TargetLink

New Features and Migration

TargetLink 2.0 – September 2004

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Technologiepark 25 33100 Paderborn Germany
Tel.: Fax:	+49 5251 1638-0 +49 5251 66529
E-mail: General Technical Support: TargetLink Support:	info@dspace.de support@dspace.de +49 5251 1638-941 support.tl@dspace.de +49 5251 1638-700
Web: Subscription to e-mail newsletter:	http://www.dspace.de http://www.dspace.de/goto?SupportNewsletter

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.de/goto?support for software updates and patches.

Important Notice

Contents

About This Document

This document provides you with a brief overview of the major new features of TargetLink 2.0 since TargetLink 1.3, and the MATLAB R14 Compatibility Update for TargetLink 2.0.

New features and enhancements

For a description of the key features, and a summary of the major enhancements made since TargetLink 1.3, refer to *Key Features of TargetLink 2.0* on page 7.

Migration

For information on the changes you may have to perform when you migrate from previous releases to TargetLink 2.0, refer to *Migrating to TargetLink 2.0* on page 41.

Last-minute information

For information on last-minute changes of TargetLink Release 2.0, refer to *Last-Minute Information* on page 71.

MATLAB R14 Compatibility Update

For a description of the key features and a summary of the major enhancements made for the MATLAB R14 Compatibility Update for TargetLink 2.0, refer to *Key Features of the MATLAB R14 Compatibility Update for TargetLink 2.0* on page 85.

For details on the limitations you have to take into account when migrating to the MATLAB R14 Compatibility Update for TargetLink 2.0, refer to *Limitations in the MATLAB R14 Compatibility Update for TargetLink 2.0* on page 89.

Legend

The following symbols are used in this document.

Warnings provide indispensable information to avoid severe damage to your system and/or your work.

Notes provide important information that should be kept in mind.

Tips show alternative and/or easier work methods.

Examples illustrate work methods and basic concepts, or provide ready-to-use templates.

Key Features of TargetLink 2.0

TargetLink 2.0 has the following key features, enhancements and changes.

New TargetLink Features

Supported MATLAB Platforms

TargetLink 2.0 offers full compatibility with the following releases of MATLAB from The MathWorks:

- MATLAB R13 SP1 with Simulink 5.1 and Stateflow 5.1.1
- MATLAB R13.0.1 with Simulink 5.0.2 and Stateflow 5.1
- MATLAB R12.1 with Simulink 4.1.1 and Stateflow 4.2.1
- MATLAB R12.1 with Simulink 4.1.2 and Stateflow 4.2.1

Stateflow 5.0 for MATLAB R13.0.1 is not supported.

For information on how to install TargetLink and a detailed description on its requirements, refer to *Installation Steps* in the *TargetLink Installation and Configuration Guide*.

If you update an existing MATLAB Release 13 installation with the Simulink 5.0.2 and Stateflow 5.1.x updates (Web download), take care to also update the MATLAB product itself. Otherwise your installation will run MATLAB, Simulink and Real-Time Workshop as a mix of Release 13 and Release 13.0.1 products, which is known to lead to various problems. It is important that MATLAB, Simulink and Stateflow all come from the same release. You can verify this by inspecting the (Rxx) version numbers output by the MATLAB `ver` command.

To install a dSPACE Release together with CalDesk and/or TargetLink in the **same folder**, you have to install the products in a certain order.

Combinations that do not comply with these installation orders will not work properly.

The table below shows you which installation orders will work:

dSPACE Product ...	Will Work When Installed After dSPACE Product ...							
	RLS 3.4.x	RLS 3.5	RLS 4.0.x	RLS 4.1	TL 1.3	TL 2.0	CAL 1.0	CAL 1.1
RLS[1)] 3.4.x	–	No	No	No	Yes	No	No	No

dSPACE Product ...	Will Work When Installed After dSPACE Product ...							
RLS 3.5	No	–	No	No	No	No	No	No
RLS 4.0.x	No	No	–	No	No	No	Yes	No
RLS 4.1	No	No	No	–	No	No	Yes	No
TL[2] 1.3	No	No	No	No	–	No	No	No
TL 2.0	Yes	Yes	Yes	Yes	No	–	Yes[4]	No
CAL[3] 1.0	Yes	Yes	Yes	No	Yes	No	–	No
CAL 1.1	Yes	Yes	Yes	Yes	Yes	Yes	No	–

1) dSPACE Solutions for Control Release
2) TargetLink
3) CalDesk
4) Only if you have installed CAL 1.0.2

dSPACE Data Dictionary

The dSPACE Data Dictionary is a data container that holds all relevant information for code generation. It helps you to keep ECU variables consistent, independently of any model partitioning and across model boundaries.

