelsysController_CODE) FuelsysControllerInit(void); /* end of memory section 'CODE' */ #define FuelsysController_STOP_SEC_CODE #include "FuelsysController_MemMap.h"</pre>

Reason Bug fix

Migration issue None

Code Changes Between TargetLink 4.3 and TargetLink 4.4

Where to go from here

Information in this section

Access Functions.....	470
AUTOSAR.....	473
Block-Specific Code Changes.....	473
Efficiency.....	478
Inheritance of Struct Data Types.....	479
Mixed Operations (Floating-Point and Fixed-Point Types).....	481
Stateflow.....	482
Other.....	483

Access Functions

Names of auxiliary variables for access functions and NvData communication

For certain access-function-related code patterns, TargetLink generates variables of global scope and names them according to this naming schema:
Aux_<OriginalVariableName>_I\$R. TargetLink 4.4, generates fewer auxiliary variables of global scope and uses the following naming schema
Aux_<OriginalVariableName>\$R:

TargetLink ≤ 4.3	TargetLink 4.4
MyType Aux_OriginalVariableName_a;	MyType Aux_OriginalVariableName;

Additionally, TargetLink 4.4 sometimes generates auxiliary variables for vectors as **_tmpu_<Originalvariablename>\$R** instead of **_tmpd_<Originalvariablename>\$R**.

Reason Code efficiency

Migration issue None

Initialization of auxiliary variables

The code changed in contexts where auxiliary variables are used for encapsulation. The auxiliary variables now more frequently have the following characteristics:

- Their storage duration is static.
- They are initialized in the same way as the encapsulated variable.

This code difference is primarily visible in unoptimized code.

Reason Code efficiency

Migration issue None

Optimization of variables with restart initialization or access function template

The code changed for user-specified scope reduction chains whose DD VariableClass objects have different settings regarding the ERASABLE value of their Optimization property. This inconsistent specification can result in the addition or removal of dead code in the form of:

- Restart initializations
- Assignments to variables that are never used

Note

This inconsistency can occur only if you reference a DD VariableClass object whose ScopeReducedClass property is set, because the Optimization property of all internal default variable classes is set to ERASABLE and MOVABLE.

Example Consider a variable `x` that has the variable class `G` with the scope-reduced class `M`. They are specified as follows:

Property	G	M
Scope	global	global
Storage	default	static
Optimization	ERASABLE, SCOPE_REDUCIBLE	SCOPE_REDUCIBLE
ScopeReducedClass	M	-
RestartFunctionName	default	default

The unoptimized code looks like this:

```
Int16 x;
void RestartFoo(void) {
    x = 42;
}
void foo(void) {
    if (cond) {
        x = 17;
    } else {
        x = 35;
    }
    o = x;
}
```

TargetLink ≤ 4.3	TargetLink 4.4
<pre>void foo(void) { if (cond) { o = 17; } else { o = 35; } }</pre>	<pre>static Int16 x; void RestartFoo(void) { x = 42; } void foo(void) { if (cond) { o = 17; } else { o = 35; } }</pre>

If you switch the ERASABLE value between M and G, this results in the opposite behavior.

If the UseName property of either G or M is set to on, a macro definition might be added to or removed from `t1Defines_$.h`.

Reason Increased consistency

Migration issue None

Reduction of the dimension of access-function-related auxiliary variables

The code has changed for access-function-related auxiliary variables in the context of dimension reduction:

TargetLink ≤ 4.3	TargetLink 4.4
Vector and matrix variables with an automatic storage duration might be substituted by a scalar variable:	
<pre>Int16 _tmpd_B[10]; for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { _tmpd_B[Aux_S32] = GetA (Aux_S32); SetBIndexed(Aux_S32, _tmpd_B[Aux_S32]); out1[Aux_S32][index] = _tmpd_B[Aux_S32] + 1; }</pre>	<pre>Int16 Aux_S16; for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { Aux_S16 = GetA (Aux_S32); SetBIndexed(Aux_S32, Aux_S16); out1[Aux_S32][index] = Aux_S16 + 1; }</pre>
Vector and matrix variables that were reduced to a scalar might now be static and initialized:	
<pre>Int16 Aux_S16; for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { Aux_S16 = GetA (index, Aux_S32); SetBIndexed(Aux_S32, index, Aux_S16); out1[Aux_S32][Aux_S32_a] = Aux_S16 + 1; }</pre>	<pre>static Int16 _tmpd_B[10][10] = {...}; for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++) { _tmpd_B[Aux_S32][index] = GetA (index, Aux_S32); SetBIndexed(Aux_S32, index, _tmpd_B[Aux_S32][index]); out1[Aux_S32][index2] = _tmpd_B[Aux_S32][index] + 1; }</pre>

Reason Correct run-time behavior

Migration issue None

AUTOSAR

Loops in production code for NvData communication with optimized writing

For optimized writing in NvData communication via Data Store Write blocks, TargetLink respects the LoopUnrollThreshold Code Generator option for generating production code with loops. This enables the generation of rolled code.

Reason Code size

Migration issue None

Improved scheduling of pointer initialization

The placement of return pointers of AUTOSAR RTE API functions with read access was improved for the following conditions :

- The RTE API function returning the pointer is called in another function.
- The corresponding RTE API function with write access is called in another function.

TargetLink ≤ 4.3	TargetLink 4.4
<pre>const sint32 * p_IRV_Vector_W5; /* Data store read: Subsystem/SWC/Run2/Read_IRV_Vector_W5 */ p_IRV_Vector_W5 = Rte_IrvIRead_Run2_IRV_Vector_W5(); ... X_Store = p_IRV_Vector_W5[CurrIndex];</pre>	<pre>const sint32 * p_IRV_Vector_W5; ... /* Data store read: Subsystem/SWC/Run2/Read_IRV_Vector_W5 */ p_IRV_Vector_W5 = Rte_IrvIRead_Run2_IRV_Vector_W5(); X_Store = p_IRV_Vector_W5[CurrIndex];</pre>

Reason Readability, matches user expectations

Migration issue None

Block-Specific Code Changes

Sqrt block

In some cases, TargetLink used to generate an unreachable else branch in the production code for the Sqrt block. This occurred in the following cases, for example:

- The Generate optimized code Code Generator option was disabled.
- The global input variable referenced a VariableClass object whose Optimization property was not set to ERASABLE

This is no longer the case.

TargetLink ≤ 4.3	TargetLink 4.4
<pre> if (in <= 0.) { out = 0; } else { if (in >= 0.) { out = sqrt(in); } else { out = -sqrt(-in); } } </pre>	<pre> if (in < 0.) { out = 0.; } else { out = sqrt(in); } </pre>

Furthermore, this change results in `out = sqrt(in);` for positive only ranges (including zero) of the input signal and `out = 0;` for negative only ranges.

Reason MISRA C compliance, code efficiency

Migration issue None

Delay block

Changes in the evaluation of ranges lead to more efficient code for the Delay block.

TargetLink ≤ 4.3
<pre> TlAuxVar_a = ((UInt8) (((Int16) (((Int16) X_Sa2_Delay_Index) + ((Int16) (30 - ((UInt16) (((UInt16) TlAuxVar) << 1)))))) % 30); </pre>
TargetLink 4.4
<pre> TlAuxVar_a = ((UInt8) (X_Sa2_Delay_Index + ((UInt8) (30 - ((UInt8) (TlAuxVar << 1)))))) % 30; </pre>

Additionally, the auxiliary calculation for the pointer initialization has been moved into the conditionally executed control flow branch.

Unoptimized
<pre> Aux_U8 = ((UInt8) (X_Sa2_Delay_Index + ((UInt8) (6 - ((UInt8) (C_U8SATI16_SATb(Sa1_DelayLength, 2, 1) * 3)))))) % 6; pAuxPtr_a = &(X_Delay[Aux_U8]); if (Sa1_Enable > 0) { ... pAuxPtr_a[0] ... } </pre>

Optimized TargetLink ≤ 4.3

```
Aux_U8 = ((UInt8) (X_Sa2_Delay_Index + ((UInt8) (6 - ((UInt8) (C__U8SATI16_SATb(Sa1_DelayLength,
2, 1) * 3)))))) % 6;
if (Sa1_Enable > 0) {
    pAuxPtr_a = &(X_Delay[Aux_U8]);
    ... pAuxPtr_a[0] ...
}
```

Optimized TargetLink 4.4

```
if (Sa1_Enable > 0) {
    Aux_U8 = Aux_U8 = ((UInt8) (X_Sa2_Delay_Index + ((UInt8) (6 - ((UInt8) (C__U8SATI16_SATb(Sa1_DelayLength,
2, 1) * 3)))))) % 6;
    pAuxPtr_a = &(X_Delay[Aux_U8]);
    ... pAuxPtr_a[0] ...
}
```

Reason Code efficiency**Migration issue** None**Direct Look-Up Table (n-D)
block**

The following code patterns of the Direct Look-Up Table (n-D) block have changed with TargetLink 4.4 :

- The loop control variable is now named `Aux_*` instead of `i*` , as is the case for other blocks.
- The `LoopUnrollThreshold` is respected, i.e., the generation of unrolled code is now possible.
- The loop control variable is always defined in the first section of a function.

Reason Increased consistency**Migration issue** None**Discrete-Time Integrator
block**

Saturation of the code generated from the Discrete-Time Integrator block changed. For the `Trapezoidal` integration method , TargetLink now saturates both additions contained in the underlying formula:

TargetLink ≤ 4.3

```

/* Discrete Integrator: Subsystem/Discrete-Time Integrator */
if (Sa1_Subsystem_FirstRun) {
    /* Discrete Integrator: first run initialization */
    TlAuxVar = 0;
}
else {
    /* Discrete Integrator: integration */
    TlAuxVar =
        ((Int32) X_Sa1_Discrete_Time_Integrator) + ((Int32) (((Int32) Sa1_InPort) +
            (Int32) U_Sa1_Discrete_Time_Integrator)));
}

/* Discrete Integrator: saturation */
X_Sa1_Discrete_Time_Integrator =
    C_I8SATI32_SATb(TlAuxVar, 100 /* 50. */, -100 /* -50. */);

/* Discrete Integrator: Subsystem/Discrete-Time Integrator */
Sa1_Discrete_Time_Integrator =
    (Int16)X_Sa1_Discrete_Time_Integrator;

```

TargetLink 4.4

```

/* Discrete Integrator: Subsystem/Discrete-Time Integrator */
if (Sa1_Subsystem_FirstRun) {
    /* Discrete Integrator: first run initialization */
    TlAuxVar_b = 0;
}
else {
    /* Discrete Integrator: integration */
    TlAuxVar = ((Int32) X_Sa1_Discrete_Time_Integrator) +
        (Int32) U_Sa1_Discrete_Time_Integrator;

    /* Discrete Integrator: saturation */
    TlAuxVar_a = C_I32SATI32_SATb(TlAuxVar, 100 /* 50. */, -100 /* -50. */);
    TlAuxVar_b = TlAuxVar_a + ((Int32) Sa1_InPort);
}

/* Discrete Integrator: saturation */
X_Sa1_Discrete_Time_Integrator =
    C_I8SATI32_SATb(TlAuxVar_b, 100 /* 50. */, -100 /* -50. */);

```

Reason Resolves differences in MIL/SIL/PIL simulation modes, matches user expectations

Migration issue None

Math block (pow function)

TargetLink's code pattern for the initial code of the Math block is different, if all of the following conditions are fulfilled:

- The block is specified to implement the **pow** function.
- The block is driven by Constant blocks that do not specify a signal or variable.

TargetLink ≤ 4.3

```

if (( in1 < 0) && (in2 != floor(in2)))
{
    out = -(pow(-in1, in2));
}
else
{
    out = pow(in1, in2);
}

```

TargetLink 4.4

TargetLink now evaluates the conditions `(in1 < 0)` and `(in2 != floor(in2))` when generating the initial code. If `in1` and/or `in2` are constants, either of the two `if` conditions or the whole if-else statement might not be generated.

Reason Code efficiency

Migration issue None

Math block (mod/rem function)

The code of the Math block is different, if all the following conditions are fulfilled:

- The block is specified to implement the `mod` or `rem` function.
- At least one of the block's inputs or its result has a floating-point type.

TargetLink ≤ 4.3

Code for `mod` Function

```

Sa1_I16_F32F64_ = (Int16) (((Float64) Sa1_F32NumInInt) -
(Sa1_F64DenomInInt *
((Float64) (Int32) (((Float64) Sa1_F32NumInInt) /
Sa1_F64DenomInInt))));

if (Sa1_I16_F32F64_ != 0) {
    if (((Sa1_F32NumInInt < 0.F) &&
(Sa1_F64DenomInInt >= 0.)) ||
((Sa1_F32NumInInt >= 0.F) &&
(Sa1_F64DenomInInt < 0.))) {
        Sa1_I16_F32F64_ += ((Int16) Sa1_F64DenomInInt);
    }
}

```

Code for `rem` Function

```

Sa1_F64_F32I16_ = ((Float64) Sa1_F32NumIn) - (((Float64)
Sa1_I16DenomIn) *
((Float64) (Int32) (((Float64) Sa1_F32NumIn) / ((Float64)
Sa1_I16DenomIn))));

```

TargetLink 4.4

```

Sa1_I16_F32F64_ = (Int16) (((Float64)
Sa1_F32NumInInt) -
(Sa1_F64DenomInInt * floor(((Float64)
Sa1_F32NumInInt) / Sa1_F64DenomInInt)));

```

```

Aux_F64 = ((Float64) Sa1_F32NumIn) / ((Float64)
Sa1_I16DenomIn);

if (Aux_F64 > 0.) {
    Aux_F64 = floor(Aux_F64);
}
else {
    Aux_F64 = ceil(Aux_F64);
}

Sa1_F64_F32I16_ = ((Float64) Sa1_F32NumIn) -
((Float64) Sa1_I16DenomIn) *
Aux_F64;

```

Reason Resolves differences in MIL/SIL/PIL simulation modes, matches user expectations

Migration issue None

Efficiency

Implicit variables in min/max operations

TargetLink used superfluous implicit variables in unoptimized code for min/max operations if all of the following was true:

- The operands had different data types.
- The result type was different from the data type resulting from the worst-case ranges of the operands.
- One of the operands had the same data type as the worst-case range.

This is no longer the case.

TargetLink ≤ 4.3	TargetLink 4.4
<pre>Int16 Aux_, Aux_a; Aux_ = Ca1_I16In; Aux_a = (Int16) Ca1_U8In; if (Aux_ > Aux_a) { Ca1_I80ut = (Int8)Aux_; } else { Ca1_I80ut = (Int8)Aux_a; } Aux_S32 = (Int32) Sa1_UInt16In; if (((Int32) Sa1_Int16In) < Aux_S32) { Sa1_I16MinI16U16_ = Sa1_Int16In; } else { Sa1_I16MinI16U16_ = (Int16) Aux_S32; }</pre>	<pre>Int16 Aux_; Aux_ = (Int16) Ca1_U8In; if (Ca1_I16In > Aux_) { Ca1_I80ut = (Int8)Ca1_I16In; } else { Ca1_I80ut = (Int8)Aux_; } if (((Int32) Sa1_Int16In) < ((Int32) Sa1_UInt16In)){ Sa1_I16MinI16U16_ = Sa1_Int16In; } else { Sa1_I16MinI16U16_ = (Int16) Sa1_UInt16In; }</pre>

Reason Code efficiency

Migration issue None

Result ranges

TargetLink 4.4 takes the available range information for a multiplication into account in more cases. Therefore, a smaller data type might be used.

TargetLink ≤ 4.3
<pre>F_I64MULI32U32(SActRet1_Rescaler, (tUI32) x_FunctionCoordinationFuncFact_ldu16, &Aux_lds32, &Aux_ldu32); C_I16DIVI64U32(Aux_lds32, Aux_ldu32, (tUI32) 1024, x_ActiveRetFuncFactor_lds16); Sa1_out_2 = (Int16) (UInt8) (SReferenced1_OutPort3 * ((UInt8) MyScalarDDMacro));</pre>

TargetLink 4.4

```
x_ActiveRetFuncFactor_lds16 = (tSI16) ((SActRet1_Rescaler * ((tSI32)
x_FunctionCoordinationFuncFact_ldu16)) / 1024);
Sa1_out_2 = (Int16) (((Int8) SReferenced1_OutPort3) * ((Int8) MyScalarDDMacro));
```

If possible, TargetLink performs the multiplication with the same signedness as the result type. This can lead to different casts during the multiplication:

TargetLink ≤ 4.3

```
Aux_U16_b = (UInt16) (Int16) (((Int16) Aux_U8) * Aux_S16_b) >> 1);
```

TargetLink 4.4

```
Aux_U16_b = (UInt16) (((UInt16) (((UInt16) Aux_U8) * ((UInt16) Aux_S16_b))) >> 1);
```

Reason Code efficiency

Migration issue None

Addition with constants

The code has changed for Sum blocks whose List of signs property is set to +- and whose first inport is driven by a constant:

TargetLink ≤ 4.3	TargetLink 4.4
Out = -In1 + 17;	Out = 17 - In1;

Reason Code efficiency

Migration issue None

Inheritance of Struct Data Types

SPI blocks

In TargetLink 4.4, a block can inherit a struct data type from a preceding block if the following conditions are met:

- The block's Inherit properties checkbox is selected.
- The preceding block references a root struct DD variable.

This was already the case for block variables referencing one of the following DD objects:

- a DD Typedef object whose BaseType property is set to **Struct**
- the DD Typedef object **IMPLICIT_STRUCT**.

However, this did not apply to structured variables.

The resulting code then contains:

TargetLink ≤ 4.3	TargetLink 4.4
Multiple non-struct variables: One for each leaf signal of the inheriting block.	One struct variable for the inheriting block.

Reason Increased consistency, code efficiency

Migration issue None

Ports of atomic subsystems

An unenhanced import or outport of an atomic subsystem can inherit a struct data type from a preceding block in the following cases.

- A bus signal is applied to the port.
- The preceding block references a DD Variable object that is specified as a formal function parameter passing a pointer to a struct.

This was already the case for block variables referencing one of the following DD objects:

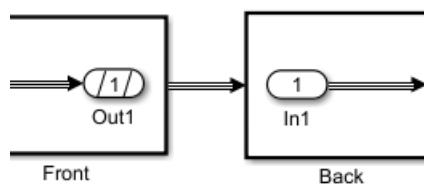
- a DD Typedef object whose BaseType property is set to Struct
- the DD Typedef object IMPLICIT_STRUCT.

However, this did not apply to structured variables.

The unenhanced port is then implemented as follows:

TargetLink ≤ 4.3	TargetLink 4.4
Implemented as multiple global non-struct variables.	Implemented as a function parameter of the atomic subsystem with type 'pointer to struct'.

Example Consider the following modeling situation:



The described changes lead to the following differences in the code for this model:

TargetLink ≤ 4.3	TargetLink 4.4
<pre> StructType DDStruct; Int16 Sa2_Out1; Int16 Sa2_Out1_a; void Sa2_Back(void) { Sa2_Out1 = DDStruct.a; Sa2_Out1_a = DDStruct.b; } void Code(void) { Sa3_Front(&DDStruct); Sa2_Back(); [...] } </pre>	<pre> StructType DDStruct; void Sa2_Back(StructType * Sa2_In1, StructType * Sa2_Out1) { *Sa2_Out1 = *Sa2_In1; } void Code(void) { StructType Sa2_Out1; Sa3_Front(&DDStruct); Sa2_Back(&DDStruct, &Sa2_Out1); [...] } </pre>

Remarks This code change can occur if you used a Bus Outport block at the interface of an incremental code generation unit and if the following conditions are fulfilled:

- One of the Bus Outport block's formal parameters passes a pointer to a struct.
- This formal parameter can be specified via a DD VariableClass object whose Scope property is set to `ref_param`.

With TargetLink ≤ 4.3, the successor block of the unenhanced Import block reads from the structured actual parameter that was generated for the Bus Outport block. This is shown in lines 6 and 7 in the table above. The `Sa2_Back` function does not have any parameter. This is shown in line 12.

With TargetLink 4.4, the unenhanced Import block is implemented as a formal pointer to struct parameter. This parameter has the same structured actual parameter as the Bus Outport block. Optimization can possibly reduce the parameter's scope to `local`. This is shown in lines 4 and 10.

Reason Increased consistency, code efficiency

Migration issue None

Mixed Operations (Floating-Point and Fixed-Point Types)

Binary floating-point operations

Binary floating-point operations with a floating-point result, a floating-point operand, and an integer or fixed-point operand are now calculated in the largest occurring floating-point type. This does not apply to additions and subtractions resulting from a Sum block.

TargetLink ≤ 4.3

```
Sa1_F64Out = (Float64) (((Float64) Sa1_I16In) * Sa1_F32In);
```

TargetLink 4.4

```
Sa1_F64Out = ((Float64) Sa1_I16In) * ((Float64) Sa1_F32In);
```

Reason MISRA C compliance**Migration issue** None

Stateflow

Log macro calling

TargetLink 4.4 calls log macros later. This results in improved optimization possibilities, including merged loop code resulting from vector or matrix allocations as shown in the following pseudo-code:

TargetLink ≤ 4.3	TargetLink 4.4
<pre>for() { out[...] = test[...] +1; } LOG_VEC(out); for() { merge[...] = out[...]; } LOG_VEC(merge);</pre>	<pre>for() { out[...] = test[...] +1; merge[...] = out[...]; } LOG_VEC(out); LOG_VEC(merge);</pre>

Reason Improved loop merging can lead to better optimization.**Migration issue** None**Floating-point expressions cast to integers**

Index expressions that contain floating-point operands that are explicitly cast to integers by the user are now always calculated as floating-points. Consider the code for the Stateflow expression `inC[int16(inC[0]/2*2.11)]`:

TargetLink ≤ 4.3 p1	TargetLink 4.4
<code>Ca1_inC[(Int16) ((Ca1_inC[0] / 2) * 2 /* 2.11 */)]</code>	<code>Ca1_inC[(Int16) (((Float32) (Int32) (Ca1_inC[0] >> 1)) * 2.11F)]</code>

Reason Bug fix**Migration issue** None**Auxiliary variables for struct parameters**

Generally, TargetLink generates auxiliary variables for input and output parameters of graphical functions in Stateflow charts that are not scalar or complex.

This was not the case when all of the following conditions were fulfilled:

- An Input or Output parameter was a bus.
- The function parameter was specified via a structured variable in the Data Dictionary.
- The function parameter was not called by reference.

In this case, TargetLink 4.4 now generates auxiliary variables as follows:

For a graphical function `F(StructFormal)` with the input parameter `StructFormal` of the type 'Bus' that is specified by the `DD_StructFormal` DD Variable object whose Scope property is set to anything but `ref_param`, *unoptimized* code looks like this:

TargetLink ≤ 4.3	TargetLink 4.4
<pre>StructFormal. ... = StructActual. ...; ... StructFormal. ... = StructActual. ...; F();</pre>	<pre>Aux_StructFormal. ... = StructActual. ...; ... Aux_StructFormal. ... = StructActual. ...; StructFormal. ... = Aux_StructFormal. ...; ... StructFormal. ... = Aux_StructFormal. ...; F();</pre>

Reason Bug fix

Migration issue None

Values of state IDs for Stateflow states

TargetLink now sorts the state IDs of Stateflow states by their Ca number:

TargetLink ≤ 4.3	TargetLink 4.4
<pre>#define Ca5_A_id 4 #define Ca6_B_id 3</pre>	<pre>#define Ca5_A_id 3 #define Ca6_B_id 4</pre>

Reason Bug fix, Increased consistency

Migration issue None

Other

Changed order of function parameters

The order of function parameters might be changed if one parameter of the function is removed during optimization.

This is known to occur for the following functions:

- Step functions of reused subsystems or charts that do not have a reuse-instance-structure as a parameter.

Reason Bug fix – Known Problem Report KPR.2017.08.03.004

Migration issue None

Naming of variable vector width auxiliary variables

TargetLink uses auxiliary variables for the initialization of variables with `ExchangeableWidth` in a restart function. The identifiers of these auxiliary variables have changed:

TargetLink ≤ 4.3	TargetLink 4.4
Temp_a, Temp_c, Temp_e, ...	Temp_, Temp_a, Temp_b, ...

Reason Matches user expectations

Migration issue None

Scalar products with floating-point operands and integer results

Scalar products of floating-point operands with integer results are now calculated in floating-point notation and then cast to integers. Usually, such products result from using Product, FIR Filter, Discrete Filter, Discrete Transfer Fcn and Discrete State-Space blocks.

TargetLink ≤ 4.3

```
Sa1_Product2D2D[3][3] = (Int32) (((Int32) (((Float32) Sa1_Matrix2D[3][0]) * Sa1_Matrix2D[0][3])) +
((Int32) (((Float32) Sa1_Matrix2D[3][1]) * Sa1_Matrix2D[1][3])) +
((Int32) (((Float32) Sa1_Matrix2D[3][2]) * Sa1_Matrix2D[2][3])) +
((Int32) (((Float32) Sa1_Matrix2D[3][3]) * Sa1_Matrix2D[3][3])) +
((Int32) (((Float32) Sa1_Matrix2D[3][4]) * Sa1_Matrix2D[4][3])));
```

TargetLink 4.4

```
Sa1_Product2D2D[3][3] = (Int32) (((((Float32) Sa1_Matrix2D[3][0]) * Sa1_Matrix2D[0][3]) +
(((Float32) Sa1_Matrix2D[3][1]) * Sa1_Matrix2D[1][3]) +
(((Float32) Sa1_Matrix2D[3][2]) * Sa1_Matrix2D[2][3]) +
(((Float32) Sa1_Matrix2D[3][3]) * Sa1_Matrix2D[3][3]) +
(((Float32) Sa1_Matrix2D[3][4]) * Sa1_Matrix2D[4][3]));
```

Reason Precision

Migration issue None

Changes in identifiers

The identifiers of the following code elements might change:

- Pointers and substructs in the context of indirect function reuse.
- StateIDs of Stateflow states

Additionally, the names of auxiliary loop variables changed. When determining their names for initial code, TargetLink now starts with `Aux_` instead of `Aux_S32_a`. During optimization, their names might change.

Reason Increased consistency

Migration issue None

Changes in function inlining

When determining whether to inline a function, TargetLink no longer takes into account simple variable definitions. This can result in the following:

- Change in inlining:
 - Small functions are more frequently inlined.
 - Different order of variable definitions within the function body.

- Change in optimization order:
 - Different order of `#combined#` comments.
 - More auxiliary variables.

Reason Matches user expectations

Migration issue None

**Memory sections for TriCore
17xx/Task32 optimization
module**

TargetLink now generates statements for memory sections that comply with the compiler's manual. Additionally, TargetLink 4.4 no longer automatically generates prefixes for section names, which allows for more customization. This applies to the sections `bss`, `data`, `rom`, and `code`. The following example applies to the `bss` section:

TargetLink ≤ 4.3	TargetLink 4.4
<code>#pragma section nearbss = "near_<name>" #pragma section farbss = "far_<name>"</code>	<code>#pragma section nearbss "<name>" #pragma section farbss "<name>"</code>

Reason Compliance with compiler's manual

Migration issue None

**Removal of suffix from
INT32MIN constants**

The L suffix was removed from INT32MIN constants.

TargetLink ≤ 4.3	TargetLink 4.4
<code>(-2147483647L -1L)</code>	<code>(-2147483647 -1)</code>

Reason Code efficiency

On targets whose long type was a 64bit type, the suffix led to inefficient code because the constant unnecessarily used 64 bits . For all other targets, the suffix was superfluous, because not using it resulted in the same integer length as using it.

Migration issue None

Changes in code comments

Several code comments changed:

- Comments preceding the code of Rescaler blocks now contain the string `Rescaler::`.
- Comments for variables with the `CONST` qualifier no longer contain range information. This can result in additional line breaks.

Reason Increased consistency, readability

Migration issue None

Copy propagation and call by reference/vector/matrix call parameters <ul style="list-style-type: none"> ▪ Call by reference parameters ▪ Vector call parameters ▪ Matrix call parameters <p>TargetLink no longer propagates the following in these kinds of call parameters:</p> <ul style="list-style-type: none"> ▪ Bitfields ▪ Macros ▪ Variables qualified as <code>const</code> or <code>volatile</code> ▪ Variables qualified via a type prefix 	<p>Copy propagation changed for call parameters of the following kinds:</p> <ul style="list-style-type: none"> ▪ Call by reference parameters ▪ Vector call parameters ▪ Matrix call parameters
---	--

TargetLink ≤ 4.3	TargetLink 4.4
<pre> const Type* MyType = Rte_IRead_SWC_MyType(); x = S.bitfield; loop i = 0..9 y[i] = MyType[i]; foo(&x, y/* Type[10] */); </pre>	<pre> 1 const Type* MyType = Rte_IRead_SWC_MyType(); 2 3 x = S.bitfield; 4 5 loop i = 0..9 6 7 y[i] = MyType[i]; 8 9 foo(&x, y/* Type[10] */); </pre> <p>x in line nine is not replaced by <code>S.bitfield</code>, because addresses of bitfields are not supported. y in line nine is not replaced by <code>MyType</code>, because <code>const</code> is not supported in contexts that are not <code>const</code>.</p>

Reason Bug fix

Migration issue If you used type prefixes to make non-compiling code compilable, check if this is still necessary.

In AUTOSAR contexts, check the value of the `/Pool/AUTOSAR/Config.RteUsePointerToConstForNonScalarReturnValues` property.

Code Changes between TargetLink 4.2 and TargetLink 4.3

Where to go from here

Information in this section

64-Bit Multiplication.....	487
AUTOSAR.....	488
Efficiency.....	491
Function Reuse.....	494
MISRA Compliance.....	495
Mixed Operations (Floating-Point and Fixed-Point Types).....	496
State Reset.....	498
Other.....	504

64-Bit Multiplication

Additional functions and macros for 64-bit multiplication

TargetLink's Fixed-Point Library now provides additional functions and macros for 64-bit multiplications that are used if one operand has a width smaller than 32-bit.

TargetLink ≤ 4.2	TargetLink 4.3
This can result in a different parameter order:	
<pre>F__I64MULU32U32((UInt32) UInt16Var, UInt32Var, &Aux_c_hi, &Aux_c_lo);</pre>	<pre>F__I64MULU32U16(UInt32Var, UInt16Var, &Aux_c_hi, &Aux_c_lo);</pre>
Operations are used as arguments: ¹⁾	
<pre>Aux_S32 = ((Int32) Sa1_F32In) << Ca1_ShiftVar; F__I64MULI32I32(Aux_S32, (Int32) Ca1_I16Var, &Aux_S32_a, &Aux_U32);</pre>	<pre>F__I64MULI32I16(((Int32) Sa1_F32In) << Ca1_ShiftVar, Ca1_I16Var, &Aux_S32, &Aux_U32);</pre>
Constant literals can now have smaller fitting data types with casts:	
<pre>F__I64MULI32U32(Int32Var, (UInt32) 25000000, &Aux_hi, &Aux_lo);</pre>	<pre>F__I64MULI32I32(Int32Var, (Int32) 25000000, &Aux_hi, &Aux_lo);</pre>
The data type of the cast is determined by the constant's value. This can result in positive values becoming signed.	
Depending on the setting of the DecimalConstants Code Generator option, they might be generated in hexadecimal format. ¹⁾	
<pre>C__I64MULI32U32(Sa1_Gain[0], (UInt32) 12582912, Aux_S32, Aux_U32);</pre>	<pre>C__I64MULI32I32(Sa1_Gain[0], (Int32) 0xc00000, Aux_S32, Aux_U32);</pre>

¹⁾ This also applies to existing functions.

Reasons

- MISRA C compliance
- Code efficiency

Migration issue None**Accumulator of FIR filter**

The code for calculating the accumulator of FIR filters changed.

TargetLink ≤ 4.2	TargetLink 4.3
<p>Calculation was protected, even without saturation, which sometimes resulted in an additional 64-bit auxiliary variable and a corresponding 64-bit multiplication macro.</p> <pre><code>/* accumulation */ Aux_S32 = 0; Aux_S16 = *(pDelayLine++); Aux_S32_a = *(pCoeff++); F_I64MULI32I32((Int32) Aux_S16, Aux_S32_a, &Aux_S32_b, &Aux_U32); Aux_S32 += ((Int32) Aux_U32); if (pDelayLine > (X_Sa5_FIR_Filter + 4)) { pDelayLine = X_Sa5_FIR_Filter; }</code></pre>	<p>The superfluous macros are no longer generated.</p> <pre><code>/* accumulation */ Aux_S32 = 0; Aux_S32 += (((Int32) *(pDelayLine++)) * *(pCoeff++)); if (pDelayLine > (X_Sa5_FIR_Filter + 4)) { pDelayLine = X_Sa5_FIR_Filter; }</code></pre>

Reason

- MISRA C compliance
- Code efficiency

Migration issue None**AUTOSAR****AUTOSAR compiler abstraction for multiple instantiation**

When generating compiler abstraction macros for the instance handle parameter of runnables, the CONSTP2CONST macro now has AUTOMATIC as second parameter, not RTE_CONST.

TargetLink ≤ 4.2	TargetLink 4.3
<pre><code>FUNC(void, Controller_CODE) controllerRunnable(CONSTP2CONST(Rte_CDS_Controller, RTE_CONST, RTE_CONST) instance)</code></pre>	<pre><code>FUNC(void, Controller_CODE) controllerRunnable(CONSTP2CONST(Rte_CDS_Controller, AUTOMATIC, RTE_CONST) instance)</code></pre>

Reason AUTOSAR compliance**Migration issue** None

AUTOSAR 4.x memory mapping

Preprocessor instructions for memory mapping in AUTOSAR 4.x code are now block statements.

TargetLink ≤ 4.2

```
*****\nARRunnable4: Default function class for AUTOSAR runnables\n*****\n#define Controller_START_SEC_CODE\n#include "Controller_MemMap.h"\nFUNC(void, Controller_CODE) controllerRunnable(Rte_ActivatingEvent_controllerRunnable activation);\n#define Controller_STOP_SEC_CODE\n#include "Controller_MemMap.h"\n\n#define Controller_START_SEC_CODE\n#include "Controller_MemMap.h"\nFUNC(void, Controller_CODE) linearizationRunnable(void);\n#define Controller_STOP_SEC_CODE\n#include "Controller_MemMap.h"\n\n#define Controller_START_SEC_CODE\n#include "Controller_MemMap.h"\nFUNC(void, Controller_CODE) Init_Controller(void);\n#define Controller_STOP_SEC_CODE\n#include "Controller_MemMap.h"
```

TargetLink 4.3

```
#define Controller_START_SEC_CODE\n#include "Controller_MemMap.h"\n*****\nARRunnable4: Default function class for AUTOSAR runnables\n*****\n\nFUNC(void, Controller_CODE) controllerRunnable(Rte_ActivatingEvent_controllerRunnable activation);\n\nFUNC(void, Controller_CODE) linearizationRunnable(void);\n\nFUNC(void, Controller_CODE) Init_Controller(void);\n\n#define Controller_STOP_SEC_CODE\n#include "Controller_MemMap.h"
```

Reason

- Suppress superfluous code
- Readability

Migration issue None**Matrix parameters of runnables and operation calls**

Matrix actual parameters of runnable and operation call functions are now by default generated with non-global scope. This might result in the following changes:

- A parameter of the `Rte_Call` RTE API function, is no longer moved to a per instance memory.
- Different naming because `$R` is evaluated differently.

- Different naming if the local parameter is removed by optimization.
- Range propagation is now possible, which can result in simpler control flow and changes in saturation and arithmetic operations by optimization.

Reason Code efficiency

Migration issue None

#include of Rte_Type.h

TargetLink's behavior for generating an `#include of Rte_Type.h` in header files that contain declarations of runnables has changed:

TargetLink ≤ 4.2	TargetLink 4.3
Always generated the include.	<p>Generates the include to <code>Rte_Type.h</code> only in the following cases:</p> <ul style="list-style-type: none"> ▪ A runnable's declaration uses macros for compiler abstraction, which are defined in <code>Compiler.h</code>. This header file in turn is included in <code>Rte_Type.h</code>. ▪ A parameter uses one of the <code>Rte_Type.h</code> data types. <p>If none of the conditions is fulfilled, TargetLink does not generate the include in the header file. In most cases, the include is generated in the corresponding source file, where it is needed if the runnable's implementation makes use of data types contained in <code>Rte_Type.h</code>, such as <code>Std_ReturnType</code>.</p>

Reasons

- Code efficiency
- Suppress superfluous code

Migration issue None

NvData and reduced write operations

The code generated for DD NvDataElement objects has changed, whose `ReduceWriteOperations` property is set to `on`. This results in better optimization of the `if` statement:

TargetLink ≤ 4.2	TargetLink 4.3
<pre>/* TL-Import */ ptr = Rte_IRead_x(); <SomeCode> a = ... /* TL-Outport */ ptr = Rte_IRead_x(); if (ptr[1] != a) { Rte_IWriteRef_x()[1] = a; }</pre>	<pre>/* TL-Import */ ptr = Rte_IRead_x(); <SomeCode> a = ... /* TL-Outport */ if (ptr[1] != a) { Rte_IWriteRef_x()[1] = a; }</pre>

Reason Code efficiency

Migration issue None

Efficiency

Restart functions generated for external subsystems Restart functions generated for external subsystems are now called at different intervals.

TargetLink ≤ 4.2	TargetLink 4.3
Called the restart function once per instance.	Calls the restart function once per CGU.

Reason Code efficiency

Migration issue None

Improved code optimization for Discrete-Time Integrator block

The code pattern for the block was optimized for some edge cases:

- It can now calculate in a smaller width
- Superfluous saturation code can now be better detected and omitted in the production code

Reason Code efficiency

Migration issue None

Propagation of initial values for matrix variables

TargetLink's optimization now also propagates initialization values of matrix variables.

TargetLink ≤ 4.2

```
static Bool Sa1_DFFMat_NQ[3][5] =
{
    {
        {
            /* [0][0..4] */ 1, 1, 1, 1, 1
            /* 1., 1., 1., 1., 1. */
        },
        {
            /* [1][0..4] */ 1, 1, 1, 1, 1
            /* 1., 1., 1., 1., 1. */
        },
        {
            /* [2][0..4] */ 1, 1, 1, 1, 1
            /* 1., 1., 1., 1., 1. */
        }
    };

    for (Aux_S32 = 0; Aux_S32 < 3; Aux_S32++)
    {
        for (Aux_S32_a = 0; Aux_S32_a < 5; Aux_S32_a++)
        {
            /* Sink: Subsystem/Sink3 */
            Sa1_Sink3[Aux_S32][Aux_S32_a] = Sa1_DFFMat_NQ[Aux_S32][Aux_S32_a];
        }
    }
}
```

TargetLink 4.3

```

for (Aux_S32 = 0; Aux_S32 < 3; Aux_S32++)
{
    for (Aux_S32_a = 0; Aux_S32_a < 5; Aux_S32_a++)
    {
        /* Sink: Subsystem/Sink3 */
        Sa1_Sink3[Aux_S32][Aux_S32_a] = 1;
    }
}

```

Reason Code efficiency**Migration issue** None**Slimmer code for expression +1/expression -1 and casts**

TargetLink's optimization now performs several simplifications:

TargetLink ≤ 4.2	TargetLink 4.3
(x + 1) - 1	x
(1 + x) - 1	
(x - 1) + 1	

For integers, relations with 1 or - 1 are simplified as follows:

x - 1 < y	x <= y
x < y + 1	
x < 1 + y	
x <= y - 1	x < y
x + 1 <= y	
1 + x <= y	
x > y - 1	x >= y
x + 1 > y	
1 + x > y	
x - 1 >= y	x > y
x >= y + 1	
x >= 1 + y	

For index expressions the topmost casts are omitted if not needed:

x[(Int32) (a + 1)]	x[a + 1]
--------------------	----------

Further simplification TargetLink further simplifies expressions like $z = x - 1$ that occur in a context that allows for any of the simplifications shown in the table above. Because TargetLink knows that the right side of the expression will be replaced by an access to x , TargetLink treats the cost of propagating the right side as a cost of an access to x .

TargetLink ≤ 4.2	TargetLink 4.3
<pre> z = x - 1; if (z < y) { m = z; } else { m = y; } </pre>	<pre> z = x - 1; if (z < y) { m = z; } else { m = y; } </pre>

In the example above, the first occurrence of `x - 1` is simplified to `x <= y` in accordance with the first table. This lowers the cost of propagating the right side of `z = x - 1;` into the `if` statement.

Reasons

- Code efficiency
- Readability

Migration issue None

Reset when enabled flags for enabled subsystems

TargetLink no longer generates an unnecessary *reset when enabled* flag (RSWE) if an IF block is placed in an enabled subsystem.

TargetLink ≤ 4.2
<pre> if (Sa1_EnableIn > 0) { /* call of function: TL_Root/Enable */ Sa2_Enable(Sa1_InPort2, Sa1_InPort3, Sa1_InPort1); /* set system state to 'enabled' */ Sa2_RSWE = 1; } else { if (Sa2_RSWE == 1) { /* set system state to 'disabled': TL_Root/Enable */ Sa2_RSWE = 0; /* set block state to 'disabled': If: TL_Root/Enable/ResetIf */ Sa2_ResetIf_LastSystem = 0; /* set block state to 'disabled': If: TL_Root/Enable/Else/Else/ResetIf */ Sa5_ResetIf_LastSystem = 0; } </pre>
TargetLink 4.3
<pre> if (Sa1_EnableIn > 0) { /* call of function: TL_Root/Enable */ Sa2_Enable(Sa1_InPort2, Sa1_InPort3, Sa1_InPort1); } else { /* set system state to 'disabled': TL_Root/Enable */ Sa2_RSWE = 0; /* set block state to 'disabled': If: TL_Root/Enable/ResetIf */ Sa2_ResetIf_LastSystem = 0; /* set block state to 'disabled': If: TL_Root/Enable/Else/Else/ResetIf */ Sa5_ResetIf_LastSystem = 0; } </pre>

Reason Code efficiency

Migration issue None

tlDefines.h TargetLink no longer generates the TL_FX_GROUND macro if it is not needed. This might result in `tlDefines.h` no longer being generated.

Reason Suppress superfluous code

Migration issue None

Multiplication with integer fractions (N/D) The code changed for multiplication with integer fractions ($N/D = 2^z$) now may sometimes be more precise with the same efficiency:

TargetLink ≤ 4.2	TargetLink 4.3
<code>Sa1_OutPort = (Int32) ((Sa1_InPort * 91) >> 15);</code>	<code>Sa1_OutPort = (Int32) ((Sa1_InPort * 2913) >> 20);</code>

Reason Precision

Migration issue None

Function Reuse

Instance-specific variables and sub-reuse structs

With TargetLink 4.3, instance-specific variables now might be placed in the same sub reuse struct. Accordingly, the number of sub reuse struct might change or the name of the sub reuse struct might appear to have changed. This change can result in mixed structures that contain volatile and nonvolatile variables.

The following code shows the definition of the structured data types of the reuse struct that results from a reused subsystem that contains two Gain blocks. One has its Class set to `default`, the other to `GLOBAL`:

TargetLink ≤ 4.2	TargetLink 4.3
<p>Example 1</p> <pre>typedef struct tagISV_Sx1_0_tp { Int16 Sx1_Gain; Int16 Sx2_Sum; } ISV_Sx1_0_tp; typedef struct tagISV_Sx1_1_tp { Int16 Sx1_Gain1; } ISV_Sx1_1_tp; typedef struct tagISV_Sx1_tp { ISV_Sx1_0_tp * pISV_Sx1_0_tp; ISV_Sx1_1_tp * pISV_Sx1_1_tp; } ISV_Sx1_tp;</pre>	<pre>typedef struct tagISV_Sx1_0_tp { Int16 Sx1_Gain; Int16 Sx1_Gain1; Int16 Sx2_Sum; } ISV_Sx1_0_tp; typedef struct tagISV_Sx1_tp { ISV_Sx1_0_tp * pISV_Sx1_0_tp; } ISV_Sx1_tp;</pre>

TargetLink ≤ 4.2	TargetLink 4.3
Example 2	
<pre>typedef struct tagISV_Sx1_1_tp { Int16 Sx1_Gain1; } ISV_Sx1_1_tp; typedef struct tagISV_Sx1_tp { ISV_Sx1_1_tp * pISV_Sx1_1_tp; } ISV_Sx1_tp;</pre>	<pre>typedef struct tagISV_Sx1_0_tp { Int16 Sx1_Gain1; } ISV_Sx1_0_tp; typedef struct tagISV_Sx1_tp { ISV_Sx1_0_tp * pISV_Sx1_0_tp; } ISV_Sx1_tp;</pre>

Reasons

- Readability
- Code size

Migration issue None

MISRA Compliance

Trigger-state-variable

TargetLink no longer generates the state-variable associated with blocks whose trigger port is driven by a constant. The generated code is marked by the `condition for never trigger` code commentary.