Structure of the dSPACE Data Dictionary

Basically, the dSPACE Data Dictionary (DD) contains a tree of objects which have child objects and properties. To represent ECU data, certain object kinds, describing variables, data types, scaling formulas, functions, C modules, RTOS tasks, messages, etc., are defined.

The DD objects are arranged in four main areas:

- Config area
- Pool area
- Subsystems area
- Application area

The Config and Pool area contain pre-code generation data, for example, variables and data types which are referenced from TargetLink blocks in a Simulink model. The Code Generator reads this information from the DD and creates the corresponding production code. It also writes detailed information about the generated code to the Subsystems and Application areas of the DD. This information is available for postprocessing tools, for example, the Document Generator and the ASAP2 File Generator.

Files replaced by the dSPACE Data Dictionary

The dSPACE Data Dictionary replaces two files that were available in former TargetLink versions. These are

- Project configuration file (`default_cfg.m`)
- Info file (`*_info.m`)

Even if TargetLink 2.0 can import TargetLink 1.x project configuration files (`*_cfg.m`), do not set up new projects in these file formats. Use the dSPACE Data Dictionary instead.

Both the project configuration file and the Info file were autonomous data structures in M script format. The dSPACE Data Dictionary makes this information convenient to access and also allows modification. For details on migrating these files, refer to:

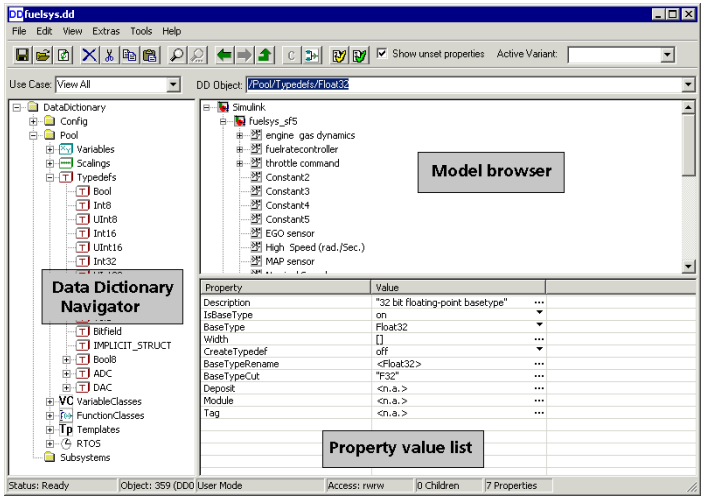
- *Upgrading the Project Configuration File* on page 43
- *Converting Data from the DD Project File to the Info File* on page 52.

Highlights of the dSPACE Data Dictionary

Default data The dSPACE Data Dictionary is filled with default data the first time a project is opened. It holds a great wealth of additional information, which is not available in a Simulink model but required for production code, for example, specifics about function calls, RTOS tasks and messages, variable classes and data variants. In addition, it provides templates for auxiliary variables, which are not defined in the block diagram. For details, refer to *Working with the Data Dictionary Manager* in the *dSPACE Data Dictionary Basic Concepts Guide*.

Defining variables and corresponding properties You can (but do not have to) define variables and corresponding properties in the dSPACE Data Dictionary. The variables can be referenced in TargetLink block dialogs via their paths in the Data Dictionary object tree. You can assign variables to block outputs, states and parameters. For details, refer to *How to Create a Variable Object in the dSPACE Data Dictionary* in the *TargetLink Production Code Generation Guide*.

Data Dictionary Manager The Data Dictionary Manager is the graphical user interface of the dSPACE Data Dictionary. It allows you to browse the DD object tree, inspect and modify objects, and invoke import/export functions.



API functions Open API interfaces offer full M script access to the dSPACE Data Dictionary and make it easy to integrate the dSPACE Data Dictionary into company-specific environments. They allow further import and export of external data formats (for example, via XML/ XSL style sheets). For details, refer to:

- *Working with the MATLAB API* in the *dSPACE Data Dictionary Basic Concepts Guide*.
- *Using TargetLink via Command Line* in the *TargetLink Advanced Practices Guide*
- *Overview of the TargetLink API Reference* in the *TargetLink API Reference*.

Import and export interfaces The dSPACE Data Dictionary supports various import/export formats, including XML, OIL and ASAM-MCD 2MC (ASAP2). For details, refer to:

- *Introduction to XML File Import and Export* in the *dSPACE Data Dictionary XML Import and Export*.

- *Introduction to ASAM-MCD 2MC File Import and Export* in the *dSPACE Data Dictionary ASAP2 Import and Export*.
- *Introduction to OIL File Import and Export* in the dSPACE Data Dictionary OIL Import and Export.

Structured Data Types

Structured data types are a common means to collect data of different data types. TargetLink supports structured data types, called structs below, using the typedef mechanism provided by the dSPACE Data Dictionary. Each struct is identified by a unique name and consists of elements called components that hold the different types of data. The allocation of the struct variable depends on the variable class of the variable representing the struct.

Define structured data types

You can define structured data types and declare structured variables according to C notation, including bitfields (":" notation in C) and nested structure hierarchies ("." notation in C).

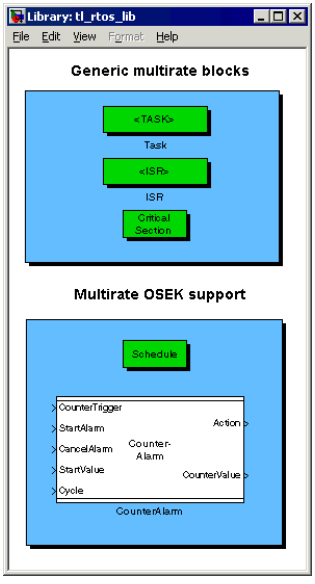
Bitfields in particular are a very RAM-efficient way to store Boolean data. Further advantages of structs are efficient passing of variables to functions, and enhanced readability and structure of the generated code. For details, refer to *Arranging Data in Structured Variables* in the *TargetLink Advanced Practices Guide*.

Multirate Models and OSEK Support

Usually, the code running on an ECU is divided into different functional units that are controlled by a multitasking real-time operating system (RTOS). TargetLink now supports multirate code generation, which allows you to create multirate models containing blocks with different sample times. When generating production code, TargetLink maps the algorithms of your multirate model to RTOS objects and services.

Features of multirate modeling

RTOS blocks TargetLink contains the RTOS Blocks library containing blocks for multirate modeling.



Tasks and interrupt service routines TargetLink provides different ways to partition your model into tasks. You can use the Task block residing in the RTOS blocks library, or let TargetLink partition the model.

TargetLink provides an ISR block to assign a function-call trigger subsystem to an interrupt service routine (ISR). For details, refer to *Task Block* in the *TargetLink Block and Object Reference*.

Specifying intertask communication Since the different tasks in your model communicate with each other by exchanging signals, the signals' data integrity must be ensured. TargetLink provides the message mechanism for this. A message is represented by one or more plain variables or by a data structure with several components. During data exchange, TargetLink protects the data integrity of a message by preventing the sender and/or receiver from being interrupted. For details, refer to *InPort Block* and the *OutPort Block* in the *TargetLink Block and Object Reference*.

Protecting critical sections It may be important not to let other tasks interrupt some parts of your code, for example, certain I/O operations. You can use the CriticalSection block to protect such functions from being interrupted during execution. For details, refer to *Critical Section Block* in the *TargetLink Block and Object Reference*.

Support of OSEK-compliant RTOSs

Using OSEK-compliant RTOSs TargetLink supports OSEK, a leading standard for real-time operating systems. In general TargetLink supports all OSEK-compliant RTOSs. It gives very convenient support to RTOSs from different vendors. Before production code generation, you can select the RTOS you want to generate the code for. For details, refer to *RTOS Page (Main Dialog Block)* in the *TargetLink Block and Object Reference*.

Counters and alarms For OSEK-compliant applications, the RTOS Blocks library provides the CounterAlarm block, which lets you set up counters and alarms. For example, you can use alarms to activate tasks or set events. For details, refer to *CounterAlarm Block* in the *TargetLink Block and Object Reference*.

OIL file import and export TargetLink lets you import OIL files and use the imported OSEK objects in your application. You can also create OSEK objects in TargetLink and export them to an OIL file. For details, refer to *Import OIL File Dialog* and the *Export OIL File Dialog* in the *TargetLink Block and Object Reference*.