TargetLink ≤ 4.2	TargetLink 4.3
Generated a state variable in non-optimized trigger code that was never read and removed by optimization.	Does not generate a state variable for trigger code.

Reasons

- Code efficiency
- MISRA C compliance

Migration issue None

Additional cast in ?-operation for min/max/abs in Stateflow code

Calls of the `abs`, `min`, and `max` functions in Stateflow that do not have a parent assignment are generated with the `?` operator.

TargetLink now casts both of the operands to the result type of the function. The following code example is for an `abs` function:

TargetLink ≤ 4.2	TargetLink 4.3
<code>(Ca14_I16In1 >= 0) ? Ca14_I16In1 : ((UInt16) (-Ca14_I16In1))</code>	<code>(Ca14_I16In1 >= 0) ? ((UInt16) (Ca14_I16In1)) : ((UInt16) (-Ca14_I16In1))</code>

Reason MISRA C compliance**Migration issue** None

Arguments of fixed-point atan2()

The arguments of the fixed-point `atan2()` function changed. The second parameter now always is of `Int32`, the third always `Int16`.

TargetLink ≤ 4.2

```
Aux_S16 = ((Int16) Sa1_YIn_Int8_4) * ((Int16) Sa1_YIn_Int8_4);
F__I16ATAN2I16((Int16) Sa1_XIn_Int8_4, Aux_S16, Sa1_YIn_Int8_4)
```

TargetLink 4.3

```
Aux_S32 = (Int32) (((Int16) Sa1_YIn_Int8_4) * ((Int16) Sa1_YIn_Int8_4));
F__I16ATAN2I16((Int16) Sa1_XIn_Int8_4, Aux_S32, (Int16)Sa1_YIn_Int8_4)
```

Reason MISRA C compliance

Migration issue None

Mixed Operations (Floating-Point and Fixed-Point Types)

Assignments in Stateflow

Certain assignments might change in production code generated from Stateflow.

The following code is for an 8-bit integer variable called `Ca1_in` with `Min = 1` and `Max = 3`:

TargetLink ≤ 4.2	TargetLink 4.3
Assignment of an integer to a floating-point	
Calculation is in integer as long as possible to increase efficiency and precision:	
<code>Var = (Int16) (((Float64) (Int8) Ca1_in) + 1.);</code>	<code>Var = (Int16) (Float64) (Int8) (((Int8) Ca1_in) + 1);</code>
Assignment of Float64 to Float32	
Calculation is in Float64 as long as possible to increase precision:	
<code>F32Var = (Float32)F64Var + (Float32)F64Var2;</code>	<code>F32Var = (Float32)(F64Var + F64Var2);</code>

Reason

- Precision
- Code efficiency

Migration issue None

Non-integral values in Stateflow

TargetLink now treats non-integral values in Stateflow as follows:

- Representable as `Float32` - TargetLink interprets the value as being a `Float32`
- Greater/smaller than the maximum/minimum of `Float32` - TargetLink interprets the value as being a `Float64`
- Cannot be precisely represented in `Float32` - TargetLink uses the `DefaultFloatType` as defined in `TargetConfig.xml`

This can result in code differences in conjunction with the `SupportSinglePrecisionLibraries` Code Generator option.

Additionally, non-integral values in an integer context result in floating-point operations:

TargetLink ≤ 4.2	TargetLink 4.3
Expression: <code>I16UnscaledVar2 = sin(I16UnscaledVar1) + 0.5;</code>	
<code>Ca2_I16UnscaledVar2 = (Int16) (((Int16) (F__I16SINI16(Aux_) >> 14)) + 1)</code>	<code>Ca2_I16UnscaledVar2 = (Int16) (sinf((Float32) Ca2_I16UnscaledVar1) + 0.5F)</code>
Expression: <code>I16UnscaledVar2 = I16UnscaledVar1 + 0.5 + I16UnscaledVar2</code>	
<code>Ca2_I16UnscaledVar2 = (Int16) (((UInt16) (Ca2_I16UnscaledVar1 + 1)) + ((UInt16) Ca2_I16UnscaledVar2))</code>	<code>Ca2_I16UnscaledVar2 = (Int16) (((Float32) Ca2_I16UnscaledVar1) + 0.5F + ((Float32) Ca2_I16UnscaledVar2))</code>

Reason Precision

Migration issue The production code contains floating-point operations.

Solution Do not use non-integral values in pure integer contexts.

Math functions in Stateflow

With TargetLink 4.3, the fixed-point implementation of math functions is called more frequently in an integer context in code generated from Stateflow, instead of float or double.

TargetLink ≤ 4.2
<code>F64Var = (Float64) (Int8) (((Int8) sin(((Float64) I16Var) * 9.5874e-05)) << 3);</code>
TargetLink 4.3
<code>F64Var = (Float64) (Int8) (((Int8) (((Int32) F__I16SINI16(I16Var)) >> 14)) << 3);</code>

Reason Code efficiency

Migration issue None

Additional auxiliary variable for mod() in Stateflow

The production code generated for Stateflow changed with respect to the `mod()` function if it is called with another operation as argument in a floating-point context. The operation in the argument is calculated only once and written to an auxiliary variable.

The following code is generated for `Out = mod(I32In * I32In + F64In, 3)`.

TargetLink ≤ 4.2

```
Out = (Int32) (((Float64) I32In) * ((Float64) I32In)) +
    F64In - (3. * ((Float64) (Int32) (((Float64) I32In) * ((Float64) I32In)) + F64In / 3.));
if (Out != 0) {
    if (((Float64) I32In) * ((Float64) I32In)) + F64In < 0.) {
        Out += 3;
    }
}
```

TargetLink 4.3

```
Aux_ = (((Float64) I32In) * ((Float64) I32In)) + Sa1_In;
Out = (Int32) (Aux_ - (3. * ((Float64) (Int32) (Aux_ / 3.))));
if (Out != 0) {
    if (Aux_ < 0.) {
        Out += 3;
    }
}
```

Reason Code efficiency

Migration issue None

State Reset

State reset

For the call of the step function, a superfluous if statement (see the bold part in the following code example) is now avoided.

TargetLink ≤ 4.2

```
if (Se1_InPort[1] > 0.) {
    if (!(Se3_RSWE)) {
        INIT_Se3_subsystem2();
        Se3_RSWE = 1;
    }

    Se3_subsystem2();
}
else {
    if (Se3_RSWE == 1) {
        Se3_RSWE = 0;
    }
}
```

TargetLink 4.3

```
if (Se1_InPort[1] > 0.) {
    if (!(Se3_RSWE)) {
        INIT_Se3_subsystem2();
        Se3_RSWE = 1;
    }

    Se3_subsystem2();
}
else {
    Se3_RSWE = 0;
}
```

Reason Suppress superfluous code

Migration issue None

Chart triggering a subsystem/chart via a function-call output event bound to a state

A Stateflow chart be configured as follows:

- The chart is located in a conditional subsystem whose States when enabling, States when action is resumed, or States when starting property is set to reset.
- The chart triggers a subsystem or chart via a function-call output event.
- The output event is bound to a state of the triggering chart by means of a bind action.

The code changes are as follows:

- A Boolean RSWE flag for the function-call subsystem or Stateflow chart is generated.
- The RSWE flag is reset after the INIT function of the function-call subsystem or Stateflow chart was called in the state's entry function.
- The INIT function is called at the beginning of the state-during code. This call is encapsulated by a request to the RSWE flag.
- The RSWE flag can also be reset by other callers of the chart function, e.g., if the chart is located in an enabled subsystem.

Tip

By inlining the INIT function and the state's entry function, the (re-)setting of the RSWE flag as well as the state-reset code of the function-call subsystem or Stateflow chart can directly become part of the code.

TargetLink ≤ 4.2	TargetLink 4.3
<pre>void Enabled(void) { if (Sa1_InPort > 0) { if (!(Sa2_RSWE)) { INIT_Sa2_Enabled(); Sa2_RSWE = 1; } Sa2_Enabled(); } else { Sa2_RSWE = 0; } [...] }</pre>	<pre>void Enabled(void) { if (Sa1_InPort > 0) { if (!(Sa2_RSWE)) { INIT_Sa2_Enabled(); Sa2_RSWE = 1; } Sa2_Enabled(); } else { Sa2_RSWE = 0; Sa1_RSWE = 0; } [...] }</pre>

TargetLink ≤ 4.2	TargetLink 4.3
<pre> void Ca2_Calling(void) { if (SIBFS_Calling_a.Ca3_B == 1) { if (Ca2_counter == 5.) { [...] Ca4_A_en(); } [...] } else { if (SIBFS_Calling_a.Ca4_A == 1) { [...] } else { Ca4_A_en(); } } } void Ca4_A_en(void) { [...] INIT_Ca1_Chart(); } </pre>	<pre> void Ca2_Calling(void) { if (SIBFS_Calling_a.Ca3_B == 1) { if (Ca2_counter == 5.) { [...] Ca4_A_en(); } [...] } else { if (SIBFS_Calling_a.Ca4_A == 1) { if (!(Sa1_RSWE)) { INIT_Ca1_Chart(); Sa1_RSWE = 1; } [...] } else { Ca4_A_en(); } } } void Ca4_A_en(void) { [...] INIT_Ca1_Chart(); Sa1_RSWE = 1; } </pre>

Reason Resolves differences in MIL/SIL/PIL simulation modes

Migration issue None

Chart triggering a referenced subsystem via a function-call output event

A Stateflow chart be configured as follows:

- The chart triggers a referenced subsystem via a function-call output event.
- The output event is not bound to any state of the triggering chart.
- For the referenced subsystem, the trigger port's States when enabling property is set to reset.

The code changes are as follows:

- A Boolean RSWE flag for the function-call subsystem is generated.
- The Stateflow chart function contains the following two statements at its beginning, both of which are encapsulated by a request to the RSWE flag:
 1. Call of the function-call subsystem's INIT function
 2. Reset of the RSWE flag

TargetLink ≤ 4.2	TargetLink 4.3
<pre>void Ca1_Chart(void) { /* Begin execution of chart <...> */ ... /* End execution of chart <...> */ }</pre>	<pre>void Ca1_Chart(void) { if (!(SMR1_RSWE)) { /* initialization of subsystem: <...> */ INIT_SMR1(); SMR1_RSWE = 1; } /* Begin execution of chart <...> */ ... /* End execution of chart <...> */ }</pre>

In combination with the code change Init of function-call-triggered referenced models described in [Other](#) on page 483, as well as additional code optimization and function inlining, the code changes can look like this:

<pre>void sr_held_mr_reset_forced(void) { /* SLStaticLocalInit: Default storage class for static Local variables with initialValue Width: 8 */ static Bool Sa2_RSWE = 0; /* sr_held_mr_reset_forced/Subsystem/Enable: Enable condition */ if (Sa1_in_ > 0) { if (!(Sa2_RSWE)) { /* initialization of subsystem: sr_held_mr_reset_forced/Subsystem/Model */ INIT_SMR1(); /* set system state to 'enabled' */ Sa2_RSWE = 1; } ... } else { /* set system state to 'disabled': sr_held_mr_reset_forced/Subsystem */ Sa2_RSWE = 0; } }</pre>	<pre>void sr_held_mr_reset_forced(void) { /* SLStaticLocalInit: Default storage class for static local variables with initialValue Width: 8 */ static Bool SMR1_RSWE = 0; /* sr_held_mr_reset_forced/Subsystem/Enable: Enable condition */ if (Sa1_in_ > 0) { if (!(SMR1_RSWE)) { /* initialization of subsystem: sr_held_mr_reset_forced/Subsystem/Model */ INIT_SMR1(); /* set system state to 'enabled' */ SMR1_RSWE = 1; } ... } else { /* set system state to 'disabled': sr_held_mr_reset_forced/Subsystem */ SMR1_RSWE = 0; } }</pre>
---	---

Reason Resolves differences in MIL/SIL/PIL simulation modes

Migration issue None

State reset for flip flop blocks

The code generated for the following blocks changed:

- D Flip-Flop blocks
- J-K Flip-Flop blocks

TargetLink ≤ 4.2	TargetLink 4.3
Scope changes for <code>J_Previous</code> , <code>K_Previous</code> , and <code>D_Previous</code> variables: The variables had local scope with static storage duration.	The variables have global scope unless otherwise specified.
State reset of the <code>J_Previous</code> , <code>K_Previous</code> , and <code>D_Previous</code> variables: No state reset.	State reset in the <code>INIT</code> function of the parenting subsystem. The <code>INIT</code> function is generated only if the parenting subsystem or one of its callers are configured for state reset. Usually, this function is inlined during optimization. The change in the variable's scope changes the context of the variables. This might cause changes in variable names because the <code>\$R</code> name macro expands differently.

Reason Resolves differences in MIL/SIL/PIL simulation modes

Migration issue None

State reset for actual parameters of latched imports of triggered subsystems	The code for latched import blocks of triggered subsystems changed if their triggered subsystems are parented by a subsystem that is configured for state reset. The state variables are now of global scope.
---	---

TargetLink ≤ 4.2	TargetLink 4.3
State reset of variables: No state reset.	State reset in the <code>INIT</code> function of the parenting subsystem. The <code>INIT</code> function is generated only if the parenting subsystem or one of its callers are configured for state reset. Usually, this function is inlined during optimization. The change in the variable's scope changes the context of the variables. This might cause changes in variable names because the <code>\$R</code> name macro expands differently.

Reason Resolves differences in MIL/SIL/PIL simulation modes

Migration issue None

INIT function for referenced models	The code changed for referenced models that are configured as follows: <ul style="list-style-type: none">▪ Contains a Trigger block with Trigger type set to <code>function-call</code> and States when enabling set to <code>held</code>▪ Contains a Function block at its root level whose <code>forceinitfunction</code> block property is set to <code>on</code>
--	---

TargetLink ≤ 4.2	TargetLink 4.3
Generate an <code>INIT</code> function for a state reset that is never called.	Generate an empty <code>INIT</code> function that is called depending on the modeling of the subsystem that contains the Model block.

Reason Suppress superfluous code

Migration issue To force TargetLink to generate the same `INIT` function as in previous versions, set the Trigger block's States when enabling property to `reset`. TargetLink then generates and calls the `INIT` function that contains the reset code.

Init of function-call triggered referenced models

With the improved support for held/reset the code changed for referenced models, if the following conditions are fulfilled:

- The Model block references a model that contains a Trigger block with Trigger type set to **function-call** and States when enabling set to **reset**.
- The Model block is triggered by a subsystem whose states are hold.

TargetLink ≤ 4.2	TargetLink 4.3
<p>The INIT function generated for the calling subsystem simply called the INIT function of the referenced model.</p> <pre>void Sa2_Subsystem(void) { /* Subsystem/Subsystem/Function-Call Generator call of function: Subsystem/MRSystem1 */ MR_STEP(); } void INIT_Sa2_Subsystem(void) { /* initialization of subsystem: Subsystem/MRSystem1 */ MR_INIT(); }</pre>	<p>The INIT function generated for the referenced model is called in the step function of the calling subsystem. The call of the INIT function is placed before the call of the step function that is generated for the referenced model. Additionally, a RSWE flag is generated to control whether the INIT function is called.</p> <pre>void Sa2_Subsystem(void) { if (!(SMR11_RSWE)) { /* initialization of subsystem: Subsystem/MRSystem1 */ MR_INIT(); SMR11_RSWE = 1; } /* Subsystem/Subsystem/Function-Call Generator call of function: Subsystem/MRSystem1 */ MR_STEP(); }</pre>

Reason Increased consistency

Migration issue None

Unconditionally executed subsystems and function-call-triggered references models

The code has changed for certain subsystems, calling referenced models if the following conditions are fulfilled:

- The calling subsystem is unconditional:
 - It is not enabled.
 - It is not triggered.
 - It is not action-triggered.
 - It is not triggered-and-enabled.
 - It is not iterated.
- The referenced model is specified as follows:
 - Contains a Trigger block with Trigger type set to **function-call** and States when enabling set to **reset**

TargetLink ≤ 4.2	TargetLink 4.3
The INIT function of the referenced model was not called in the generated code. Message W15371 was displayed.	The INIT function of the referenced model now is called in the generated code.

Reason Resolves differences in MIL/SIL/PIL simulation modes

Migration issue None

Related topics**Basics**

[Basics on Resettable Subsystems](#) ([TargetLink Customization and Optimization Guide](#))

Other

Sorting of internal default variable classes

TargetLink sorts internal default variable classes below those loaded from the TargetLink Data Dictionary.

This sorting now also works for the following elements:

- Default variable classes derived with a new Type prefix from an existing one, e.g., in case of indirect function reuse.
- Width-specific DD VariableClass templates.

TargetLink ≤ 4.2

```
/*****************************************************************************\
 SLGlobal: Default storage class for global variables | Width: 64
\*****
Float64 Sa1_InPort;
 5 /* Sr1_Gain_gain */
};

/*****************************************************************************\
 UserSLFcnOutput: SLFcnOutput = { default GLOBAL default default default } | Width: 64
\*****
Float64 Sa1_OutPort; /* Considered default because UserSLFcnOutput is changed to GLOBAL via template only for Width =
Bitfield */
/*****************************************************************************\
 UserSLFcnOutput_a: Derived TL_CG default variable class. | Width: 32
\*****
ISV_Sr1_tp * pISVSr1; /* Considered default because UserSLFcnOutput is changed to GLOBAL via template only for Width =
Bitfield */
```

TargetLink 4.3

```
*****
UserSLFcOutput: SLFcOutput = { GLOBAL } | Width: 64
\*****
Float64 Sa1_OutPort; /* Not considered default because UserSLFcOutput is changed to GLOBAL via template */
\*****
UserSLFcOutput_a: Derived TL_CG default variable class. | Width: N.A.
\*****
ISV_Sr1_tp * pISVSr1; /* Not considered default because UserSLFcOutput is changed to GLOBAL via template */
\*****
SLGlobal: Default storage class for global variables | Width: 64
\*****
Float64 Sa1_InPort;
5 /* Sr1_Gain_gain */
};
```

Reason Increased consistency**Migration issue** None**Changed code comments**

Several of TargetLink's code comments changed:

- Declaration comments for enum and pointer variables now have **Width: N.A.**
- Comments in the code pattern associated with non-scalar interruptable variables, or NvDataElements are sorted differently.
- Comments associated with function headers, code sections, or preprocessor control flow can now contain additional whitespaces.
- Comments for the structured data types of sub reuse structs changed from `/* Description: Reuse linker section structure */` to `/* Description: Reuse substructure */`.
- Comments generated for blocks of declarations and definitions of functions and variables are no longer limited to functions or variables with the same FunctionClass or VariableClass, resulting in fewer, larger blocks.

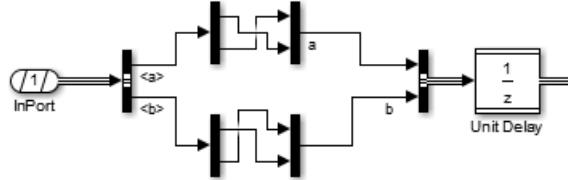
Reasons

- Increased consistency
- Readability

Migration issue None**Inheritance of structured data types**

The inheritance of structured data types changed. For model and code consistency, bus-capable simulation blocks now inherit structured data types only if the bus signal that connects them to the inheriting block is not modified along the signal line.

The following code is for a Unit Delay block that is contained in a model that looks like this:



TargetLink ≤ 4.2	TargetLink 4.3
<pre>struct BS_IP_Sa1_InPort Sa1_InPort; struct BS_IP_Sa1_InPort Sa1_OutPort; struct BS_IP_Sa1_InPort X_Sa1_Unit_Delay = { { 0, 0 }, { 0, 0 } }; void TL_Root(void) { /* BusOutput: TL_Root/OutPort */ Sa1_OutPort = X_Sa1_Unit_Delay; /* Unit delay: TL_Root/Unit Delay */ X_Sa1_Unit_Delay.Sa1_a[0] = Sa1_InPort.Sa1_a[1]; X_Sa1_Unit_Delay.Sa1_a[1] = Sa1_InPort.Sa1_a[0]; X_Sa1_Unit_Delay.Sa1_b[0] = Sa1_InPort.Sa1_b[1]; X_Sa1_Unit_Delay.Sa1_b[1] = Sa1_InPort.Sa1_b[0]; }</pre>	<pre>struct BS_IP_Sa1_InPort Sa1_InPort; struct BS_IP_Sa1_InPort Sa1_OutPort; Int16 X_Sa1_Unit_Delay[2] = { 0, 0 }; Int16 X_Sa1_Unit_Delay_a[2] = { 0, 0 }; void TL_Root(void) { /* BusOutput: TL_Root/OutPort */ Sa1_OutPort.Sa1_a[0] = X_Sa1_Unit_Delay[0]; Sa1_OutPort.Sa1_a[1] = X_Sa1_Unit_Delay[1]; Sa1_OutPort.Sa1_b[0] = X_Sa1_Unit_Delay_a[0]; Sa1_OutPort.Sa1_b[1] = X_Sa1_Unit_Delay_a[1]; /* Unit delay: TL_Root/Unit Delay */ X_Sa1_Unit_Delay[0] = Sa1_InPort.Sa1_a[1]; X_Sa1_Unit_Delay[1] = Sa1_InPort.Sa1_a[0]; X_Sa1_Unit_Delay_a[0] = Sa1_InPort.Sa1_b[1]; X_Sa1_Unit_Delay_a[1] = Sa1_InPort.Sa1_b[0]; }</pre>

Reason Bug fix

Migration issue In TargetLink 4.2, the bus signal could be modified as follows without impeding inheritance:

- Changing the order of bus elements via Selector, Demux, or Mux blocks
- Transposing matrices via Permute Dimensions blocks

This is no longer possible in TargetLink 4.3. If you need inheritance, adjust your models accordingly.

Solution Explicitly specify a structured data type at the inheriting block.

Sorting of variable definitions

TargetLink now sorts the definitions of certain variables more consistently. This applies to the following contexts:

- Variables associated with variable classes that were derived from internal default variable classes
- Type prefixes
- TOMs

- Initialized variables that are relevant for the variable vector width and are declared between preprocessor directives.
- Scope reduction of variables to function-local scope that initially had a variable class with empty ScopeReducedClass property

Reason Increased consistency

Migration issue None

Code generated from model-based CGUs	The code generated from modules whose <code>ModuleInfo.CodeGenerationBasis</code> property is set to <code>ModelBased</code> changed, if the CGU from which the code was generated is not the specific owner of the module:
---	---

TargetLink ≤ 4.2	TargetLink 4.3
The module was treated as if its <code>ModuleInfo.CodeGenerationBasis</code> property was set to <code>Model1AndDDBased</code> : it contained DD-based code that was not used in the model by the current CGU.	The model is now treated as if its <code>ModuleInfo.CodeGenerationBasis</code> property is set to <code>ModelBased</code> : it now only contains code belonging to the model. DD-based code that is not used in the model by the current CGU is no longer generated in the module.