There are points you have to note when you generate production code for a specific OSEK RTOS. For details, refer to the following topics:

- *Points to Note When Generating Production Code for ProOSEK*
- *Points to Note When Generating Production Code for OSEKturbo*
- *Points to Note When Generating Production Code for osCAN*
- *Points to Note When Generating Production Code for OSEKWorks*
- *Points to Note When Generating Production Code for RTA*

The above documents are available only in the online version of this documentation (dSPACE HelpDesk for TargetLink 2.0).

Worst-Case Autoscaling

Worst-case autoscaling is a tool-based method to determine the scaling of variables according to the worst-case range of the signals they represent. A worst-case range is specified by the minimum and maximum values which a block's output or state variable can take on with respect to the range of its input(s).

Use case

The worst-case autoscaling method helps you to determine the range of a variable, for example, if you want to find appropriate scaling parameters for it (data type, LSB, offset). For details, refer to *Example: Worst-Case Autoscaling* in the *TargetLink Production Code Generation Guide*.

Preconditions

The worst-case range determination method requires a sufficient number of blocks whose output range is known in advance, that is, user-defined range limits are specified.

Usually, the ranges of signals that go into or leave a TargetLink subsystem are known. In addition, the ranges of look-up table entries are usually known. Often the ranges of displayable variables or variables that represent certain physical quantities are known as well.

Worst-case range calculation

Ranges known in advance are propagated to adjacent blocks following the signal lines forward and backward. User-defined ranges of TargetLink InPort and OutPort blocks, for example, are propagated through the related TargetLink subsystem as far as possible – at optimum into the deepest subsystem.

Finally, a worst-case output range is derived for each block in the model from the ranges of its input(s) or (with backward calculation) the worst-case input range of each block is derived from the output ranges. Block-specific calculation rules are applied. For details, refer to *Range Determination Basics* in the *TargetLink Production Code Generation Guide*.

No controlled system model and stimulus signals

Calculation of worst-case ranges is possible without a model of the controlled system and stimulus signals – unlike range determination based on simulation.

No overflow risk Worst-case ranges are never exceeded and are therefore immune to overflows if the assumptions on the range of the input signals are correct.

Autoscaling When worst-case ranges have been determined, the Autoscaling tool can calculate appropriate scaling parameters. For details, refer to *TargetLink Autoscaling tool* on page 61 and to *Autoscaling Basics* in the *TargetLink Production Code Generation Guide*.

Stand-Alone Mode of the TargetLink Blockset

The blockset that is used with the TargetLink Base Suite can also be used in stand-alone mode, that is, detached from the TargetLink Base Suite. This TargetLink Blockset is supplied free of charge.

Use cases

You can use the TargetLink Blockset in MATLAB® to design, simulate, and prototype controller models. Production code generation, however, is not possible. The models are compatible with:

- The MathWorks' Real-Time Workshop®
- dSPACE's Real-Time Interface

The simulation behavior of the TargetLink Blockset corresponds to the simulation behavior of the Simulink® blockset.

Production code generation

Models that have been designed with the TargetLink Blockset are compatible with the TargetLink Base Suite. This comprehensive compatibility allows the linkage of two crucial development phases:

1. Prototyping using Real-Time Workshop
2. Production code generation using TargetLink

Model conversion not necessary

Prototyping teams, using the TargetLink Blockset and dSPACE's Real Time Interface, and production code teams, using the TargetLink Base Suite, can exchange models without model conversion. This is useful as these teams frequently exchange models in the course of development.

TargetLink Base Suite

If you work with the TargetLink Base Suite, you can switch between the full-featured and stand-alone mode within the same TargetLink session, and your models remain unchanged.

For details, refer to *Introduction to the TargetLink Blockset* in the *TargetLink Blockset Guide*.

The TargetLink CD contains a separate setup for the TargetLink Blockset. The TargetLink Blockset can be used on its own (stand-alone mode), that is, detached from the TargetLink Base Suite. You can distribute this setup to your team members.

Block Library and Block Properties

TargetLink 1.x models must be upgraded to TargetLink 2.0 For details, refer to *Upgrading TargetLink Models* on page 48.

dSPACE Data Dictionary

All TargetLink blocks can contain references to the dSPACE Data Dictionary, for example, a complete variable specification, a user-defined data type or a fixed-point scaling formula.

New simulation blocks

Preprocessor IF block Lets you control the compilation of conditionally executed subsystems. It enables TargetLink to generate code sections enclosed in preprocessor directives for conditional compilation. Along with *Action* subsystems, it lets you implement standard C-like #if - #else logic. For details, refer to *Preprocessor IF Block* in the *TargetLink Block and Object Reference*.

Unit Delay Reset Enabled This block is an extension of the standard Unit Delay block. It has additional external Reset, Enable, and InitialValue inputs. Thus you can implement, for example, filters with resettable states, counters, or watchdog algorithms. For details, refer to *Unit Delay Reset Enabled Block* in the *TargetLink Block and Object Reference*.

Direct Look-Up (n-D) Lets you select single elements or columns from a matrix. Typically, this block is used for adaptive control algorithms. For details, refer to *Direct Look-Up Table Block* in the *TargetLink Block and Object Reference*.

FCN block Allows you to apply a specified C language style expression to its input. For details, refer to *Fcn Block* in the *TargetLink Block and Object Reference*.

RTOS Blocks TargetLink provides the RTOS Blocks library containing blocks for multirate modeling. For details, refer to *Multirate Models and OSEK Support* on page 16.

Enhanced block properties

Function block Provides various new features:

- Scaling invariance

  Takes scaled integer input values and produces scaled output values as its result can sometimes be implemented so that the code is invariant with respect to the scaling. This means the code can be reused for inputs of different scalings without explicit scaling adjustment code. For details, refer to *Scaling-Invariant Functions* in the *TargetLink Advanced Practices Guide*.

- Order of function arguments

  Lets you specify the order of arguments in a function call. For details, refer to *Arguments Page (Function Block)* in the *TargetLink Block and Object Reference*.

- Incremental code generation

  For details, refer to *Incremental Code Generation* in the *TargetLink Production Code Generation Guide*.

Trigonometric Function block Lets you use fixed-point data types for trigonometric functions. This implementation is identical to the behavior in Simulink. For details, refer to *Trigonometric Function Block* in the *TargetLink Block and Object Reference*.

Fixed-point data types for hyperbolic functions are not supported.

Unit Delay Lets you specify the name and class of the Unit Delay state variable. This was not possible in TargetLink 1.3.

Data Store Memory Lets you specify the name of the C code variable independently of the data store tag. In TargetLink 1.3, the variable name was identical to the data store tag.

Rate Limiter block The Rate Limiter block can be used in enabled subsystems. Within enabled subsystems with the parameter *states when enabling* set to "held", this block requires the elapsed time between the disable/enable event. Consequently, the block uses the predefined variable *SystemTime* to access a related counter or timer to get the current sample time value. This was not implemented in former versions of TargetLink. For details, refer to *SystemTime Variable* in the *TargetLink Advanced Practices Guide*.

Discrete-Time Integrator block The Discrete-Time Integrator block can be used in triggered subsystems. Because the elapsed time (delta T = T[k]-T[k-1]) needed to calculate the output signal for this block is not necessarily constant in triggered subsystems, the current sample time value is essential. Access to the current sample time was not implemented in former versions of TargetLink. For details, refer to *Time Evaluation in Triggered Subsystems* in the *TargetLink Advanced Practices Guide*.

Code Generator Extensions

This section shows the new features and enhancements in TargetLink's code generation.

Function interface with busses

TargetLink 2.0 supports internal bussed signals. They can be used as inputs and outputs of function subsystems.

This TargetLink version does not support Simulink busses that cross the TargetLink root system boundaries. Note that a subsystem that is configured for incremental code generation is regarded as such a system.

Boolean base data type

Boolean data is common in ECU development. It typically codes control flow information and stores state information. Because of this, Bool is introduced as a new base data type. It can be selected for block outputs and internal state variables where {0,1} signals are generated. For details, refer to the BaseType description in the Data Model API of the *Data Dictionary MATLAB API Reference*.

Unscaled integer data interface

A new concept has been implemented, allowing unscaled integer data to be passed to function subsystems. This includes signed and unsigned integers of bit width 8, 16 or 32. This feature facilitates function reuse for functions that are called with different scaling parameters but with the same scaling ratio.

Access functions

New variable class properties have been introduced that allow the implementation of interfaces by access functions for variables. To ensure the maximum usability and configurability of this feature, you have a choice of different implementations (macro or C function), access variants (direct access or access through local variables) and addressing (one function for each variable or generic function with the variable name passed as a function argument). For details, refer to *Access Functions* in the *TargetLink Advanced Practices Guide*.