Reason Matches user expectations

Migration issue To generate code that matches the code of prior TargetLink versions, set the module's `ModuleInfo.CodeGenerationBasis` property to `Model1AndDDBased`.

VariableTemplates with VariableKind GlobalInterfaceVar	DD <code>VariableTemplate</code> objects whose <code>VariableKind</code> property is set to <code>GlobalInterfaceVar</code> and whose <code>WidthSpec</code> property is not set or empty now also apply to implicitly generated interface variables with a structured data type.
---	---

The following code example shows an implicitly generated global interface variable. It results from a variable template that is specified as follows:

- `Filter.VariableKind = GlobalInterfaceVar`
- `Filter.WidthSpec = <value not set>`
- `Filter.VariableClassSpec = <value not set>`
- `Settings.NameTemplate = GLB_IF_VAR_BR`
- `Settings.VariableClass = <value not set>`
- `Settings.Type = <value not set>`

TargetLink ≤ 4.2	TargetLink 4.3
<pre>struct BS_IP_Sa2_In1 IF_Sa2_Out1 = { 0, /* Sa2_a */ 0 /* Sa2_b */ };</pre>	<pre>struct BS_IP_Sa2_In1 GLB_IF_VAR_Out1 = { 0, /* Sa2_a */ 0 /* Sa2_b */ };</pre>

Reason Matches user expectations

Migration issue If you want to restrict the `VariableTemplate` to plain variables, select all the bit widths in the `Filter.WidthSpec` property's bitfield.

Code optimization and Stateflow TargetLink no longer partially optimizes the code generated from Stateflow if code optimization is disabled. This affects the propagation of initial values and folding of arithmetic operations. Additionally, if optimization is enabled, suitable code comments are generated.

Reasons

- Matches user expectations
- Readability

Migration issue None

2-D look-up table functions with matrix parameters The code changed for 2-D look-up table functions with matrix parameters:

TargetLink ≤ 4.2	TargetLink 4.3
Access to the matrix itself: <pre>int tab[3][4]; int axis[6]; fcn(&(axis[0]), N, x); fcn2(tab, N, x, y);</pre>	Access to the matrix's first element address: <pre>int tab[3][4]; int axis[6]; fcn(&(axis[0]), N, x); fcn2(&(tab[0][0]), N, x, y);</pre>

Reason Bug fix for access functions**Migration issue** None

Addition and subtraction with saturation The code might change for additions and subtractions with saturation, if the ExploitComputeThroughOverflow Code Generator option is set to 1 - Never.

TargetLink ≤ 4.2
<pre>Sa1_I32ADDI32I32 = Sa1_InPort_Int32 + Sa1_InPort1_Int32; if ((Sa1_InPort_Int32 >= 0) && (Sa1_InPort1_Int32 >= 0) && (Sa1_I32ADDI32I32 < 0)) { Sa1_I32ADDI32I32 = 2147483647; } else { if ((Sa1_InPort_Int32 < 0) && (Sa1_InPort1_Int32 < 0) && (Sa1_I32ADDI32I32 >= 0)) { Sa1_I32ADDI32I32 = (-2147483647L -1L) /* INT32MIN */; } }</pre>

TargetLink 4.3

```
if ((Sa1_InPort1_Int32 > 0) && (Sa1_InPort_Int32 > (2147483647 - Sa1_InPort1_Int32))) {  
    Sa1_I32ADDI32I32 = 2147483647;  
}  
else {  
    if ((Sa1_InPort1_Int32 < 0) && (Sa1_InPort_Int32 < ((-2147483647L -1L) /* INT32MIN */ /* -2147  
483648. */ - Sa1_InPort1_Int32))) {  
        Sa1_I32ADDI32I32 = (-2147483647L -1L) /* INT32MIN */;  
    }  
    else {  
        Sa1_I32ADDI32I32 = Sa1_InPort_Int32 + Sa1_InPort1_Int32;  
    }  
}
```

Reason Matches user expectations

Migration issue None

Code Changes between TargetLink 4.1 and TargetLink 4.2

Code Changes between TargetLink 4.1 and TargetLink 4.2

Changed calculation sequence in atomic subsystems	The production code calculation sequence in atomic subsystems can change in contexts where the block execution order was not completely determined by the control and data flow.
	Related documentation: Improved Code Stability on page 91

Non-boolean operands in boolean contexts	The code for non-boolean operands in boolean contexts changed. This affects logical operations, control flow conditions, constants, and enums in unclean modeling situations (i.e., feeding non-boolean signals):
---	---

TargetLink ≤ 4.1	TargetLink 4.2
<pre>Int16Var && BoolVar if(Int16Var) { ... } 42 0</pre> <p>No support for enum constants.</p>	<pre>(Int16Var != 0) && BoolVar if (Int16Var != 0) { ... } 42 != 0 0 != 0</pre> <p>For operands that are an enum variable or enum constant, whose enum type provides an enum-constant for 0, this constant is used:</p> <pre>enum BasicColors = {RED = 0, GREEN = 1, ...} BasicColors color; color != RED GREEN != RED</pre>

Reason To comply with MISRA C.

Migration issue To revert to code as generated by TargetLink versions ≤ 4.1, deactivate the ForceBooleanOperandsInBooleanOperations Code Generator option.

Changes to the enum code section	The code section for enums is moved directly behind the code section for includes. This affects source and header files.
	Reason Enums can be required for other definitions whose section occurs after the include section.

Auxiliary variables for interruptable communication	For improved code readability, auxiliary variables created for interruptable communication now have the same name as the DD InterRunnableVariable object:
--	---

TargetLink ≤ 4.1	TargetLink 4.2
<pre>sint32 Aux_; Aux_ = (sint32) Sa2_Rescaler; Rte_IrvWrite_Run_MyIrv(Aux_);</pre>	<pre>sint32 MyIrv; MyIrv = (sint32) Sa2_Rescaler; Rte_IrvWrite_Run_MyIrv(MyIrv);</pre>

Pointer variables

The names of IRV variables that are pointers are prefixed by p_.

Math block's pow function

When using the pow function of the Math block, calculation differences between Simulink and TargetLink's production code in corner cases are removed.

TargetLink ≤ 4.1	TargetLink 4.2
<pre>if ((Sa1_u_ <= 0.F) && (((Float64) Sa1_v) ! = floor((Float64) Sa1_v))) { ... }</pre>	<pre>if ((Sa1_u_ < 0.F) && (((Float64) Sa1_v) ! = floor((Float64) Sa1_v))) { ... }</pre>

Discrete-Time Integrator block

The code generated for the Discrete-Time Integrator block with Integration set to Forward Euler and Limits other than -inf/inf can change:

TargetLink ≤ 4.1	TargetLink 4.2
<pre>UInt16 X_Sa1_DTI_FW_Sat = 10 /* 5. */ /* LSB: 2^-1 OFF: 0 MIN/MAX: 2 .. 730 */;</pre>	<pre>Int16 X_Sa1_DTI_FW_Sat = 10 /* 5. */ /* LSB: 2^-1 OFF: 0 MIN/MAX: -98 .. 630 */;</pre>

Reason To fit the data type chosen during code generation due to a corrected range calculation.

The code generated for the External reset changed.

TargetLink ≤ 4.1	TargetLink 4.2
<pre>/* Discrete Integrator: Subsystem/DTI: Condition for either edge trigger */ if ((Sa1_Reset >= 0) && (X_Sa1_DTI_TriggerIn <= 0) && ((Sa1_Reset != X_Sa1_DTI_TriggerIn) && (Sa1_DTI_LastEvent != 1))) { Sa1_DTI_LastEvent = 1; } else { if ((Sa1_Reset <= 0) && (X_Sa1_DTI_TriggerIn >= 0) && ((Sa1_Reset != X_Sa1_DTI_TriggerIn) && (Sa1_DTI_LastEvent != -1))) { Sa1_DTI_LastEvent = -1; } else { Sa1_DTI_LastEvent = 0; } }</pre>	<pre>/* Discrete Integrator: Subsystem/DTI: Condition for either edge trigger */ if ((Sa1_Reset > 0) && (X_Sa1_DTI_TriggerIn <= 0) && ((Sa1_Reset != X_Sa1_DTI_TriggerIn) && (Sa1_DTI_LastEvent != 1))) { Sa1_DTI_LastEvent = 1; } else { if ((Sa1_Reset <= 0) && (X_Sa1_DTI_TriggerIn > 0) && ((Sa1_Reset != X_Sa1_DTI_TriggerIn) && (Sa1_DTI_LastEvent != -1))) { Sa1_DTI_LastEvent = -1; } else { Sa1_DTI_LastEvent = 0; } }</pre>

Reason Matching Simulink's behavior where only positive values are considered as rising edges.

Rate Limiter

The code generated for the `RemainderState` variable of the Rate Limiter block can change:

TargetLink ≤ 4.1	TargetLink 4.2
<code>static Int8 XRem_Sa1_Rate_Limiter1;</code>	<code>static Int16 XRem_Sa1_Rate_Limiter1;</code>
<code>static Int16 XRem_Sa1_Rate_Limiter2;</code>	<code>static Int32 XRem_Sa1_Rate_Limiter2;</code>

Reason To fix an incorrectly chosen data type that was too small.

An additional cast can also be inserted in the calculation of the remainder:

TargetLink ≤ 4.1	TargetLink 4.2
<code>Aux_S16 = Aux_U16 % 100;</code>	<code>Aux_S16 = (Int16) (Aux_U16 % 100);</code>

Reason To comply with MISRA C.

Function reuse and pointer-to-struct for Stateflow

The code optimization for function reuse and pointer-to-structs in Stateflow was improved.

TargetLink ≤ 4.1	TargetLink 4.2
<code>pISV->comp = 0; pISV->comp = 1;</code>	<code>pISV->comp = 1;</code>

Order of # combined # comments

In some cases, the order of the `# combined` `#` comments above function calls can change:

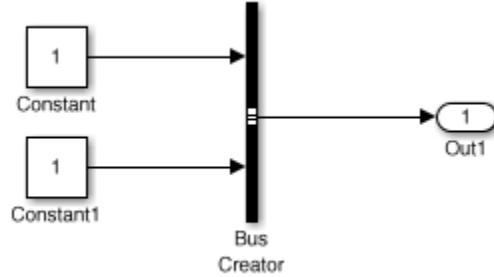
TargetLink ≤ 4.1	TargetLink 4.2
<code>/* call of function: SimpleCall/Subsystem # combined # update(s) for inport SimpleCall/Subsystem/InPort # combined # Gain: SimpleCall/Gain # combined # Gain: SimpleCall/Gain1 # combined # TargetLink outport: SimpleCall/OutPort */ foo(a, h);</code>	<code>/* call of function: SimpleCall/Subsystem # combined # update(s) for inport SimpleCall/Subsystem/InPort # combined # Gain: SimpleCall/Gain1 # combined # Gain: SimpleCall/Gain # combined # TargetLink outport: SimpleCall/OutPort */ foo(a, h);</code>

IF variable in classic initialization mode

In [classic initialization mode](#), the generated code changes for unenhanced Outport blocks that receive data from a non-virtual bus created by a Bus Creator block that directly precedes them. This can result in `IF` variables having a different name:

TargetLink ≤ 4.1	TargetLink 4.2
<code>void Sa2_Subsystem(void) { /* update of variable(s) associated with TL_Root/Subsystem/Constant */ IF_Sa2_a = 1; /* update of variable(s) associated with TL_Root/Subsystem/Constant1 */ IF_Sa2_b = 1; }</code>	<code>static void Sa2_Subsystem(void) { /* Outport: TL_Root/Subsystem/Out1 */ IF_Sa2_Out1 = 1; IF_Sa2_Out1_a = 1; }</code>

Model used The code example in the preceding table was generated from a model such as the following:



The signals in the non-virtual bus are called **a** and **b**.

Unit Delay Reset Enabled

The code generated for Unit Delay Reset Enabled blocks whose Use external initial value checkbox is selected was changed. When calculating the block's state variable, TargetLink no longer checks the `enable` condition for the `FirstRun` variable:

TargetLink ≤ 4.1	TargetLink 4.2
<pre> /* Subsystem/UDRE: reset and external initial value condition */ if ((Sa3_R != 0) Sa3_Subsystem_FirstRun) { /* Subsystem/UDRE: state initialization */ X_Sa3_UDRE = Sa3_IV; } /* Subsystem/UDRE: output */ Sa3_UDRE = X_Sa3_UDRE; /* Subsystem/UDRE: enable condition */ if ((Sa3_E != 0) && ((!(Sa3_R != 0)) && (! (Sa3_Subsystem_FirstRun != 0)))) { /* Subsystem/UDRE: state update */ X_Sa3_UDRE = Sa3_u; } </pre>	<pre> /* Subsystem/UDRE: reset and external initial value condition */ if ((Sa3_R != 0) Sa3_Subsystem_FirstRun) { /* Subsystem/UDRE: state initialization */ X_Sa3_UDRE = Sa3_IV; } /* Subsystem/UDRE: output */ Sa3_UDRE = X_Sa3_UDRE; /* Subsystem/UDRE: enable condition */ if ((Sa3_E != 0) && (!(Sa3_R != 0))) { /* Subsystem/UDRE: state update */ X_Sa3_UDRE = Sa3_u; } </pre>

Reason The block's behavior for the internal and external initial values was not consistent, if the following conditions applied:

- The R input received the value **false**.
- The E input received the value **true**.

In this case the internal and external initial values are now identical in MIL and SIL simulation modes.

AUTOSAR version as code comment

When generating code in AUTOSAR mode, TargetLink now places the AUTOSAR version in the code files' opening headers.

Reason The AUTOSAR version impacts on code generation and thus should be clearly visible.

Auxiliary variables instead of iteration variables

Iteration variables are in some cases replaced by auxiliary variables, if optimization is enabled.

TargetLink ≤ 4.1	TargetLink 4.2
Sa3_For_Iterator_it ¹⁾	Aux_S32_a ²⁾

¹⁾ Name configured as \$S_\$B_it.

²⁾ Name configured as Aux_\$T\$R.

Reason Changes in the code generator optimization.

Definitions of variables and macros

The definitions of variables and macros can be different:

- There can be variables of an `enum` type.
- Macro definitions are now sorted within functions and they receive code comments depending on the variable class.
- Parameterized macros (like macro access functions) are now grouped and sorted.

Reason

- Support for C enum types
- Code readability

Iteration variable of while-subsystems

The iteration variable generated for do-while-subsystems now always is incremented at the very end of the `while` loop. This also affects variables for further conditions:

TargetLink ≤ 4.1	TargetLink 4.2
<pre>while (Sa2_While_Iterator_cond && (Sa2_While_Iterator_it <= 4)) { ... Sa2_While_Iterator_cond = 1; Sa2_While_Iterator_it++; /* Unit delay: TL_SS/For Iterator Subsystem/Unit Delay */ X_Sa2_Unit_Delay = (Int16) Sa2_While_Iterator_it; }</pre>	<pre>while (Sa2_While_Iterator_cond && (Sa2_While_Iterator_it <= 4)) { ... /* Unit delay: TL_SS/For Iterator Subsystem/Unit Delay */ X_Sa2_Unit_Delay = (Int16) Sa2_While_Iterator_it; Sa2_While_Iterator_cond = 1; Sa2_While_Iterator_it++; }</pre>

Reason There was a possibility for incorrect code in the context of Unit Delay blocks.

Code format of exponents

The code format of exponents changed from three to two digits in case one or two digits suffice.

Reason To follow the change in Microsoft's CRT runtime library to improve floating-point precision and better compliance with the C standard. For more information, refer to https://msdn.microsoft.com/en-us/library/bb531344.aspx#BK_CRT.

Reuse of table maps

Code has changed for table map variables of look-up table functions with regard to function reuse:

TargetLink ≤ 4.1	TargetLink 4.2
Tables and axes referenced via pointers remained in the reused system.	Tables and axes referenced by pointers are moved to the file that contains the reuse structure.

Reason To facilitate component-based development.

If Action and Switch Case Action subsystems with StateReset

The code generated for If Action and Switch Case Action subsystems with StateReset was changed. The variable that stores the currently executed system is no longer superfluously initialized.

TargetLink ≤ 4.1	TargetLink 4.2
<pre>UInt8 Sa1_If_System = 0; if (Sa1_t1In1 > 16384 /* 0.5 */) { Sa1_If_System = 1; ... } else { Sa1_If_System = 2; }</pre>	<pre>UInt8 Sa1_If_System; if (Sa1_t1In1 > 16384 /* 0.5 */) { Sa1_If_System = 1; ... } else { Sa1_If_System = 2; } LastSystem = Sa1_If_System;</pre>

Reason To comply with MISRA C .

State charts with change detection

The code generated for state charts with change detection for `input`, `Datastore`, or `Machine` `data` receives a new variable called `FirstRun`.

Reason Remove erroneous differences between Stateflow and the generated production code pattern.

New code comments

TargetLink now generates additional or changed code comments for the following code elements:

- Additional comments for grouping declarations and definitions of variables or functions.
- PreBlockStatements and PostBlockStatements are now always enclosed by empty lines.
- Non-empty CodeSection-StatementTables like includes or typedefs are now always followed by an empty line.

Reason To increase consistency across code patterns.

Algebraic simplification

TargetLink's code changed for algebraic simplification of expressions like `UInt16Var > 0`:

TargetLink ≤ 4.1	TargetLink 4.2
<code>UInt16Var</code>	<code>UInt16Var != 0</code>

Reason Keep boolean-context for MISRA C compliance and possibly facilitate further code optimizations.

Logical operations with floating-point operands

For logical operations that contain at least one operand of a floating-point type, TargetLink no longer casts all the operands to the greatest floating-point type contained within the operation:

TargetLink ≤ 4.1	TargetLink 4.2 ¹⁾
<pre>Sa1_F32F64AND = ((Float64) F32Var) && F64Var; Sa1_F32I16OR = ((Float32) I16Var) F32Var; Sa1_F32ScaledAND = F32Var && (((Float32) ScaledI16Var) * 0.5F) + 2.F); // ScaledI16Var: Lsb = 0.5, Offset = 2.0 Sa1_F32IntValueAND = F32Var && 1.F; Sa1_F32F1pValueOR = 1.25F F32Var;</pre>	<pre>Sa1_F32F64AND = (F32Var != 0.F) && (F64Var != 0.); Sa1_F32I16OR = (I16Var != 0) (F32Var != 0.F); Sa1_F32ScaledAND = (F32Var ! = 0.F) && ((ScaledI16Var + 4) != 0); // ScaledI16Var: Lsb = 0.5, Offset = 2.0 Sa1_F32IntValueAND = (F32Var != 0.F) && (1 != 0); Sa1_F32F1pValueOR = (1.25F != 0.F) (F32Var != 0.F);</pre>

¹⁾ With the ForceBooleanOperandsInBooleanOperations Code Generator option set to on.

Reason To comply with MISRA C.

Scaling-invariant graphical functions in Stateflow

The code generated for scaling-invariant graphical functions in Stateflow can lack optimizations.

Reason This is a fail-safe to prevent false code optimization of code carrying side effects.

Solution Use a function class whose Optimization property is set to SIDE_EFFECT_FREE for the graphical function.

Note

Do this only if all the variables that this function operates on are function local and the function does not trigger any event.

Simulink enum constants

The code for Simulink enum constants changed for following modeling elements. They now longer are integers but enumeration constants of an enum data type that is implicitly generated by TargetLink:

- Case conditions of Simunlink's Switch-Case block
- Simulink enum constants in the Stateflow action language
- Data port indices of TargetLink's Multiport Switch block
- Threshold of TargetLink's Switch blocks specified as follows:
 - They do not reference a DD Variable object.
 - Their Class is set to default.
- TargetLink's Constant blocks specified as follows:
 - They do not reference a DD Variable object.
 - Their Allow signal specification checkbox is cleared.
 - Their Class is set to default.

The following code is generated from a Multiport Switch block:

TargetLink ≤ 4.1	TargetLink 4.2
<pre>switch (Sa1_InPort1) { case 0: { Sa1_Merge = Sa1_InPort2; break; } case 2: case 3: { Sa1_Merge = Sa1_InPort3; break; } default: { Sa1_Merge = Sa1_InPort4; break; } }</pre>	<pre>switch (Sa1_InPort1) { case BasicColors_Red: { Sa1_Merge = Sa1_InPort2; break; } case BasicColors_Green: case BasicColors_Blue: { Sa1_Merge = Sa1_InPort3; break; } default: { Sa1_Merge = Sa1_InPort4; break; } }</pre>

The definition of TargetLink's enum data type looks like this:

```
typedef enum BasicColors_tag {
    BasicColors_Red = 0,
    BasicColors_Yellow = 1,
    BasicColors_Green = 2,
    BasicColors_Blue = 3,
    BasicColors_Orange = 4,
    BasicColors_Black = 5,
    BasicColors_Pink = 6,
    BasicColors_Brown = 7,
    BasicColors_Magenta = 8
} BasicColors; /* Description: Enumeration type derived from Simulink type
BasicColors */
```

Reason To fully support enum data types.

Solution To revert to code as generated by TargetLink versions ≤ 4.1, create a DD EnumTemplate object whose Filter property is set to ALWAYS and that references the DD EnumImplementation object.

Additionally, you may adjust your mapping files for model preparation to **EnumImplementation**.

Constraint ranges of reused variables and data variants

The code for variables has changed in the context of function reuse and data variants, if the corresponding DD Variable object is specified as follows:

- Its Min property is not specified.
- Its Max property is not specified.
- Its VariableClass object's Const property is set to on.
- Its VariableClass object's Info property is set to **none**, **readonly**, or **bypassing_READONLY**.

TargetLink now uses the type ranges as these variables' constraint range limits. This can lead to code that is less efficient.

Reason To prevent possibly incorrectly over-optimized code in the context of function reuse and data variants.

Migration issue If you observe less efficient code for such variables, consider specifying their Min and Max values.

Code comments for type definitions and declarations of variables

The code comments for the following code elements changed:

- The constraints (ranges, scaling) of type definitions are now documented at the type definition:

```
typedef UInt16 ADC; /*  
    Unit: m  
    LSB: 2^-11 OFF: 0 MIN/MAX: -10 .. 10 */
```

- The order within code comments for declarations was changed. The information about the scaling and the description have changed places.

```
Int16 Sa1_InPort3; /*  
    <BlockComment>  
    Unit: Gigawatt  
    LSB: 2^-8 OFF: 0 MIN/MAX: 0 .. 0.99  
    Description: wash dish */
```

Additionally, a superfluous whitespace was removed from OFF and MIN/MAX.