Template mechanism

You can use templates to influence implicit objects during code generation, for example, to adapt their naming to a company coding style. Several different kinds of templates are available in the dSPACE Data Dictionary, for example, for variables, types, functions and files. For details, refer to *Changing Implicit Objects of the Code Generator* in the *TargetLink Advanced Practices Guide*.

Incremental Code Generation

Incremental code generation is available on the level of Simulink subsystems which contain a TargetLink Function block. It lets you generate production code for modified parts of your model.

Tailored code generation

This allows more tailored code generation, that is, you can create smaller code fragments that can be tested separately and thus may reduce development time. After a modification, only the modified model part must be regenerated, which also facilitates the review and testing processes. TargetLink performs the necessary consistency checks when building the overall application from all the parts. For details, refer to *Incremental Code Generation* in the *TargetLink Production Code Generation Guide*.

In TargetLink 2.0, incremental code generation cannot be used for the following subsystems:

- Subsystems containing external function-call connections
- Subsystems for which you enabled the **Make function reusable** checkbox
- Subsystems containing RTOS blocks or intertask communication

Customer-Specific Code Formatting

With TargetLink 2.0, you are able to customize the style and layout of your generated C code, for example, according to your company-specific standards. This includes comment headers, comment lines of variable declarations and other options changing the appearance of TargetLink's code output. However, this is an optional feature, you can also let TargetLink use it's predefined standards.

For details, refer to *Customizing TargetLink's Code Formatting* in the *TargetLink Advanced Practices Guide*.

Implemented XML sheets

During each code generation, TargetLink reads predefined or user-written XML (eXtensible Markup Language) sheets defining the code formatting. If you edit them, your code output will be formatted the way you defined it.

Style Definition File

As a quick and easy way to customize general code formatting, you can edit a predefined Style Definition File (`cconfig.xml`). In this file, you can simply set predefined attributes according to your needs, which change some general code formatting aspects, for example, the way statement comments appear in your C code.

Root style sheet and detailed style sheets

A more advanced way, which is useful for experienced XML users, is to use and organize custom-made (user-written) XSL style sheets. A root style sheet lists multiple style sheets for specific projects you carry out. You can edit the sheets for detailed access to TargetLink's code formatting mechanics.

Stateflow Optimizations and Extensions

This section shows the new features, enhancements and changes that were made in TargetLink's Stateflow code generation.

Stateflow optimization

Stateflow code in TargetLink has been further optimized, for example, by improved decisions between If-Else and Switch-Case for exclusive states and scope reduction of Stateflow variables.

Stateflow reuse

Stateflow charts and graphical functions within a chart can be instance-specific. This allows the reuse of Stateflow systems or separate functions with Stateflow blocks. For details, refer to *Function Reuse* in the *Advanced Practices Guide*.

Stateflow data logging

Variables in Stateflow are available for TargetLink data logging. One sample is recorded at the end of each simulation step. The same options for data logging are available as for Simulink blocks.

Stateflow Logger The Stateflow Logger block optimizes the logging of all Stateflow charts that reside in the related subsystem and underlying subsystems. The Stateflow Logger block ensures that an internal Stateflow variable is only logged when its value changes. This behavior prevents deviation between MIL and SIL (or PIL) simulation results. For details, refer to *Stateflow Logger Block* in the *TargetLink Block and Object Reference*.

Stateflow FcnCall Logger The Stateflow Function Call Logger block optimizes the logging of the related Stateflow chart. It ensures that all internal chart variables are only logged when their value changes. This behavior prevents deviation between MIL and SIL (or PIL) simulation results.

For details, refer to *Working with Stateflow* in the *Advanced Practices Guide*.

Variant Coding

With variant coding, TargetLink provides a method of specifying different sets of properties of one signal or parameter.

Variant coding types

Data variants The value property of a variable can have data variants which you can specify in the Data Dictionary Manager. All data variants are generated into the same code and can be switched in the initialization phase of the ECU. There are different options to represent data variants in the C code, for example, as separate structs for each variant, or as an array of structs.

Code variants at code generation time You can specify different code variants in the Data Dictionary Manager, for example, two data types for one variable. Code variants lead to source code that is generated differently depending on the active variant.

For details, refer to *Variant Coding* in the *TargetLink Advanced Practices Guide*.

Code Coverage Testing

TargetLink supports a method of analyzing the test coverage of possible program execution paths of your production code. The tests allow you to find parts of your production code which are not executed and thus are not tested adequately. TargetLink provides different levels of code coverage tests. For details, refer to *Basics of Code Coverage Tests* in the *TargetLink Production Code Generation Guide*.