- The ; in variable declarations and the , in struct initializers are now placed directly behind the identifier, not behind the optional comment:

```
Int16 Sa1_InPort3; /* Description: something */  
  
struct my_tiny_struct s {  
    5, /* Description: five */  
    6  
};
```

This does not affect comments describing the physical value of numeric constants:

```
Int32 MyVal = 37 /* 0.145 */; /* LSB: 2^-8 OFF: 0 MIN/MAX: 0 .. 0.99 */
```

- Comments for variables with the default scaling (LSB = 1, Offset = 0, no unit, relevant for all dimensions) no longer receive a scaling comment except when ranges are explicitly specified.
- Float variables with explicitly specified ranges now get MIN/MAX comments but never scaling comments.
- Macro variables (with the exception of RDI-macros) only get MIN/MAX comments for explicitly specified ranges but not for calculated ranges.
- Float variables now get Unit comments:

```
Float32 Sa1_Out1; /* Unit: <unit> */
```

Reason To establish consistency across code patterns and improve readability.

Moving code into conditionally executed branches

TargetLink no longer moves variables that are defined in all branches of a conditionally executed control flow, which is contained within a function, into a conditionally executed branch of a following control flow contained in the same function, if this variable is accessed within another function.

This can lead to less efficient code.

Reason To prevent incorrect code calculations by conservative optimization.

ExtendedLifeTimeOptimization

TargetLink now performs the ExtendedLifeTimeOptimization optimization for formal out parameters of conditionally executed subsystems:

TargetLink ≤ 4.1	TargetLink 4.2
<pre>Foo(Int * out) { Int B; B = ... ; ... //some Code *out = B; }</pre>	<pre>Foo(Int * out) { Int B; B = ... ; *out = B; ... //some Code }</pre>

Reason Improved optimization and code efficiency.

This optimization also affects access to return variables passed by reference in the `Rte_IWriteRef` and `Rte_IrvIWriteRef` functions of the AUTOSAR RTE.

Stateflow and casts to boolean

Boolean casts of non-booleans in Stateflow are now forced to become effectively `boolean` in the generated production code, independent of the coded type of TargetLink's `Bool` basetype:

TargetLink ≤ 4.1	TargetLink 4.2
<code>F32Var = (Float32)((Bool)I16Var);</code>	<code>F32Var = (Float32)(I16Var != 0);</code>

Reason Improve MISRA C compliance and fix different behavior in MIL and SIL simulation.

Details This code change cannot be influenced by the `ForceBooleanOperandsInBooleanOperations` Code Generator option.

Stateflow state encoding

For states whose `Create output for monitoring` property is set to `Self activity`, Targetlink now always uses this state's Active State Data to code activity of this state.

If TargetLink uses a boolean active state data for one child of S, it uses boolean state data for every child of S.

Reason To increase consistency accross code patterns.

Inheritance for blocks with multiple outputs

The code of the following blocks changed if they have multiple inputs and their `Inherit properties` checkbox is selected:

- Switch block
- Multiport Switch block
- MinMax block
- Merge block

This can also affect the code generated for the Bitwise Operator block under the following conditions:

- The block's Use bit mask checkbox is cleared.
- The block does not reference a DD Variable or ReplaceableDataItem object.

The output data type in TargetLink 4.2 is determined in the following way:

- The output data type is the data type of the first block input with valid signal specification provided all inputs with valid signal specification have the same minimum/maximum constraints.
- Otherwise, the output data type is the base data type of the first block input with valid signal specification if at least one input with valid signal specification has a different minimum/maximum constraints.

TargetLink ≤ 4.1

```
DataTypeSecondInput Sa1_SwitchScalar /* LSB: 2^0 OFF: 0 MIN/MAX: -100 .. 100 */;
```

TargetLink 4.2

```
DataTypeFirstInput Sa1_SwitchScalar; /* LSB: 2^0 OFF: 0 MIN/MAX: -100 .. 100 */
```

Reason To increase consistency across code patterns.

Further effect One further effect of this change is that in some cases structure types may not be inherited through multi input blocks. This may affect the code optimizations because structure variables are optimized differently than a collection of plain variables:

TargetLink ≤ 4.1	TargetLink 4.2
<pre>struct StructUDT_tag Sa1_SwitchStruct; /* Switch: Subsystem/SwitchStruct */ if (Sa1_CtrlIn >= 0) { /* Switch: Subsystem/SwitchStruct */ Sa1_SwitchStruct.a = Sa1_InPortNoUDTWithMinMax; Sa1_SwitchStruct.b = Sa1_InPortInt16; } else { /* Switch: Subsystem/SwitchStruct */ Sa1_SwitchStruct = X_Sa1_Unit_Delay; }</pre>	<pre>Int16 Sa1_SwitchStruct; /* LSB: 2^0 OFF: 0 MIN/MAX: -100 .. 100 */ Int16 Sa1_SwitchStruct_a; /* Switch: Subsystem/SwitchStruct */ if (Sa1_CtrlIn >= 0) { /* Switch: Subsystem/SwitchStruct */ Sa1_SwitchStruct = Sa1_InPortNoUDTWithMinMax; Sa1_SwitchStruct_a = Sa1_InPortInt16; } else { /* Switch: Subsystem/SwitchStruct */ Sa1_SwitchStruct = X_Sa1_Unit_Delay.a; Sa1_SwitchStruct_a = X_Sa1_Unit_Delay.b; }</pre>

Solution If you want structured variables in production code as in TargetLink 4.1, do one of the following:

- Make sure that the corresponding bus elements at all the inputs of the multi-input block have the same data type (i.e., the first bus element at all the inputs have the same data type, the second bus element at all the inputs have the same data type and so on).
- Explicitly specify the structure in the Data Dictionary and reference the type or variable at each block in the model.

Code Changes between TargetLink 4.0 and TargetLink 4.1

Code Changes between TargetLink 4.0 and TargetLink 4.1

Code generated from Stateflow charts

In order to follow special Stateflow semantics more precisely (e.g., to avoid MIL/SIL differences for certain modeling styles), code generated from Stateflow charts might be less efficient in the following Stateflow chart scenarios:

- Graphical functions imported from other charts are called.
- Function call output events occur.

As a result, constants might not be propagated, or unused code fragments might not be removed.

To change this, assign a function class whose `SIDE_EFFECT_FREE` optimization flag is enabled to the imported graphical functions and to the subsystems/charts that are triggered by function call output events.

Note

You must guarantee that the function is actually side-effect-free. For example, side-effect-free functions do not:

- Modify global variables
- Call functions that are not side-effect-free

Assignment blocks in iterated subsystems

If multiple Assignment blocks reside in an iterated subsystem, only one first iteration flag is implemented, which increases code efficiency. As a result, the name of such flags changes, for example:

TargetLink ≤ 4.0	TargetLink 4.1
<code>{Subsystem_AssignmentBlock}_FirstIter</code>	<code>{Subsystem_IterationBlock}_FirstIter</code>

The flag is explicitly initialized at the beginning of the iteration loop:

```
..._FirstIter = 1
```

The flag is reset at the end of the loop:

```
..._FirstIter = 0
```

Assignment blocks in nested subsystems If at least one Assignment block resides in an atomic subsystem and this atomic subsystem resides in an iterated subsystem, the following variable scopes are assigned to the iteration variable:

- *Global* if the function of the atomic subsystem is *not inlined*.
- *Local* if the function of the atomic subsystem is *inlined*.

Code patterns for saturated additions or subtractions	For TargetLink 4.1, compute-through-overflow (CTO) code patterns are never used in saturation code of additions or subtractions, if the ExploitComputeThroughOverflow Code Generator option is set to NEVER. The following example shows an addition <code>out = in + Const</code> with <code>I16Out = I16In +1</code> :
--	--

TargetLink ≤ 4.0	TargetLink 4.1
<pre>if (I16In > 32766) { /* Max(Result type) - Const */ I16Out = 32767; } else { I16Out = (Int16) ((/* CTO */ (UInt16) I16In) + 1); }</pre>	<pre>if (I16In > 32766) { /* Max(Result type) - Const */ I16Out = 32767; } else { I16Out = (Int16) (I16In + ((Int16) 1)); }</pre>

If the ExploitComputeThroughOverflow Code Generator option is set to Always or to Optimized, the production code remains unchanged compared to previous versions.

No Void typedef	For improved MISRA C compliance, TargetLink by default no longer generates the <code>Void</code> typedef within <code>tl_basatypes.h</code> . It uses the standard C <code>void</code> data type instead. You can change this back to the old behavior (using <code>Void</code>) by editing the <code>TargetConfig.xml</code> file that belongs to your target-compiler combination. This file is located in <code><TL_InstRoot>\Matlab\Tl\TargetConfiguration\<MicrocontrollerFamily>\<CompilerFamily></code> or similar in the TSM extension folder. To instruct TargetLink, to generate the <code>Void</code> base type in <code>tl_basetypes.h</code> again, do the following: 1. Open the <code>TargetConfig.xml</code> file that belongs to your target-compiler combination. 2. Locate the <code>ddObj</code> XML element whose name attribute is set to <code>Void</code> . 3. Locate the child <code>ddProperty</code> XML element whose name attribute is set to <code>CodedType</code> . 4. Change its value from <code>Use standard C void type</code> to <code>void</code> . 5. Save the <code>TargetConfig.xml</code> file and generate code.
------------------------	---

Additional default scaling code comments	For readability, TargetLink now places additional code comments concerning default scalings at variable definitions:
---	--

TargetLink ≤ 4.0	TargetLink 4.1
<code>Int16 SX1_OutPort1_FR_Actual;</code>	<code>Int16 SX1_OutPort1_FR_Actual /* LSB: 2^0 OFF: 0 MIN/MAX: -32768 .. 32767 */;</code>

Encapsulation of preprocessor #IF statements	TargetLink now encapsulates preprocessor <code>#include</code> directives by <code>#IF</code> directives only if all of the included file's definitions and declarations are encapsulated by <code>#IF</code> directives.
---	---

As an example, consider the following `FuncDefModule.h` header file:

```
extern GLOBAL Int16 SEnc1_Out1;

#if FLAG
extern void Encapsulated(void);
#endif
```

TargetLink's generated code changes as follows:

TargetLink ≤ 4.0	TargetLink 4.1
<pre>#if FLAG #include "FuncDefModule.h" #endif ... void TL_Root(void) { #if FLAG Encapsulated(Sa1_InPort); #endif SEnc1_Out1 = ... }</pre>	<pre>#include "FuncDefModule.h" ... void TL_Root(void) { #if FLAG Encapsulated(Sa1_InPort); #endif SEnc1_Out1 = ... }</pre>

Code comments at indices of vector or matrix variables

For improved readability, the code comments concerning the initial values of vector or matrix variables have changed:

TargetLink ≤ 4.0	TargetLink 4.1
Vectors	
<pre>Int16 MyVar[3] = { /*[0..2]*/ 61, 52, 43 };</pre>	<pre>Int16 MyVar[3] = { /* [0..2] */ 61, 52, 43 };</pre>
Matrices	
<pre>Int16 MyVar[2][3] = { { /*[0..2]*/ 61, 52, 43 }, { /*[0..2]*/ 34, 25, 16 } };</pre>	<pre>Int16 MyVar[2][3] = { { /* [0][0..2] */ 61, 52, 43 }, { /* [1][0..2] */ 34, 25, 16 } };</pre>

Identifier of implicitly generated struct types

TargetLink's identifiers for implicitly generated struct types now comply with the AUTOSAR standard. They now comply with the following regular expression:

[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?

For typedef identifiers that you specified by using the \$C, \$R, or \$S name macros, TargetLink now does the following:

- Removes underscores at the beginning of the identifier of implicitly generated typedefs
- Replaces double underscores by a single one

TargetLink ≤ 4.0	TargetLink 4.1
<pre>/* update(s) for import Subsystem/Func/In1_with_super_Long_name_to_break_Limit */ Rte_Pim_ACP_a(instance)->Sa2_In1_with_s__to_break_limit = (sint16) DataElement;</pre>	<pre>/* update(s) for import Subsystem/Func/In1_with_super_Long_name_to_break_Limit */ Rte_Pim_ACP_a(instance)->Sa2_In1_with_s_to_break_limit = (sint16) DataElement;</pre>

Identifiers that consist only out of underscores or that begin with an underscore immediately followed by a numeral are not changed. These typedefs are not generated into an autogenerated per instance memory (PIM):

TargetLink ≤ 4.0	TargetLink 4.1
<pre>/* update(s) for import Subsystem/Func/In1 */ Rte_Pim_ACP_a(instance)->_1Var = (sint16) DataElement;</pre>	<pre>/* update(s) for import Subsystem/Func/In1 */ _1Var = (sint16) DataElement;</pre>

IF-variable for outports of conditionally executed systems

To eliminate possible differences between MIL and SIL simulations, TargetLink now generates an additional variable, IF_<Suffix>, for unenhanced outports of conditionally executed subsystems if these are preceded by one of the following blocks:

- Data Type Conversion
- (Matrix) Concatenate
- Permute Dimensions
- Reshape
- Selector
- Bus Assignment
- Zero Order Hold
- Rate Transition

Code comments for optimization

For improved readability, TargetLink's code comments for optimization changed. The chain A replaced by ... replaced by ... X is now replaced by A replaced by X:

TargetLink ≤ 4.0	TargetLink 4.1
<pre>/* Gain: foo/Gain Variable 'Sa1_Gain' replaced by 'Aux_f' Variable 'Aux_f' replaced by 'Aux_F32_e' */</pre>	<pre>/* Gain: foo/Gain Variable 'Sa1_Gain' replaced by 'Aux_F32_e' */</pre>

No access functions for auxiliary variables

You can no longer define access function templates (AFTs) for auxiliary variables that result from access functions. The former behavior created more access functions than desired or even caused a near-infinite loop during code generation.

New CTO avoidance macros for additions and subtractions

To improve compliance with MISRA C, TargetLink's Fixed-Point Library provides new macros for additions and subtractions for operands ≤ 16 bit. These macros are generated only when you suppress CTO code patterns (via the **ExploitComputeThroughOverflow** Code Generator option). The macro names always end with _PROT and contain an I16 or U16.

TargetLink ≤ 4.0	TargetLink 4.1
<code>C__I32ADDI32U32_PROT((Int32) Sa1_I16InPort, Sa1_U32InPort)</code>	<code>C__I32ADDI16U32_PROT(Sa1_I16InPort, Sa1_U32InPort)</code>

Writing dereferences in logical operations in parentheses

Dereferences in logical operations are now written in parentheses to comply with MISRA C.

TargetLink ≤ 4.0	TargetLink 4.1
<code>Sa1_OutPort1 = (Int16) (*In2 *In1);</code>	<code>Sa1_OutPort1 = (Int16) ((*In2) (*In1));</code>

MinMax block code pattern

For MinMax blocks with more than two inputs or an input which has more than two signals, the code pattern changes when the first two inputs or signals do not fit in the output. This can mean one of the following:

- The order of the comparison changes.
- The output is initialized with minimum or maximum values before comparing it with each signal.

Status of RTE_Invalidate

TargetLink can now evaluate the return value of the `Rte_Invalidate` RTE API function. The following table shows a code sample of a `Rte_Invalidate` call for a SenderComSpec block with enabled Invalidate and Status ports:

TargetLink ≤ 4.0	TargetLink 4.1
<pre>if(InvalidationCondition) { Rte_Invalidate_x_y(); status = 0; } else { status = Rte_Write_x_y(); }</pre>	<pre>if(InvalidationCondition) { status = Rte_Invalidate_x_y(); } else { status = Rte_Write_x_y(); }</pre>

Changed subsystem naming for incremental code generation

The code of subsystems configured for incremental code generation that were converted from referenced models might now be stored at a different location than in previous versions. For details, refer to [Various Migration Aspects](#) on page 341.

Base types for logging variables

If an auxiliary variable is created for a Sink block for logging, it always gets a TargetLink base type (in AUTOSAR: a platform type). However, logging code is not production code.

Abs pattern changes

For better readability, condensed fixed-point Abs patterns are no longer supported. Instead, if-else expressions are used (e.g., an unscaled `Int16` Abs with unscaled `Int16` input).

TargetLink ≤ 4.0	TargetLink 4.1
<pre>Sa1_OutPort = Sa1_InPort; if (Sa1_InPort < 0) { Sa1_OutPort = -Sa1_OutPort; }</pre>	<pre>if (Sa1_InPort >= 0) { Sa1_OutPort = Sa1_InPort; } else { Sa1_OutPort = (Int16) (-Sa1_InPort); }</pre>

The following changes might apply:

- Subsequent code patterns might change because another (usually better) range information is inherited: e.g., after the paragraph is followed by a sign, its negative branch is omitted, because the output of the Abs is always a positive value.
- With saturation, additional 64-bit operations are generated when `KeepSaturationStatements` is set.
- In Stateflow expressions like `FloatOutVar = abs (IntVar)`, another code pattern is generated (with the ? operator).

Additional casts in nonscalar AUTOSAR or indirect function reuse

Whenever a pointer with indices on the left side is accessed, there might be additional casts that now better conform with TargetLink's general casting style:

TargetLink ≤ 4.0	TargetLink 4.1
<code>pISV->pISV_SL1_0_tp->pSL1_ImplicitOut1[Aux_S32] = 0;</code>	<code>pISV->pISV_SL1_0_tp->pSL1_ImplicitOut1[Aux_S32] = (sint16) 0;</code>

Note

In this particular case, the cast is created because the ground symbol was originally on the right side.

Overflow-free, unary minus

The following example shows code for an Abs block with `Int16` input and `UInt16` output:

TargetLink ≤ 4.0	TargetLink 4.1
<pre>if(I16In >= 0) { ... } else { U16Out = (UInt16)(-I16In) }</pre>	<pre>if(I16In >= 0) { ... } else { if (I16In == -32768) { U16Out = 32768; } else { U16Out = (UInt16)(-I16In) } }</pre> <p>Now the smallest negative value is used.</p>

This pattern is critical, because the value -32768 (`INT16MIN`) might cause an undefined overflow. 32768 does not fit in a 16-bit-platform `int` and the minus is calculated in `int`.

Note

The second *if-else* might be replaced by a ? operator.

```
... (I16In == -32768) ? 32768 : (UInt16)(-I16In)
```

This is always the case when the unary minus is not the last operation on the right side of an assignment. A typical example is an Abs block with a different input/output scaling. The minus is then the operand of the rescaling (shift/division). As a result, the ? operator is used.

The following expressions and blocks can be affected:

- Stateflow expressions with unary minus
- Abs blocks
- Gain blocks with negative gain values
- Product blocks with a negative constant default
- Sum blocks with the corresponding settings

Saturation macros: FIT macro calls replace SAT macro calls

There are two kinds of saturation macros in TargetLink's Fixed-Point Library:

- FIT macros that perform saturation on range limits (implemented range)
- SAT macros that perform saturation on user-defined limits (e.g., in the Saturation block or in Stateflow)

Prior to TargetLink 4.1, in rare cases a SAT macro was called although a saturation on range limits was performed. This is now changed for consistency and for improved MISRA C compliance.

The following example shows saturation in Stateflow with

```
UInt16 U16Out ; // LSB = 0.002 and checkmax = 1  
U16Out = U16Out + I8In;
```

TargetLink ≤ 4.0

```
Aux_I32 = ...  
U16Out = C_U16SATI32_SATb(Aux_I32, 65535 /* 131.07 */, 0)
```

TargetLink 4.1

```
Aux_I32 = ...  
U16Out = C_U16FITI32_SAT(Aux_I32, 65535 /* 131.07 */ )
```

Saturation macros

When calling SAT macros that are generated for the Saturation block or Stateflow, the type of the limits (parameter 2,3) is adjusted to match the type of the output if the limits have a variable class that is different from the default.

TargetLink ≤ 4.0	TargetLink 4.1
<pre>GLOBAL Int32 Sb1_Saturation_lower = 0; GLOBAL Int32 Sb1_Saturation_upper = 100; Int16 Aux_S16; Int16 Aux_S16_a; Aux_S16 = (Int16) Sb1_Saturation_upper; Aux_S16_a = (Int16) Sb1_Saturation_lower; Sb1_OutPort = C__U16SATI16_SATb(Sb1_InPort, Aux_S16, Aux_S16_a);</pre>	<pre>GLOBAL Int32 Sb1_Saturation_lower = 0; GLOBAL Int32 Sb1_Saturation_upper = 100; UInt16 Aux_U16; UInt16 Aux_U16_a; Aux_U16 = (UInt16) Sb1_Saturation_upper; Aux_U16_a = (UInt16) Sb1_Saturation_lower; Sb1_OutPort = C__U16SATI16_SATb(Sb1_InPort, Aux_U16, Aux_U16_a);</pre>

Accessing matrix variables within loops

For code efficiency, accessing matrix variables within loops can now be replaced by scalar variables. In certain cases, matrix and vector variables are replaced by one or more scalar variables, even if they are accessed from outside the loops.

TargetLink ≤ 4.0	TargetLink 4.1
<pre>Int16 vec[...]; vec[0] = = v[0]; loop (i = 1:n) { v[i] = ... = v[i]; }</pre>	<pre>Int16 scalar; scalar = = scalar; loop (i = 1:n) { scalar = = scalar; }</pre> <p>Or, if more scalar variables are introduced:</p> <pre>Int16 scalar1; Int16 scalar2; scalar1 = = scalar1; loop (i = 1:n) { scalar2 = = scalar2; }</pre>

Accessing matrix variables outside of loops

Accessing matrix variables from outside the loops can now be replaced by one or more scalar variables. The combination of accesses within and outside of loops can also now be optimized. In addition, unknown accesses for indices can also be optimized if the index expression accesses is built up identically.

During replacement by scalar variables, memory savings typically emerge (in the aggregate). However, in some situations it is not possible to reduce memory consumption: i.e., the new scalar variables use as much memory as the initial multidimensional variable.

This takes place only for completely unrolled code, precisely if:

- (For a vector variable) Number of elements < LoopUnrollThreshold
- (For a matrix variable) Number of elements < LoopUnrollThreshold²

Accesses from ...	TargetLink ≤ 4.0	TargetLink 4.1
Outside of loops, analog for vectors	<code>M[0][0] = ...; = M[0][0];</code>	<code>Aux = ...; = Aux;</code>
Within and outside (of partially unrolled) loops, analog for vectors	<code>M[0][0] = ...; loop { M[1][i] = ... M[2][i] = = M[1][i]; ... = M[2][i]; }</code>	<code>Aux = ...; loop { Aux_a = ... Aux_b = = Aux_a; ... = Aux_b; }</code>
Outside of loops, analog for matrices	<code>V[<expr>] = ...; = V[<expr>];</code>	<code>Aux = ...; = Aux;</code>

Tangents code patterns

The tangents code pattern from within Stateflow and for the TargetLink Trigonometric and Math blocks was modified.