Levels of code coverage tests

Statement coverage Statement coverage tests are also referred to as C1 tests. They measure whether each block of non-branching statements of your production code is executed.

Decision coverage Decision coverage tests are also referred to as C2 tests. In addition to statement coverage, they measure if Boolean expressions in your production code evaluate to both true and false. In addition, all branches of Switch statements, including the "default" branch, must be executed at least once.

Analyzing test results

Progress bar During the code coverage tests, a progress bar shows the percentage of code coverage. The results can be accumulated over multiple simulation runs.

Test report After you perform code coverage tests, TargetLink generates a test report indicating the parts of your production code which are not executed and thus are not tested. For details, refer to *How to View the Code Coverage Test Report* in the *TargetLink Production Code Generation Guide*.

Model Conversion

There are several model conversion enhancements in TargetLink. Basically, model conversion in TargetLink means that a Simulink model is prepared for production code development with TargetLink.

Mapping properties

Fixed-point properties mapping of MATLAB R13 blocks The Simulink-to-TargetLink Blockset converters map fixed-point specific properties from the Simulink blockset to the TargetLink Blockset. Other block-specific properties are mapped to corresponding TargetLink properties, whenever feasible. This conversion is bidirectional.

Graphical user interface

A new GUI helps you to configure options and invoke the model conversion routine.

Conversion of libraries

A new GUI guides you through the workflow of library conversion. There is an option to automatically generate libmap entries for a specific library with masked Simulink blocks. Additionally, there is an option to automatically convert referenced libraries, even nested libraries.

Specifying the model name

A new option lets you specify the model name upon conversion.

Restoring TargetLink data

The current version of TargetLink lets you restore TargetLink-specific block data that was saved during TargetLink to Simulink conversion, beforehand.

Document Generation

With TargetLink 2.0, the automatic document generation feature has been considerably enhanced. This includes:

- Documentation of Statecharts
- Screen copies in EPS format
- Customization of tables in the document
- Customization of the language
- File format conversion utility

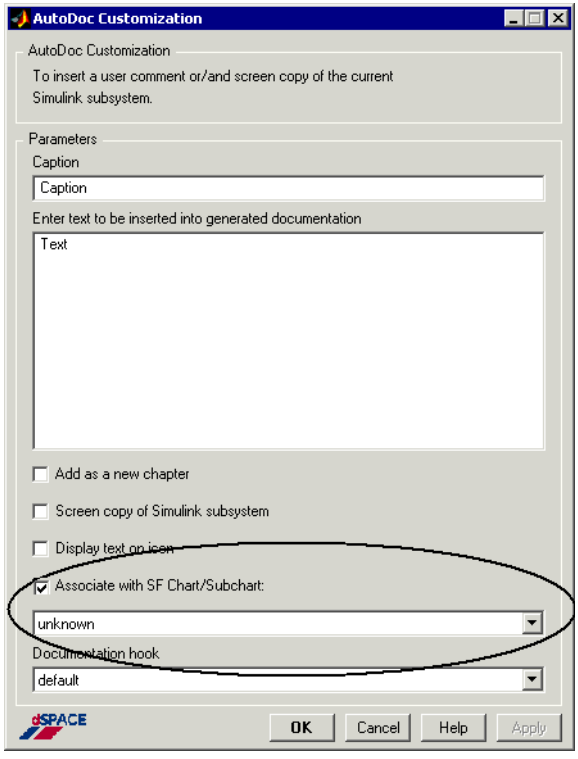
The documentation process is started via the `tldoc_default` M file. You can influence the layout of the generated documentation by means of the `tldoc_layout.m` configuration file, which is located in `%DSPACE_ROOT%\Matlab\TL\config\`. It provides a set of properties that allow you to define the language and table layout set to use, for example. For details, refer to *Structure of the Generated Documentation* in the *TargetLink Production Code Generation Guide*.

Including user-defined information

The new AutoDoc Customization block allows you to customize the generated documentation. For example, you can insert the following type of information into the generated documentation:

- A user comment
- A screen copy of the Simulink® subsystem where the block resides
- New user-defined chapters

The `Associate with SF Chart/Subchart` option and the chart selection list let you specify that Stateflow charts and subcharts (if any) are treated in the same manner as Simulink subsystems. That means additional screen shots of statecharts can be added to the generated documentation.



For details, refer to *How to Include User-Defined Information in the Documentation* in the *TargetLink Production Code Generation Guide*.