Tangents within Stateflow Up to TargetLink 4.1, saturation was erroneously omitted, especially for saturated expressions.

Now the following rules apply:

- If a tangent is used in an assignment such as `out = tan(expr)`, then the tangent is calculated in fixed-point when `out` has a fixed-point type.
- In all other cases, e.g., complex expressions, TargetLink performs tangent calculations in floating-point. If a pure fixed-point context is detected during code generation, an error message occurs.

Tangents for the Trigonometric and Math blocks Now, unnecessary saturations are omitted by default if the result is either 32-bit or 16-bit with $LSB \geq 2^{-14}$. Therefore, *Omitted Saturation* comments are dropped in some cases.

Assignments to bitfields

Assignment of Stateflow state IDs If Use bitfields in state machines = On and there is a multi-valued state variable, casts to `unsigned int` are now superfluous for Stateflow state IDs:

TargetLink ≤ 4.0	TargetLink 4.1
<code>SIBFS_Chart_a.Ca1_Chart_ns = (unsigned int) Ca5_OFF_id;</code>	<code>SIBFS_Chart_a.Ca1_Chart_ns = Ca5_OFF_id;</code>

In addition, the class default settings of Stateflow state IDs has changed. Stateflow state IDs become global macros in the generated code without the initial value being cast.

Bitfield semantic for genuine bitfields TargetLink evaluates whether a bitfield is a Boolean bitfield (`UseGlobalBitfieldsForBooleans = On`) or a genuine bitfield (`Base type = Bitfield`). For TargetLink versions ≤ 4.0 , a `bool` semantic is applied to both, Boolean bitfields and genuine bitfields. If required, `! = 0` is added. As of TargetLink version 4.1, a bitfield semantic is applied to genuine bitfields:

TargetLink ≤ 4.0	TargetLink 4.1
<code>BooleanBitfield = (Int16Var != 0)</code>	<code>BooleanBitfield = (Int16Var != 0)</code>
<code>GenuineBitfield = (Int16Var != 0)</code>	<code>GenuineBitfield = (Int16Var & 1)</code>

The bitfield semantic can also be applied to constants, for example:

```
GenuineBitfield = 3 & 1
```

Code optimization is feasible only if a constant equals the values 0 or 1.

Immediate assignment to bitfields An immediate assignment to a bitfield is generated if the right side is one of the following:

- A bitfield
- A bool expression
- A & 1 operation
- A Stateflow state ID macro

Note

For the assignment of floating-point or scaled operands to genuine bitfields, TargetLink no longer generates `!= 0` but returns an error.

Better Z/N values for rescaling

The algorithm for calculating the Z and N values has been improved. This leads to code that is either more precise or more efficient with calculations based on smaller bit widths:

TargetLink ≤ 4.0	TargetLink 4.1
<code>Sb1_PRODUCT = (Int16) (((Int32) (((Int32) 0p1) * ((Int32) 0p2)) << 2)) / 39;</code>	<code>Sb1_PRODUCT = (Int16) (((Int32) (((Int32) OP1) * ((Int32) OP2)) << 6)) / 625;</code>

Note

Parameter tolerance is now more frequently used to increase code efficiency. This means that a parameter tolerance greater than 0 can result in code that is less precise. If required, you can change this by reducing the parameter tolerance.

Comparison of UInt32 and Int32 variables

TargetLink now checks if the signed variable is negative and can avoid 64-bit macros when comparing UInt32 and Int32 variables:

TargetLink ≤ 4.0	TargetLink 4.1
<code>C_I64COPYU32(U32Var, I64Var_hi, I64Var_lo); C_I64COPYI32(I32Var, I64Var2_hi, I64Var2_lo); if(C__LE64(I64Var_hi, I64Var_lo, I64Var2_hi, I64Var2_lo))</code>	<p>For <=, <, and !=:</p> <pre>if(I32Var < 0 ((UInt32)I32Var <op> U32Var))</pre> <p>For >, >=, and ==:</p> <pre>if(I32Var >= 0 && ((UInt32)I32Var <op> U32Var))</pre>

As an effect, superfluous logical branches can be eliminated from the generated code by subsequent optimization.

Comparison of 64-bit variables

When comparing two 64-bit variables that both have an internally calculated worst-case range of \leq 32-bit, TargetLink now also considers the upper 32 bits. For example, `Int64Var` could become negative:

TargetLink \leq 4.0	TargetLink 4.1
<code>if(Int64Var_lo < Int64Var2_lo)</code>	<code>if(C_LT(Int64Var_hi, Int64Var_lo, Int64Var2_hi, Int64Var2_lo))</code>

This change is particularly relevant for the Discrete-Time Integrator block.

Elimination of intermediate variables

TargetLink can now perform intermediate variable elimination in such a way that a computation followed by an index selection is no longer performed for the complete width, only for the desired element.

TargetLink \leq 4.0	TargetLink 4.1
<code>VectorVar[<iteration range>] = expression(<iteration range>) ... = VectorVar[cnst]</code>	<code>... = expression(cnst)</code>

Note

The following conditions must be met:

- `VectorVar[cnst]` is the only usage of `VectorVar[<iteration range>]`.
- Index `cnst` is a subset of the iteration range. This is fulfilled if `<iteration range>` covers all elements.

Assignment blocks in For Iterator subsystems

If an Assignment block resides in an atomic subsystem that resides in a For Iterator subsystem, the following applies:

If the block property Omit dispensable initializations is set to off, the initialization via the Y0 signal is performed only during the first sample step. This is in accordance with the Simulink behavior.

Division-by-zero check for floating-points

TargetLink improves compliance with MISRA C:

No superfluous rescaling

TargetLink avoids irrelevant rescaling:

TargetLink \leq 4.0	TargetLink 4.1
<code>if (((Float32) Sa1_ScaledDenom) * 0.25F) != 0) { Sa1_OutPort1 = Sa1_F32Num_ / (((Float32) Sa1_ScaledDenom) * 0.25F); } else { ... }</code>	<code>if (Sa1_ScaledDenom != 0) { Sa1_OutPort = Sa1_F32Num / (((Float32) Sa1_ScaledDenom) * 0.25F); } else { ... }</code>

Auxiliary variables for offsets

TargetLink uses auxiliary variables:

TargetLink ≤ 4.0	TargetLink 4.1
<pre>if (((((Float32) Sa1_ScaledDenomWithOffset) * 0.25F) + 42.F) != 0) { Sa1_OutPort1 = Sa1_F32Num_ / (((((Float32) Sa1_ScaledDenomWithOffset) * 0.25F) + 42.F); } else { ...</pre>	<pre>Aux_F32 = (((Float32) Sa1_ScaledDenomWithOffset) * 0.25F) + 42.F; if (Aux_F32 != 0.F) { Sa1_OutPort1 = Sa1_F32Num_ / Aux_F32; } else { ...</pre>

Subtractions with unsigned result

For TargetLink ≤ 4.0, subtractions with an unsigned result were turned into additions: e.g., U8Var = U8Var2 + 255;.

For improved readability with TargetLink 4.1, a subtraction remains a subtraction if the constant fits the result type, U8Var = U8Var2 - 1;. Calculations might be performed in smaller types.

TargetLink ≤ 4.0	TargetLink 4.1
U8Var = U8Var2 + 255;	U8Var = U8Var2 - 1;

Predefined AUTOSAR variable classes

To suppress additional `#include t1Defines.h` directives, the predefined AUTOSAR variables were changed: When updating existing DD files, TargetLink sets the `UserName` property of predefined AUTOSAR VariableClass objects to `off` if they reference a DD AccessFunctionTemplate object.

This results in a different declaration or definition of variables that have these variable classes:

TargetLink ≤ 4.0	TargetLink 4.1
<code>extern EXPLICIT_IRV S16_LinPos Rte_Irv_Controller_LinPos</code>	<code>extern S16_LinPos Rte_Irv_Controller_LinPos</code>

This might cause missing `#include t1Defines.h` directives.

Code pattern for additions and subtractions without CTO

To increase efficiency and MISRA C compliance, the code pattern of additions and subtractions can change if all of the following conditions are true:

- The ExploitComputeThroughOverflow Code Generator option is set to `never`.
- The operands of the operation are unsigned.
- The operands of the operation have fewerl than 32 bits.

TargetLink ≤ 4.0	TargetLink 4.1
<code>UInt8Var = (UInt8)((Int16)UInt8Var + (Int16)UInt8Var)</code>	<code>UInt8Var = (UInt8)(UInt8Var + UInt8Var)</code>
<code>UInt8Var = (UInt8)((Int16)UInt8Var - (Int16)UInt8Var)</code>	<code>UInt8Var = (UInt8)(UInt8Var - UInt8Var)</code>

Definition of Data Store Memory block variables

To increase consistency and user control, the definitions of Data Store Memory block variables are now always made in the module that the step function of the subsystem that contains the Data Store Memory block is generated in.

Accordingly, the definitions of Data Store Memory block variables might occur in a different header file if all of the following conditions are true:

- The Data Store Memory block is placed in a subsystem that also contains a nested subsystem.
- The nested subsystem contains a Data Store Read or Data Store Write block that accesses the data store memory.
- The step functions of the subsystems are generated in different modules:
 - The step function of the subsystem containing the Data Store Memory block is generated into module A.
 - The step function of the subsystem containing the Data Store Read or Data Store Write block is generated into module B.

TargetLink ≤ 4.0	TargetLink 4.1
In certain modeling situations, TargetLink might have defined the block variable of the Data Store Memory block in module B.	TargetLink now always defines the block variable of the Data Store Memory block in module A.

Rounding of doubles in Float32 comparisons

TargetLink's rounding behavior changed for literal values in Float32 comparison. This is due to the following reasons:

- Code is easier to understand because the value is used in the same way as specified in the model.
- More user control:
 - You can specify the rounded value in the model.
 - You can use C compiler options to control the rounding and keep it consistent between TargetLink-generated production code and external code.

In the following example, the value `0.1` is compared with a variable whose data type is `Float32`:

```
0.1 > Float32Var
```

The following table shows TargetLink's rounding behavior:

TargetLink ≤ 4.0	TargetLink 4.1
<code>Sa1_Relational_Operator = 0.1000000015F > Sa1_F32In;</code>	<code>Sa1_Relational_Operator = 0.1F > Sa1_F32In;</code>

Double precision for Float32 values

For improved traceability between model and production code and also because of the above stated rounding control, TargetLink now generates literal values for `Float32` with double precision and by adding the `F` suffix. The following table shows an example of a `Float32` gain with `GainValue = pi`:

TargetLink ≤ 4.0	TargetLink 4.1
<code>Sa1_OutPort = Sa1_InPort * 3.141592654F;</code>	<code>Sa1_OutPort = Sa1_InPort * 3.1415926535897931F;</code>

This might cause more decimal places being printed in the generated code if the value was not exactly given in float/single precision in the model, as it was in TargetLink ≤ 4.0 .

Improved code efficiency

With TargetLink 4.1, code efficiency has been improved. This might also cause changes in the generated code, in comparison to older TargetLink versions.

Code Changes between TargetLink 3.5 and TargetLink 4.0

Code Changes between TargetLink 3.5 and TargetLink 4.0

ADDRESS_BY_PARAMETER access functions The type of the `_var` macro argument changed from `scalar` type to `pointer to scalar` type for vector variables. For details, refer to [Changes of ADDRESS_BY_PARAMETER Access Functions](#) on page 348.

Display of signal widths in block comments Block comments no longer provide information on the signal indices.

≤ TargetLink 3.5	TargetLink 4.0
<pre>for (Aux_S32 = 0; Aux_S32 < 5; Aux_S32++) { /* Gain: Subsystem/Gain [0..4] */ Sa1_Gain[Aux_S32] = Sa1_InPort[Aux_S32] * 3; /* Gain: Subsystem/Gain [5..9] */ Sa1_Gain[Aux_S32 + 5] = Sa1_InPort[Aux_S32 + 5] * 5; }</pre>	<pre>for (Aux_S32 = 0; Aux_S32 < 5; Aux_S32++) { /* Gain: Subsystem/Gain */ Sa1_Gain[Aux_S32] = Sa1_InPort[Aux_S32] * 3; Sa1_Gain[Aux_S32 + 5] = Sa1_InPort[Aux_S32 + 5] * 5; }</pre>

Block comments in if-else branches Block comments are now consistently placed at each statement:

≤ TargetLink 3.5	TargetLink 4.0
<pre>/* Switch: Subsystem/Switch1 */ if (c == 0) { ... <code> ... } } else { ... <code> ... }</pre>	<pre>/* Switch: Subsystem/Switch1 */ if (c == 0) { /* Switch: Subsystem/Switch1 */ ... <code> ... } else { /* Switch: Subsystem/Switch1 */ ... <code> ... }</pre>

Vector versus matrix variables If the Interpret as 1D flag is not set, TargetLink treats block variables (e.g., at the Constant block) as follows (in accordance to Simulink):

≤ TargetLink 3.5	TargetLink 4.0
Creates a vector variable.	Creates a matrix variable.

Muxed signals at Simulink Imports/Outports

If muxed signals are fed to an unenhanced in-/outport block of an atomic subsystem, TargetLink implements the whole block either as one vectorized function parameter or as one vectorized global variable:

≤ TargetLink 3.5	TargetLink 4.0
E.g., one function parameter for each muxed signal, or in some cases a struct: <pre>Void Sa2_Inner(Int16 Sa2_InPort[2], Int16 Sa2_InPort_a);</pre> — or — <pre>struct BS_IS_Sa1_BusImport { Int16 Sa1_Signal1[2]; Int16 Sa1_Signal2; } [...] Void Sa2_Inner(struct BS_IS_Sa1_BusImport * Sa2_InPort);</pre>	E.g., one function parameter for the whole port: <pre>Void Sa2_Inner(Int16 Sa2_InPort[3]);</pre>

Property inheritance and mapping struct components to variables

If a block inherits its data types via Inherit properties from a bus port block whose bus elements are mapped on one DD structure, and TargetLink cannot propagate the struct type, then TargetLink generates a separate variable for each DD structure component, for example:

≤ TargetLink 3.5	TargetLink 4.0
<pre>static Int16 X_Sa1_Unit_Delay[2] = { 0, 0 }; [...] Sa1_a = X_Sa1_Unit_Delay[0]; X_Sa1_Unit_Delay[0] = StructVar.a; Sa1_b = X_Sa1_Unit_Delay[1]; X_Sa1_Unit_Delay[1] = StructVar.b;</pre>	<pre>static Int16 X_Sa1_Unit_Delay = 0; static Int16 X_Sa1_Unit_Delay_a = 0; [...] Sa1_a = X_Sa1_Unit_Delay; X_Sa1_Unit_Delay = StructVar.a; Sa1_b = X_Sa1_Unit_Delay_a; X_Sa1_Unit_Delay_a = StructVar.b;</pre>

Pointer-to-const return type for RTE API functions

For Rte_IRead, Rte_Calprm, and Rte_CDData function calls using non-scalar data types, TargetLink uses the Pointer-to-const return type if the /Pool/Autosar/Config/UseRtePointerToConstForNonScalarReturnValues DD property is set to on.

≤ TargetLink 3.5	TargetLink 4.0
<pre>StructWSubStructsType * DE_StructWSubStructs; DE_StructWSubStructs = Rte_IRead_Swc_Run_ReceiverPort_DE_StructWSubStructs();</pre>	<pre>const StructWSubStructsType * DE_StructWSubStructs; DE_StructWSubStructs = Rte_IRead_Swc_Run_ReceiverPort_DE_StructWSubStructs();</pre>

Pointer-to-const function signatures for RTE API functions

IN function parameters that are read-only, and of non-scalar data types, provide an additional qualifier (const), if the /Pool/Autosar/Config/UseRtePointerToConstForInArguments DD property is set to on.

≤ TargetLink 3.5	TargetLink 4.0
<code>Rte <Fcn>(..., <Vector Address>, ...);</code>	<code>Rte <Fcn>(..., (const T*) <Vector Address>, ...);</code>

Pointers to AUTOSAR return variables A new name template is used for pointers to AUTOSAR pointer return variables in order to improve readability and to avoid global name collisions.

≤ TargetLink 3.5	TargetLink 4.0
<code>\$(DataElement)\$R</code>	Local pointer: <code>p_\$(DataElement)\$R</code> Global pointer: <code>p_\${I}_\$(DataElement)\$R</code>

Selector Block If TargetLink generates an implicit variable for the input, its name has now the suffix `_In`, which is more intuitive.

≤ TargetLink 3.5	TargetLink 4.0
<code>Int16 Sa1_Selector_a[20];</code>	<code>Int16 Sa1_Selector_In[20];</code>

Rate Limiter Block An implicit variable is generated for the block input only if it is required, i.e., when the input has a different data type or scaling, or when saturation is enabled for the output.

DisableArbitraryOptimizations Code Generator option `DisableArbitraryOptimizations` is renamed in `Allow64BitMultiplicationsForArbitraryScaled16BitOperands`, and the default setting was changed from `off` to `on`. In consequence, TargetLink generates more 64-bit operations, but the precision of simulation results improves as rescaling becomes more accurate.

Direct LUT block For Direct LUT block pattern with `Slices < LoopUnrollThreshold`, this TargetLink version applies the following code generation approach:

- Fewer implicit variables for the indices
- Changed sequence of code lines for improved efficiency
- More code comments for improved traceability

≤ TargetLink 3.5	TargetLink 4.0
<pre> Aux_U8 = C__U8SATI16_SATu(Sa1_Index1D_inport[0], 10); Aux_U8_a = C__U8SATI16_SATu(Sa1_Index1D_inport[1], 10); Aux_U8_b = C__U8SATI16_SATu(Sa1_Index1D_inport[2], 10); /* TargetLink outport: RootSystem/OutPort2 # combined # RootSystem/Direct Look-Up Table (n-D) Ext */ Sa1_OutPort2[0] = Sa1_TableData_inport[Aux_U8]; Sa1_OutPort2[1] = Sa1_TableData_inport[Aux_U8_a]; Sa1_OutPort2[2] = Sa1_TableData_inport[Aux_U8_b]; </pre>	<pre> Aux_U8 = C__U8SATI16_SATu(Sa1_Index1D_inport[0], 10); /* RootSystem/Direct Look-Up Table (n-D) Ext # combined # TargetLink outport: RootSystem/OutPort2 */ Sa1_OutPort2[0] = Sa1_TableData_inport[Aux_U8]; Aux_U8 = C__U8SATI16_SATu(Sa1_Index1D_inport[1], 10); /* RootSystem/Direct Look-Up Table (n-D) Ext # combined # TargetLink outport: RootSystem/OutPort2 */ Sa1_OutPort2[1] = Sa1_TableData_inport[Aux_U8]; Aux_U8 = C__U8SATI16_SATu(Sa1_Index1D_inport[2], 10); /* RootSystem/Direct Look-Up Table (n-D) Ext # combined # TargetLink outport: RootSystem/OutPort2 */ Sa1_OutPort2[2] = Sa1_TableData_inport[Aux_U8]; </pre>

Improved transparency of implicit AUX variables

For functions, implicit variables are initially always created at function level. Activating the Code Generator option `ReduceScopeOfVariablesOnlyDownToFunctionLevel` now affects all variables. If the options `ReduceScopeOfVariablesOnlyDownToFunctionLevel` and `Optimization` have default values, changes can only be observed for implicit vector variables. This can lead to changes in the number of `Aux_` variables and/or in their names.

≤ TargetLink 3.5	TargetLink 4.0
<pre> MyFunc() { ... if(...) { Int32 Aux_S32; Aux_S32 = IN * 42; MinOut = C__16FITI32_SAT(Aux_S32, ...); } } </pre>	<pre> MyFunc() { Int32 Aux_S32; ... if(...) { Aux_S32 = IN * 42; MinOut = C__16FITI32_SAT(Aux_S32, ...); } } </pre>

Actual parameters of function return values in incremental systems

Suppose the following:

- The outport of an incremental system is specified as a function return value, and
- The name of the actual parameter, i.e., the variable appearing in the function call statement, is not explicitly specified, or specified as `<FormalparameterName>$R`, and
- In the code generated for the incremental system, a different variable is used as the return value because the original variable has been deleted due to optimization.

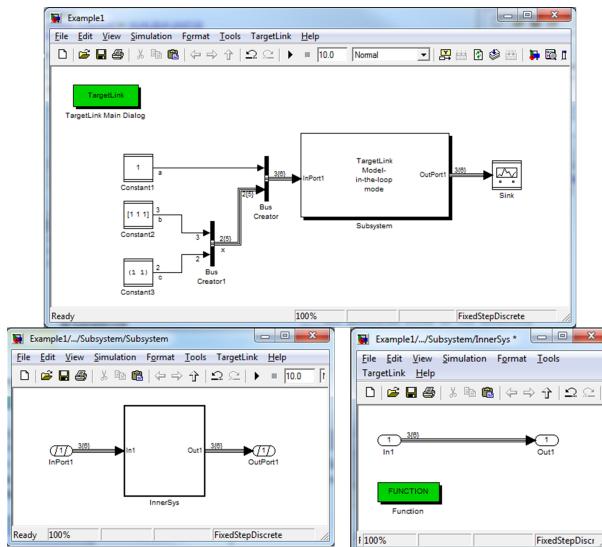
Hence, the actual parameter of the wrapping system is named as follows (independently of whether the system is incremental or not):

≤ TargetLink 3.5	TargetLink 4.0
<FormalparameterName>	<FormalparameterName>_a

Implicit interface variables for Simulink imports/outputs

For de-enhanced imports and outports of atomic subsystems, the code generated for implicit interface variables (IF variables) differs:

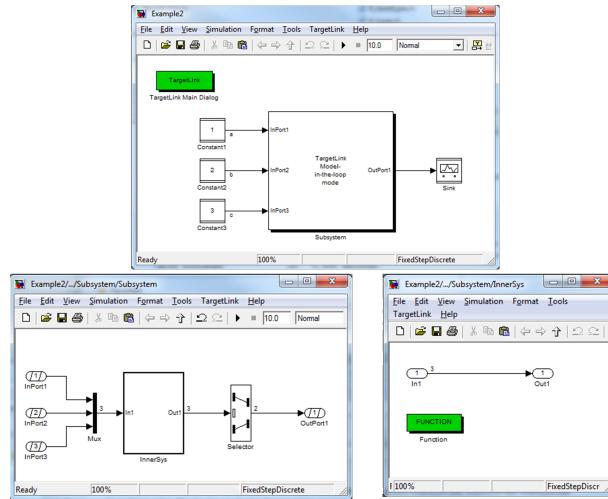
Bus signals



If a nested bus signal ($a; (b[3];c[2])$) is fed to a de-enhanced outport, then the IF variable for the outport is generated as follows:

≤ TargetLink 3.5	TargetLink 4.0
One IF variable for the whole bus signal: <pre>static Int16 IF_Sa2_Out1[6];</pre>	An IF variable for each bus element: <pre>static Int16 IF_Sa2_Out1; static Int16 IF_Sa2_Out1_a[3]; static Int16 IF_Sa2_Out1_b[2];</pre>

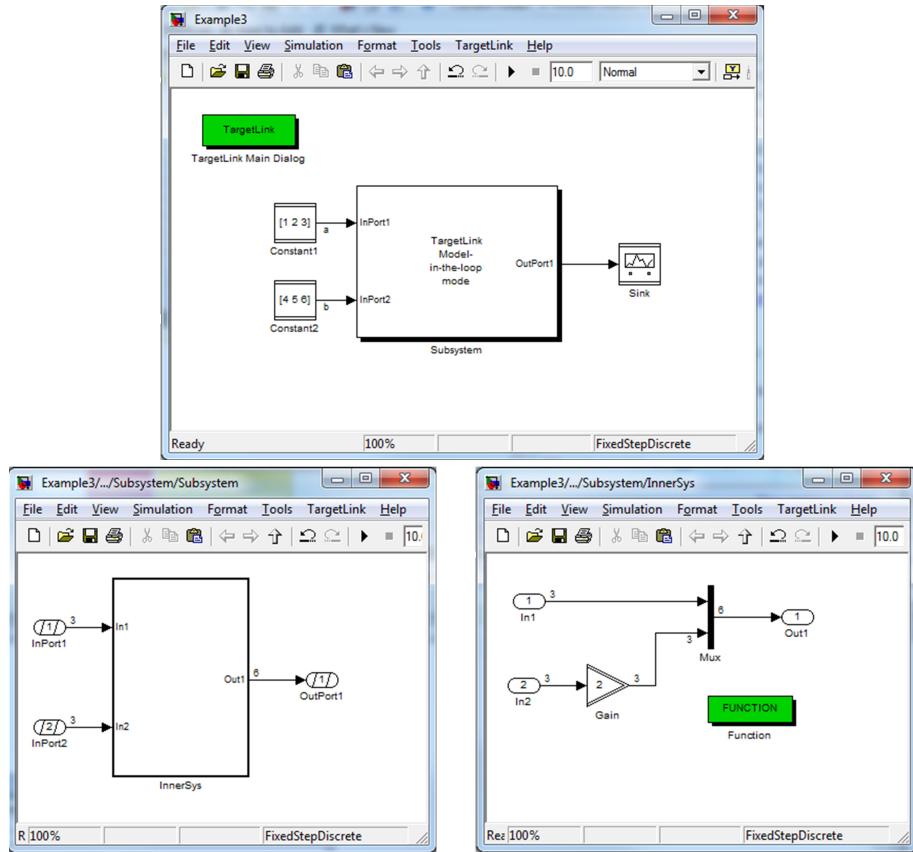
Muxed signals in combination with Selector block



If muxed scalar signals (a; b; c) are fed to a de-enhanced outport, then the IF variable for the outport is generated as follows:

≤ TargetLink 3.5	TargetLink 4.0
Two IF scalar variables: <pre>static Int16 IF_Sa2_Out1; static Int16 IF_Sa2_Out1_b;</pre>	One IF vector variable: <pre>static Int16 IF_Sa2_Out1[3];</pre>

Muxed signals arising from different blocks inside an atomic subsystem



If muxed vector signals ($a[3]$; $b[3]$) are fed to a de-enhanced outport, then the IF variable for the outport is generated as follows:

≤ TargetLink 3.5	TargetLink 4.0
One IF vector variable: <pre>static Int16 IF_Sa2_Gain[3]; static Int16 IF_Sa2_Out1[3];</pre>	Three IF scalar variables: <pre>static Int16 IF_Sa2_Gain[3]; static Int16 IF_Sa2_Out1; static Int16 IF_Sa2_Out1_a; static Int16 IF_Sa2_Out1_b;</pre>

No negative offsets in loop assignments

For assignments within **for** loops, TargetLink does not generate negative offsets:

≤ TargetLink 3.5	TargetLink 4.0
<pre>for(int Aux__a= 4; Aux__a < 10; Aux__a++) { Aux[Aux__a - 3] = In[Aux__a] }</pre>	<pre>for(int Aux__a= 1; Aux__a < 7; Aux__a++) { Aux[Aux__a] = In[Aux__a + 3] }</pre>

Mitigation of qualifier loss TargetLink introduces an implicit variable to mitigate qualifier loss if the qualifier is `const`:

≤ TargetLink 3.5	TargetLink 4.0
<pre>const volatile Int16 MyConstVolArray[3] = {0,0,0}; void Fcn(volatile Int16 * pFormalParam); Fcn((volatile Int16*) MyConstVolArray); /* Loss of const */</pre>	<pre>const volatile Int16 MyConstVolArray[3] = {0,0,0}; void Fcn(volatile Int16 * pFormalParam); Int16 MyConstVolArray_Aux[3]; for i = 0:2 MyConstVolArray_Aux[i] = MyConstVolArray[i]; Fcn((volatile Int16 *)MyConstVolArray_Aux);</pre>

When such an implicit variable is introduced, the message A17363 is emitted by the Code Generator to inform you about this possible inconsistent specification and solutions.

These implicit variables can occur in the following contexts:

- Functions of the FIR Filter block
- Subsystems that contain a TargetLink Function block
- Reused functions whose reuse structures were set to `const` via templates
- RTE calls resulting from model entities placed immediately before subsystems

The following contexts are not analyzed:

- Custom look-up script functions
- Custom code inputs
- Stateflow Extern C Functions

Bitfield casts

TargetLink now eliminates superfluous `unsigned int` casts in assignments involving bitfields, such as `BitfieldVar = BoolVar;` or `BitfieldVar = <BoolOperation>;`.

`<BoolOperation>` stands for one of the following, `>=`, `==`, `!=`, `<=`, `<`, `&&`, `||`, `!`

≤ TargetLink 3.5	TargetLink 4.0
<code>GIBFS_b_.X_Sb21_Memory_U = (unsigned int) T_SwHDIPushed_b;</code>	<code>GIBFS_b_.X_Sb21_Memory_U = T_SwHDIPushed_b;</code>

Removal of EfficientVectorHandling

The EfficientVectorHandling Code Generator option was removed from TargetLink 4.0.

TargetLink now always checks the dimensions of vectors and matrices against the `LoopUnrollThreshold` and generates a `for` loop for dimensions greater than or equal to the defined threshold. For details, refer to [Migration Aspects Regarding Code Generator Options](#) on page 350.

Moreover, Stateflow signals and Simulink signals are now treated equally with respect to loop generation. This is especially visible in code generated for Stateflow.

Code changes cannot be ruled out if EfficientVectorHandling was set to off or if the vector signal's dimension was smaller than the LoopUnrollThreshold:

≤ TargetLink 3.5	TargetLink 4.0
<pre> do { Ca1_a[idx1] = Ca1_b[idx1]; if (Ca1_b[idx1] < 0) { Ca1_a[idx1] = -Ca1_a[idx1]; } idx1++; } while (idx1 < 4); </pre>	<pre> Ca1_a[0] = Ca1_b[0]; Ca1_a[1] = Ca1_b[1]; Ca1_a[2] = Ca1_b[2]; Ca1_a[3] = Ca1_b[3]; if (Ca1_b[0] < 0) { Ca1_a[0] = -Ca1_a[0]; } if (Ca1_b[1] < 0) { Ca1_a[1] = -Ca1_a[1]; } if (Ca1_b[2] < 0) { Ca1_a[2] = -Ca1_a[2]; } if (Ca1_b[3] < 0) { Ca1_a[3] = -Ca1_a[3]; } </pre>

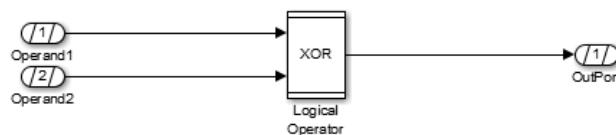
In TargetLink 4.0 **for** loops are generated according to the principles described in [Basics on Processing Vectors and Matrices](#) ([TargetLink Preparation and Simulation Guide](#)):

Dimension < LoopUnrollThreshold	Dimension ≥ LoopUnrollThreshold
<pre> Ca1_a[0] = Ca1_b[0]; Ca1_a[1] = Ca1_b[1]; Ca1_a[2] = Ca1_b[2]; Ca1_a[3] = Ca1_b[3]; if (Ca1_b[0] < 0) { Ca1_a[0] = -Ca1_a[0]; } if (Ca1_b[1] < 0) { Ca1_a[1] = -Ca1_a[1]; } if (Ca1_b[2] < 0) { Ca1_a[2] = -Ca1_a[2]; } if (Ca1_b[3] < 0) { Ca1_a[3] = -Ca1_a[3]; } </pre>	<pre> for (Aux_S32 = 0; Aux_S32 < 4; Aux_S32++) { Ca1_a[Aux_S32] = Ca1_b[Aux_S32]; if (Ca1_b[Aux_S32] < 0) { Ca1_a[Aux_S32] = -Ca1_a[Aux_S32]; } } </pre>

Logical Operator block (exclusive OR)

The code generated for Logical Operator blocks whose Operator is set to XOR changed.

TargetLink now generates the logical expression != instead of the arithmetic expression ^ to consistently separate logical and arithmetic expressions. This also means improved MISRA C compliance.

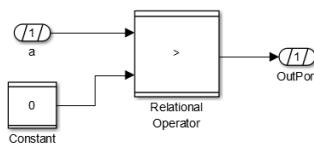


≤ TargetLink 3.5	TargetLink 4.0
<pre>/* Logical: Logical Operator TL_Root/Logical Operator */ block_out = (!(Operand1)) ^ !(Operand2);</pre>	<pre>/* Logical: TL_Root/Logical Operator */ block_out = (Operand1 != 0) != (Operand2 != 0);</pre>

Rescheduling of state updates TargetLink's analysis for state update rescheduling has improved. This can result in a different order of state updates in the code.

Relations and logical operations TargetLink's code pattern and optimization behavior for Logical Operator and Relational Operator blocks changed.

Example



Optimization = OFF

≤ TargetLink 3.5	TargetLink 4.0
<pre>if (a > 0) { Sa1_Relational_Operator = 1; } else { Sa1_Relational_Operator = 0; }</pre>	(AssignmentOfConditions > 0) <pre>Sa1_Relational_Operator = (a > 0);</pre>

Optimization = ON

≤ TargetLink 3.5	TargetLink 4.0
(NoAssignmentOfBooleanIfThenElse=OFF) (default) Sa1_Relational_Operator = (a > 0); — or — (dependent on the data type of a) Sa1_Logical_Operator = (a != 0);	<pre>Sa1_Relational_Operator = (a > 0);</pre>

This change also affects logical or relational operations from other sources. By default, TargetLink now only transforms the non-optimized control flow version into an assignment if the assignee is a Boolean or bitfield variable.

```

Sa1_Logical_Operator1 = b && c;
if (Sa1_Logical_Operator1 != 0) {
    Sa1_BoolSwitch = 0;
} else {
    Sa1_BoolSwitch = 1;
}

Sa1_Logical_Operator2 = d <= 0;
if (Sa1_Logical_Operator2 != 0) {
    Sa1_Int8Switch = 0;
} else {
    Sa1_Int8Switch = 1;
}

Sa1_Logical_Operator3 = e != 0;
if (Sa1_Logical_Operator3 != 0) {
    Sa1_FloatSwitch = 0.F;
} else {
    Sa1_FloatSwitch = 1.F;
}

```

≤ TargetLink 3.5	TargetLink 4.0
(NoAssignmentOfBooleanIfThenElse=OFF (default))	(AssignmentOfConditions=2 (BooleanOutputsOnly; default))
<pre> Sa1_BoolSwitch = (!b !c); Sa1_Int8Switch = (d > 0); Sa1_FloatSwitch = (e == 0); </pre>	<pre> Sa1_BoolSwitch = (!b !c); if (d <= 0) { Sa1_Int8Switch = 0; } else { Sa1_Int8Switch = 1; } if (e != 0) { Sa1_FloatSwitch = 0.F; } else { Sa1_FloatSwitch = 1.F; } </pre>
	(AssignmentOfConditions=3 (All Outputs))
	<pre> Sa1_BoolSwitch = (!b !c); Sa1_Int8Switch = (Int8) (d > 0); Sa1_FloatSwitch = (Float32) (e == 0); </pre>

This change in the initial code pattern can lead to different optimization results in the generated code, e.g., assignments involving a logical or relational operation take place later than with TargetLink ≤3.5.

RDI macro definitions

TargetLink now places brackets around the initial values that receive a cast. This also means improved MISRA C compliance.

≤ TargetLink 3.5	TargetLink 4.0
<pre>#define PIM_RDI (sint32 (*)[2]) Rte_Pim_PIM_Matrix3x2()</pre>	<pre>#define PIM_RDI ((sint32 (*)[2]) Rte_Pim_PIM_Matrix3x2())</pre>

Improved code efficiency changes

With TargetLink 4.0, code efficiency was improved, which can also lead to changes in the generated code compared to older TargetLink versions.

Code Changes between TargetLink 3.4 and TargetLink 3.5

Code Changes between TargetLink 3.4 and TargetLink 3.5

Rounding of user-specified min/max values

TargetLink's rounding behavior for user-specified min/max values that cannot exactly be expressed in their corresponding scaling has changed with respect to range propagation.

That means that if a min or max value is specified for a block output/parameter or state and if that value cannot be exactly represented in the scaling of the output/parameter/state, a value rounded to the next representable value will be propagated to the connected block(s). This can result in:

- Additional saturation code
- Additional relational operations
- Changes in scale transformation operations (changing an operand with scaling A to scaling B)
- Casts to wider/different types compared to previous TargetLink versions

Note

The easiest and recommended way to avoid this issue is to always specify min/max values that can exactly be represented in the associated scaling.

Background In previous TargetLink versions, the floor() function was applied to the propagated value.

Example Consider the following TargetLink block output:

```
LSB    =  0.5
Offset =  0.0
Min    = -2.5
Max    =  2.4
```

The **Min** value can exactly be represented by a fixed-point value of -5 in the given scaling:

```
FxpValue = (FlpValue - Offset)/LSB => -5 = (-2.5 - 0.0)/0.5
```

But to represent the **Max** value its fixed-point representation would have to be 4.8:

```
FxpValue = (FlpValue - Offset)/LSB => 4.8 = (2.4 - 0.0)/0.5
```

Since the fixed-point value needs to be an integer, either 5.0 or 4.0 has to be chosen.

By applying the floor functionality, previous TargetLink versions chose 4.0, which corresponds to the floating point value of 2.0 (4.0 * 0.5).

TargetLink 3.5 performs rounding and the resulting floating-point value will be 2.5. Note that this rounding behavior might occur unexpectedly. Many floating-

point values, such as `0.1`, cannot exactly be represented using double. There is always a very small deviation and the double value is slightly smaller or greater than the expected value (`0.1`).

Unary minus in Stateflow

For unary-minus operations specified in a Stateflow expression that contains at least one operand with a scaling (`LSB != 1.0` and/or `Offset != 0.0`) and no floating-point operand, the order of scale transformation operations (shift, multiplication, division) and the unary-minus itself has changed.

TargetLink 3.5	\leq TargetLink 3.4
<code>-((Int16)(x >> 5))</code>	<code>((Int16)(-x)) >> 5</code>

Optimization of runnables

Runnables that are not used in the code are no longer removed by TargetLink's code optimization procedures.

Discrete Time Integrator block

TargetLink now considers the full range of limits specified at Discrete-Time Integrator blocks, if they reference a DD VariableClass object with one of the following specifications:

- The Info property is set to a value other than `none` or `readonly`.
- The Alias property is set to `on`.
- The Macro property is set to `on` and the Storage property is set to `extern`.

This range is needed to define an adequate range and data type for the block's state variable. So its range and data type might change in the generated code compared to code from previous TargetLink versions.

Discrete Filter, Discrete Transfer Function and Discrete State Space blocks

If the Keep vector structure checkbox (Coefficients page) of Discrete Filter/Discrete Transfer Fcn blocks or the Keep matrix structure checkbox (Matrices page) of Discrete State-Space blocks is not selected, TargetLink creates a structure for the coefficients/matrices.

TargetLink now carries the min/max values of coefficient vectors/matrices specified in the blocks's dialogs to the structure's components. This can yield calculations in smaller bit-width.

Conditional operator (?:)

TargetLink now brackets the arguments of the conditional operator (`? :`), if the argument is an operation. This increases MISRA C compliance.

TargetLink 3.5	\leq TargetLink 3.4
<code>(a > 3) ? 1 : 2</code>	<code>a > 3 ? 1 : 2</code>

Inlining of restart functions

TargetLink now inlines restart functions occurring within the *same* file.

TargetLink 3.5	≤ TargetLink 3.4
<pre>Void RESTART_main(Void) { myGlobalVar1 = 0; myGlobalVar2 = 0; }</pre>	<pre>Void RESTART_f1(Void) { myGlobalVar1 = 0; } Void RESTART_f2(Void) { myGlobalVar2 = 0; } Void RESTART_main(Void) { RESTART_f1(); RESTART_f2(); }</pre>

Copy propagation and block's state variables

Elimination of unnecessary state variables TargetLink can eliminate variables used as states if they can be replaced by a variable with static storage duration and equal initial value. With TargetLink 3.5, this optimization is extended:

- TargetLink makes use of the [ANSI C](#) default initialization to zero of static storage duration variables without explicit initial value. In order to document this assumption, the replacement variable gets an explicit initialization to zero; if this is not possible because there is no initialization method at the variable class, then no optimization takes place.
- Block state variables now can be eliminated; the respective code situations can occur only due to code efficiency improvements in TargetLink 3.4.

Initialization of static variables When the UtilizeZeroInitializationOfStaticStorageDurationVariables Code Generator Option is set to **on** (default), variables with static duration are initialized with **0**. This especially affects the elimination of variables generated for the outputs of conditionally executed subsystems, whose values are written to global OutPort variables.

TargetLink 3.5	≤ TargetLink 3.4
<pre>Int16 Out = 0; foo() { ... if (cond) { ... Out = ...; } }</pre>	<pre>Int16 Out; foo() { static Int16 X = 0; ... if (cond) { ... X = ...; } Out = X; }</pre>

AUTOSAR operation calls	<p>The production code can contain statements such as <code>foo = Rte_Call_<Operation>()</code> that were not contained in the code of prior versions.</p> <p>Explanation: Operation calls can safely be removed by TargetLink's optimization procedures, if the following conditions hold:</p> <ul style="list-style-type: none">▪ Their return values are not used in the code.▪ The operation call is known to have no states or side effects. <p>For operation calls explicitly modeled as server runnables in a Code Generation Unit, TargetLink can determine whether the operation call has states or side effect from the operation call's implementation.</p> <p>Operation calls whose implementation is opaque to TargetLink (not explicitly modeled as server runnables), can be removed by TargetLink's optimization procedures only if the following conditions hold:</p> <ul style="list-style-type: none">▪ Their return values are not used in the code.▪ Their DD Operation object has its <code>NoStatesOrSideEffects</code> property set to <code>on</code>.▪ Their DD <code>NoDataFlowWithOtherOperations</code> property is set to <code>on</code>, either directly or via the <code>AssumeOperationCallsHaveNoUnknownDataFlow</code> Code Generator option.
(User-)casts in Stateflow	<p>User-specified casts in Stateflow are treated differently. This can yield differences in code in the following contexts:</p> <ul style="list-style-type: none">▪ For casting MinMax/Abs calls, the conditional operator (?) is used.▪ Compute through overflow behavior (unsigned casts) changed with respect to sequences of additions.▪ Integer operations with floating-point constants:<ul style="list-style-type: none">▪ If a suboperation contains a floating-point operand.▪ If this operand is part of a cast or shift operation.▪ Index expressions of type float (additional integer cast).
No scope reduction of unaccessed variables	<p>Variables whose DD Optimization property is set to <code>SCOPE_REDUCIBLE</code> but not <code>ERASABLE</code> are no longer reduced in scope, if they are not read/written in functions or initializers.</p> <p>Typically, these variables now have global scope instead of static global scope.</p>

Code Changes between TargetLink 3.3 and TargetLink 3.4

Code Changes between TargetLink 3.3 and TargetLink 3.4

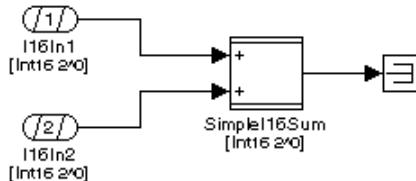
Compute-through-overflow

The following tables list code changes that are due to the TargetLink ExploitComputeThroughOverflow Code Generator option setting. The production code pattern presented in the TargetLink 3.4 column differ with respect to the Code Generator setting's value. Annotations are italicized.

Note

The representation of a constant value in an operation using compute-through-overflow can be different from its representation in an operation without the instrumentation of overflows.

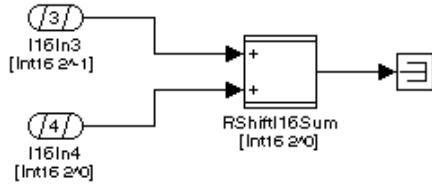
Sum with operands sharing the output's scaling and data type



The following table shows the code changes for a Sum block taking as input two Int16s that have the same scaling and data type as the output.

TargetLink 3.4	≤ TargetLink 3.3
ExploitComputeThroughOverflow = Optimized	<pre>Sa1_SimpleI16Sum = (Int16) (Sa1_I16In1 + Sa1_I16In2);</pre>
No CTO casts are made, because analysis of context shows that no overflow can occur.	<p>The CTO casts (UInt16) are made to guarantee defined overflow behavior.</p>
ExploitComputeThroughOverflow = Never	
<pre>Sa1_SimpleI16Sum = (Int16) (Sa1_I16In1 + Sa1_I16In2);</pre>	
No CTO casts are made, because CTO is forbidden by Code Generator option.	
ExploitComputeThroughOverflow = Always	
<pre>Sa1_SimpleI16Sum = (Int16) (((UInt16) Sa1_I16In1) + ((UInt16) Sa1_I16In2));</pre>	
Same code pattern as in TargetLink 3.3.	

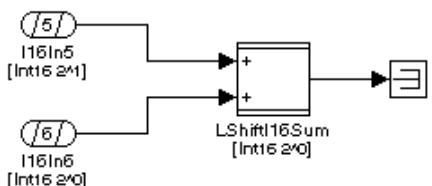
Sum with one operand differently scaled as output, implying a right shift



The following table shows the code changes for a Sum block taking two `Int16`s as input. The scaling of `I16In3` differs from the output's scaling, which implies a right shift operation:

TargetLink 3.4	≤ TargetLink 3.3
<pre>ExploitComputeThroughOverflow = Optimized Sa1_RShiftI16Sum = (Int16) (((Int16) (Sa1_I16In3 >> 1)) + Sa1_I16In4); No CTO casts are made, because analysis of context shows that no overflow can occur. The second Int16 cast is the result cast of the right shift operation.</pre> <pre>ExploitComputeThroughOverflow = Never Sa1_RShiftI16Sum = (Int16) (((Int16) (Sa1_I16In3 >> 1)) + Sa1_I16In4); No CTO casts are made, CTO is forbidden by Code Generator option. The second Int16 cast is the result cast of the right shift operation.</pre> <pre>ExploitComputeThroughOverflow = Always Sa1_RShiftI16Sum = (Int16) (((UInt16) (Int16) (Sa1_I16In3 >> 1)) + ((UInt16) Sa1_I16In4)); Same code pattern as in TargetLink 3.3.</pre>	<pre>Sa1_RShiftI16Sum = (Int16) (((UInt16) (Int16) (Sa1_I16In3 >> 1)) + ((UInt16) Sa1_I16In4));</pre> <p>The CTO casts (<code>UInt16</code>) are made to guarantee defined overflow behavior. The second <code>Int16</code> cast is the result cast of the right shift operation.</p>

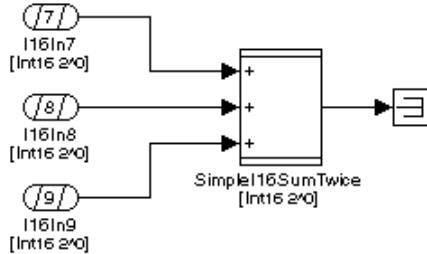
Sum with one operand differently scaled as output, implying a left shift



The following table shows the code changes for a Sum block taking two `Int16`s as input. The scaling of `I16In5` differs from the output's scaling, which implies a left shift operation:

TargetLink 3.4	≤ TargetLink 3.3
<pre>ExploitComputeThroughOverflow = Optimized</pre> <pre>Sa1_LShiftI16Sum = (Int16) (((UInt16) (Sa1_I16In5 << 1)) + ((UInt16) Sa1_I16In6));</pre> <p>CTO casts are made, because analysis of context shows that the first input does not fit in an 16-bit integer data type. The Int16 cast is omitted because suboperations of additions and subtractions are given an unsigned type, if an overflow could occur (refer to Unsigned suboperations for addition/subtraction on page 554).</p>	<pre>Sa1_LShiftI16Sum = (Int16) (((UInt16) (Int16) (Sa1_I16In5 << 1)) + ((UInt16) Sa1_I16In6));</pre> <p>The CTO casts (UInt16) are made to guarantee defined overflow behavior. The second Int16 cast is the result cast of the left shift operation.</p>
<pre>ExploitComputeThroughOverflow = Never</pre> <pre>Sa1_LShiftI16Sum = (Int16) (((Int32) (((Int32) Sa1_I16In5) << 1)) + ((Int32) Sa1_I16In6));</pre> <p>Because CTO is forbidden, calculations are performed in Int32 to avoid overflows.</p>	
<pre>ExploitComputeThroughOverflow = Always</pre> <pre>Sa1_LShiftI16Sum = (Int16) (((UInt16) (Sa1_I16In5 << 1)) + ((UInt16) Sa1_I16In6));</pre> <p>Same code pattern as in TargetLink 3.3.</p>	

Sum with three inputs without assign arithmetic for accumulations



The following table shows the code changes for a Sum block taking three Int16s as input sharing the same scaling as the output. Note that the DoNotUseAssignArithmeticForAccumulation Code Generator option has been enabled to suppress assign arithmetic for accumulations:

TargetLink 3.4	\leq TargetLink 3.3
<pre>ExploitComputeThroughOverflow = Optimized Sa1_SimpleI16SumTwice = (Int16) (((UInt16) (((UInt16) Sa1_I16In7) + ((UInt16) Sa1_I16In8))) + ((UInt16) Sa1_I16In9)); CTO casts are made, because analysis of context shows that the result of the first addition does not fit in an 16-bit integer data type. The second Int16 cast is omitted because suboperations of additions and subtractions are given an unsigned type if an overflow could occur (refer to Unsigned suboperations for addition/subtraction on page 554). ExploitComputeThroughOverflow = Never Sa1_SimpleI16SumTwice = (Int16) (((Int32) (((Int32) Sa1_I16In7) + ((Int32) Sa1_I16In8))) + ((Int32) Sa1_I16In9)); Because CTO is forbidden, calculations are performed in Int32 to avoid overflows. ExploitComputeThroughOverflow = Always Sa1_SimpleI16SumTwice = (Int16) (((UInt16) (((UInt16) Sa1_I16In7) + ((UInt16) Sa1_I16In8))) + ((UInt16) Sa1_I16In9)); Same code pattern as in TargetLink 3.3.</pre>	<pre>Sa1_SimpleI16SumTwice = (Int16) (((UInt16) (Int16) (((UInt16) Sa1_I16In7) + ((UInt16) Sa1_I16In8))) + ((UInt16) Sa1_I16In9)); The CTO casts (UInt16) are made to guarantee defined overflow behavior.</pre>

Modified code pattern for addition/subtraction

If an addition or subtraction meets the following conditions, the resulting code guarantees a defined overflow behavior.

- The result would allow a smaller data type than the data type specified.
- An operand does not fit this smaller data type.

Code example Suppose an addition is defined as follows:

```
I32Out = I32In1 + I32In2
```

The first operand also had the user-defined limits **Max** = 70000 and **Min** = 60000, and the second had **Max** = -50000 and **Min** = -70000. Thus, the result would always fit **Int16**. However, both operands do not fit **Int16** (overflow). In consequence, the operands are cast to **UInt16** so that the overflow is defined.

The code would be generated as follows:

TargetLink 3.4	\leq TargetLink 3.3
<pre>I32Out = (Int32) (Int16) (((UInt16) I32In1) + ((UInt16) I32In2));</pre>	<pre>I32Out = (Int32) (Int16) (((Int16) I32In1) + ((Int16) I32In2));</pre>

Unsigned suboperations for addition/subtraction

Operations that are operands themselves in addition/subtraction are assigned an unsigned data type, if they could overflow and if they apply Compute-Through-Overflow (refer to [Sum with one operand differently scaled as output, implying a left shift](#) on page 552).

Code example Suppose an addition is defined as follows:

```
I16Out = I16In1 + I16In2
```

with LSB of I16Out, I16In2 = 2^-1, and LSB of I16In1 = 2^0

The code would be generated as follows:

TargetLink 3.4	≤ TargetLink 3.3
I16Out = (Int16)((UInt16)(I16In1 << 1) + (UInt16) I16In2)	I16Out = (Int16)((UInt16)((Int16)(I16In1 << 1)) + (UInt16) I16In2)

Deletion of unused function parameters

The following applies to function parameters that are specified for chart functions and graphical functions and are propagated to all the subfunctions (Stateflow state and auxiliary functions):

Function parameters of these subfunctions that are not used are deleted although the Optimization property of their variable class in the TargetLink Data Dictionary is not set to ERASABLE.

Casts in Stateflow shift operations

Shift operations in Stateflow make use of additional casts, if the operand meets the following conditions:

- Operand uses a TargetLink data type different from the Stateflow data type.
- Operand is not scaled.

Casts of pointers (function calls)

With this version of TargetLink the cast behavior of pointers has changed as follows:

- If a function parameter has a type prefix that is interpreted as an address qualifier (e.g., _far), the type prefix is evaluated to check whether casts are necessary. In older TargetLink versions, a cast was applied for any type prefix.
- If a function parameter is a vector and the required and the actual type qualifiers differ, the actual type qualifier is cast to the required type qualifier. In older TargetLink versions, no such cast was applied.

Type guard identifiers (MISRA)

Type guard identifiers are now output without leading or trailing underscores, which is in accordance with MISRA. For example:

TargetLink 3.4	≤ TargetLink 3.3
#ifndef MyType_TYPE #define MyType_TYPE typedef MyType; #endif /* MyType_TYPE */	#ifndef _MyType_TYPE_ #define _MyType_TYPE_ typedef MyType; #endif /* _MyType_TYPE_ */

Macro initialization (MISRA)

If a macro has a negative initialization value, the value is wrapped in parentheses. This is in accordance with MISRA. For example:

TargetLink 3.4	≤ TargetLink 3.3
#define MACRO ((Int8) -1) #define SECOND_MACRO (-10)	#define MACRO (Int8) -1 #define SECOND_MACRO -10

&& and || operations (MISRA) For function calls, access to vector components via [] operator, and access to the elements of structs, the following new rule applies, which is in accordance with MISRA: The arguments of && and || operations are wrapped in parentheses, for example:

TargetLink 3.4	≤ TargetLink 3.3
(A()) && b c (D[1]) (Struct.e) f	A() && b c D[1] Struct.e f

Relay block For the TargetLink Relay block, the Switch On condition is now checked first to mimic the Simulink behavior in case the SwitchOn and SwitchOff values are equal.

Unit Delay block If the block output has saturation enabled, a block output variable is always generated. Additionally, the assignment output = state is saturated.

In addition, the block output variable is always read by the succeeding block, if it meets one of the following conditions:

- Block output variable is logged.
- Block output variable is referenced from the TargetLink Data Dictionary.
- Block output variable has a user-defined variable class.
- Block output variable has saturation enabled.

Changes in Future TargetLink Versions

Where to go from here

Information in this section

Features to Be Discontinued.....	557
API Functions to Be Discontinued.....	558
Deprecated Code Generator Options.....	558

Features to Be Discontinued

RTOS/OSEK code generation mode	Support for the TargetLink RTOS/OSEK code generation modes will be discontinued in a future TargetLink version.
Target Optimizations Module (TOM)	Support for the TargetLink Target Optimizations Module (TOM) will be discontinued in a future TargetLink version.
Clean code and Do not log anything	<p>Variables selected for logging cannot be fully optimized. When generating code with the Global logging option Do not log anything or Log according to block data, TargetLink does not fully optimize the code to facilitate testing. This means the code differs only with regard to the log macros. This contrasts the Clean code checkbox on the Code Generation page of the TargetLink Main Dialog block, which always activates full code optimization.</p> <p>The special Do not log anything behavior will be removed in a future TargetLink version.</p>

Simulink classic initialization mode	Support for the Simulink classic initialization mode will be discontinued in a future TargetLink version.
Dynamic components	Support for specifying dynamic components for DD Variable objects will be discontinued in a future TargetLink version.
Automatic interpretation of Boolean	The automatic interpretation of certain integer data types as Boolean by TargetLink will be discontinued in a future TargetLink version.
Unit Delay Reset Enabled	The TargetLink support of the Unit Delay Reset Enabled block will be discontinued in a future TargetLink version.
Model change detection via checksum	The TargetLink support of model change detection via checksum will be discontinued in a future TargetLink version.
Sample blocks	The TargetLink support of Sample blocks will be discontinued in a future TargetLink version.

API Functions to Be Discontinued

Discontinued API functions The following API functions are deprecated and will be removed in a future TargetLink version:

Function	Deprecated Since	Replacement Function
<code>tl_compare_fcn_signature</code>	TargetLink 5.0	-

Deprecated Code Generator Options

Deprecated Code Generator options The following Code Generator options are deprecated and will be removed in future TargetLink versions:

- [AllowStructAssignments](#) ([TargetLink Model Element Reference](#))

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.....	560
A.....	561
B.....	564
C.....	565
D.....	568
E.....	570
F.....	571
G.....	572
I.....	572
L.....	574
M.....	575
N.....	577
O.....	578
P.....	579
R.....	580
S.....	582
T.....	584
U.....	586
V.....	586
W.....	587

Numerics

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

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

A

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

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

See also [read/write access function](#) and [variable access function](#).

Acknowledgment Notification from the [RTE](#) that a [data element](#) or an [event message](#) have been transmitted.

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

See also [activation reason](#).

Activation reason The [activating RTE event](#) that actually triggered the runnable.

Activation reasons can group several RTE events.

Active page pointer A pointer to a [data page](#). The page referenced by the pointer is the active page whose values can be changed with a calibration tool.

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

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

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

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

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

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

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

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

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

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

See also [implementation data type \(IDT\)](#).

Application layer The topmost layer of the [ECU software](#). The application layer holds the functionality of the [ECU software](#) and consists of [atomic software components \(atomic SWCs\)](#).

ARA adapter code Adapter code that connects [Adaptive AUTOSAR behavior code](#) with the Adaptive AUTOSAR API or other parts of an adaptive application.

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

Artifact A file generated by TargetLink:

- Code coverage report files
- Code generation report files
- [Metadata files](#)
- Model-linked code view files
- [Production code](#) files
- Simulation application object files
- Simulation frame code files
- [Stub code](#) files

Artifact location A folder in the file system that contains an [artifact](#). This location is specified relatively to a [project folder](#).

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

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

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

See also [operation result provider subsystem](#).

Asynchronous server call returns event An [RTE event](#) that specifies whether to start or continue the execution of a [Runnable](#) after the execution of a [server Runnable](#) is finished.

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

Atomic software component instance An [atomic software component \(atomic SWC\)](#) that is actually used in a controller model.

AUTOSAR Abbreviation of *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 Import, 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., variantized at code generation time). For example, if the Type property of a variable has the two variants Int16 and Float32, you can generate either source code for a fixed-point ECU with one variant, or floating-point code with the other.

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

See also [? vendor mode](#).

Compiler inlining The process of replacing a function call with the code of the function body during compilation by the C compiler via [? inline expansion](#).

This reduces the function call overhead and enables further optimizations at the potential cost of larger [code size](#).

Composition A structuring element in the [application layer](#). A composition consists of [software components](#) and their interconnections via [ports](#).

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

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

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

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

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

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

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

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

Min/Max values are entered, the [implemented range](#) limits are used during production code generation.

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

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

Container Manager A tool for handling [containers](#).

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

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

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

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

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

D

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

Data page A structure containing all of the [calibratable variables](#) that are generated during code generation.

Data prototype The generic term for one of the following:

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

Data receive error event An [RTE event](#) that specifies to start or continue the execution of a [Runnable](#) related to receiver errors.

Data received event An [RTE event](#) that specifies whether to start or continue the execution of a [Runnable](#) after a [Data element](#) is received by a [Receiver port](#) or [Sender-receiver port](#).

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

Data send completed event An [RTE event](#) that specifies whether to start or continue the execution of a [Runnable](#) related to a sender [Acknowledgment](#).

Data transformation A transformation of the data of inter-ECU communication, such as end-to-end protection or serialization, that is managed by the [RTE](#) via [Transformers](#).

Data type map Defines a mapping between [Implementation data types](#) (represented in TargetLink by DD Typedef objects) and [Application data types](#).

Data type mapping set Summarizes all the [Data type maps](#) and [Mode request type maps](#) of a [Software component \(SWC\)](#).

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

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

DD child object The [DD object](#) below another DD object in the [DD object tree](#).

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

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

DD object Data item in the Data Dictionary that can contain [DD child objects](#) and DD properties.

DD object tree The tree that arranges all [DD objects](#) according to the [DD data model](#).

DD project file A file containing the [DD objects](#) of a [DD workspace](#).

DD root object The topmost [DD object](#) of the [DD workspace](#).

DD subtree A part of the [DD object tree](#) containing a [DD object](#) and all its descendants.

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

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

Direct reuse The Code Generator adds the [instance-specific variables](#) to the reuse structure as leaf struct components.

E

ECU Abbreviation of *electronic control unit*.

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

ECU State Manager A piece of software that manages [modes](#). An ECU state manager is part of the [basic software](#).

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

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

Enumeration constant An enumeration constant defines the name for an [enumerated value](#).

Enumeration data type A data type with a specific name, a set of named [enumerated values](#) and a [default enumeration constant](#).

Enumeration value An enumeration value defines the integer value for an [enumerated value](#).

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

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

ExchangeableWidth A DD object that defines [code variants](#) or improves code readability by using macros for signal widths.

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

Executable application The generic term for [offline simulation applications](#) and [real-time applications](#).

Explicit communication A communication mode in [Classic AUTOSAR](#). The data is exchanged whenever data is required or provided.

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

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

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

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

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](#).

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.

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

M

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

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

MATLAB code elements MATLAB code elements include [MATLAB local functions](#) and [local MATLAB variables](#). MATLAB code elements are not available in the Simulink Model Explorer or the Property Manager.

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

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

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

Matrix signal Collective term for 2-D signals implemented as [matrix variable](#) in [production code](#).

Matrix variable Collective term for 2-D arrays in [production code](#) that implement 2-D signals.

Measurement Viewing and analyzing the time traces of [calibration parameters](#) and [measurement variables](#), for example, to observe the effects of ECU parameter changes.

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

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

Memory mapping The process of mapping variables and functions to different [memory sections](#).

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

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

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

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

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

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

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

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

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

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

Mode An operating state of an [ECU](#), a single functional unit, etc..

Mode declaration group Contains the possible [operating states](#), for example, of an [ECU](#) or a single functional unit.

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

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

Mode switch event An [RTE event](#) that specifies to start or continue the execution of a [Runnable](#) as a result of a [mode change](#).

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

Model component A model-based [code generation unit \(CGU\)](#).

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

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

Model port block A block in [ConfigurationDesk](#) that has one or more [model ports](#). It is used to connect the [behavior model](#) in [ConfigurationDesk](#).

Model port variable A DD Variable object that represents a [model port](#) of a [behavior model](#) in [ConfigurationDesk](#).

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

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

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

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

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

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

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 Import/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.

tllib 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.

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](#).

A

adaptation
 Simulink configuration parameters 353
 address operator
 brackets 384
 AUTOSAR
 TargetLink related
 migration 360, 365
 TargetLink-related
 migration 338

C

changed Code Generator options 380
 changes
 TargetLink API 368, 376, 388
 code changes 245, 274
 Code Generator options
 changed options 370
 migration aspects 370
 obsolete options 370
 comments
 constants 383
 Common Program Data folder 22
 CommonProgramDataFolder 22
 constants
 comments 383
 improvements 383

D

dereference operator
 brackets 384
 discontinued compiler versions 215, 242, 264
 discontinued evaluation boards 215, 242, 264
 Documents folder 22
 DocumentsFolder 22

K

key features
 dSPACE Data Dictionary 232, 253, 283

L

limitations
 obsolete limitations
 general limitations 427
 Limitations
 TargetLink
 obsolete limitations 405, 407, 410, 413,
 416, 420, 423, 424
 Local Program Data folder 22
 LocalProgramDataFolder 22

M

MATLAB
 startup 321, 335

N

new API commands
 dSPACE Data Dictionary 206, 236, 259, 284
 new Code Generator options 220, 244, 272
 new evaluation boards 215, 242, 264
 new evaluation compiler versions 215, 242,
 264

O

obsolete Code Generator options 272, 380
 optimization
 improvements regarding constants 383
 overview
 TargetLink API changes 368, 376, 388

S

Simulink
 configuration parameters
 adaptation 353
 Stateflow
 loop variables 383
 supported targets 215, 242, 264

T

target support 215, 242, 264
 TargetLink
 API commands
 changes 318, 328
 API functions
 changes 337, 343, 359, 365
 AUTOSAR features, new
 compatibility mode 197
 component-based development 166, 197
 exchanging SWCs as object code 197
 frame model generation, improvements
 197
 frame models, updating 197
 hook functions, new 182, 194
 multiple instantiation of SWCs 165
 supported releases 46, 63, 78, 96, 117,
 146, 165, 197
 code changes
 migration 470, 510, 521, 535, 547, 551
 code efficiency, improved 91, 112
 code efficiency, improvement of 140, 160,
 195
 code generator options
 backward compatibility 304, 314, 319,
 333, 339, 350, 366, 412, 415, 420, 426,
 429, 430
 changed default value 304, 314, 319,
 333, 339, 350, 366, 412, 415, 420, 426,
 429, 430
 custom code, improved 95
 discontinued features 403, 406, 409, 411,
 418
 documentation changes
 migration 361
 documentation improvements 177
 enumeration, improved 95
 hook functions
 AUTOSAR 182, 194
 new 182, 194
 migration
 AUTOSAR related 360, 365
 AUTOSAR-related 338
 code changes 470, 510, 521, 535, 547,
 551
 documentation changes 361
 obsolete limitations 405, 407, 410, 413,
 416, 420, 423, 424
 Property Manager 322
 various aspects 324, 336, 341, 355, 363,
 367
 new API commands
 V-ECU generation 181, 194
 virtual ECU generation 181, 194
 new API functions 55, 69, 83, 108, 131, 155
 new code generator options 53, 68, 82,
 102, 125, 151, 172, 189
 new features 41, 57
 component-based development 183
 compute-through-overflow, optional
 deactivation 184
 custom code, improvements to 184
 enumeration support 157
 general changes 56, 70, 85, 104, 127,
 153, 179, 196
 general enhancements 56, 70, 85, 104,
 127, 153, 179, 196
 improved Stateflow support 159
 Improved System Preparation 158
 modular development 183
 virtual ECU (V-ECU) generation 185
 newly supported Simulink blocks 88, 109,
 134, 157
 target support
 discontinued compiler versions 50, 66, 81,
 99, 120, 149, 170, 186
 discontinued evaluation boards 50, 66,
 81, 99, 120, 149, 170, 186
 new compiler versions 50, 66, 81, 99,
 120, 149, 170, 186
 new evaluation boards 50, 66, 81, 99,
 120, 149, 170, 186
 supported targets 50, 66, 81, 99, 120,
 149, 170, 186
 TargetLink API
 changes 368, 376, 388
 TargetLink block options
 new 224
 TargetLink Data Dictionary
 API commands
 changes 318, 328
 new commands 124, 143, 164
 API functions
 changes 337, 343, 359, 365
 discontinued features 418, 422, 424
 migration 286
 discontinued documentation 286

manually upgrading libraries and models
290
upgrading existing data dictionaries 289
new features
 compare and merge 202
 data management 203
 DD IncludeFileGroup objects 205
 diff and merge 202
Loading one DD to different workspaces
205
Object Explorer improvements 205
points of inclusion 203
Property Value List improvements 205