If you set the `AutodocFunctionSelection` property in the TargetLink subsystems command to `'on'`, only subsystems or statecharts associated with an AutoDoc Customization block are documented.

File format conversion utility

The file format conversion utility allows conversion between HTML format and Portable Document Format (PDF), or Rich Text Format (RTF). In addition, you can specify the image format (PNG or EPS) for the converted documentation. The file format is selected with the new command `tldoc('Convert',...)` which is called at the end of a documentation configuration file. Select one of the predefined files `tldoc_pdf.m` or `tldoc_rtf`.m to get proper printable documents.

For details, refer to *Customizing the Generated Documentation* in the *TargetLink Production Code Generation Guide*.

Target Optimization Modules for MPC5xx

In order to maximize efficiency, TargetLink provides target optimization modules for several microcontroller/compiler combinations. TargetLink 2.0 provides new target optimization modules for Motorola MPC5xx in combination with Diab compilers and Green Hills compilers.

For details, refer to *Motorola MPC5xx* in the *TargetLink Target Reference*.

Data Logging

The simulation concept of TargetLink 2.0 has been enhanced by the ability to log signals via direct access to their memory addresses. In previous TargetLink versions, signal logging was possible only via log macros (selection in the block dialog). With the new simulation concept, you can choose any global variable in the generated production code. No logging macro and recompile of the production code are required.

Specifying data logging in the dSPACE Data Dictionary is possible only when production code has been generated. This is because the list of variables which can potentially be logged is not available until code generation.

Testing production code without rebuild

Selecting a variable for logging via the dSPACE Data Dictionary is useful if you want to test the production code without the impact of log macros or to log additional variables without rebuilding the simulation application. The latter aspect saves time and allows you to quickly acquire additional data to investigate it. Possible optimizations can be executed, since no variables are created due to logging. As a consequence, the Code Generator is able to eliminate superfluous block outputs to obtain leaner and faster code.

For details, refer to *Logging Basics* in the *Production Code Generation Code Guide*.

Miscellaneous Features

The following new features facilitate your work with TargetLink.

Project data type preselection

The project configuration includes a property that defines the default data type for each TargetLink block. New blocks taken from the block library are preset to this data type, for example, Int16 or Float32. This applies mainly to block output variables. Parameters and internal state variables can be set to different data types, if this seems advisable. The default data type to be used for floating-point calculations and integer calculations can be set separately. For details, refer to *How to Preselect the Default Basic Data Type* in the *TargetLink Installation and Configuration Guide*.

Unicode support

While English is commonly used in the specification of models and C source code, it is often desirable to enter comments or descriptions in the language best known to the user. To be able to display names or descriptions in different languages, TargetLink supports different character sets, for example, Japanese and Cyrillic. You can specify the character set used on your computer. For details, refer to *Using Different Character Sets* in the *TargetLink Advanced Practices Guide*.

Floating-point exception handling

Floating-point exception handling, for example, overflow and underflow, is implemented for SIL simulation mode and for certain floating-point processors in PIL simulation mode. The events are reported in the Message Browser.

Long TargetLink subsystem ID's

The TargetLink subsystem ID can be a string of up to 10 characters. This allows you to specify longer strings, for example, ID = "abc" for the ECU feature *Active Body Control*.

The subsystem ID is used in the generated production code for variable names, type names and so on. The name macro $S_$B, for example, expands to `Sabc1_Gain`.

Strong data typing in Simulink models

Strong data typing of Simulink signals and parameters makes models easier to construct and read. Strong data typing means setting and displaying the data type really needed so that use of the double data type (during modelling) can be avoided if possible. This makes models easier to understand and read, as the actual data types can be seen at the outputs of Simulink blocks. For details, refer to *Using Strong Data Typing in Simulink Models* in the *TargetLink Production Code Generation Guide*.

Cast output signal to TargetLink type

You can force TargetLink to use its current (integer) production code data type for block outputs instead of the normal double data type in MIL simulation mode. This is useful for blocks that use pure integer signals, for example, the input of a Multiport Switch block, or the index value of an Assignment block. If you keep to the rule that integer types should be used only for blocks whose signals are integers (not fixed-point) by nature, there are no quantization errors. For details, refer to *Data Type Processing Basics* in the *TargetLink Production Code Generation Guide*.

Exporting a TargetLink subsystem into a test frame

Usually, during SIL/PIL simulation you test the complete production code generated for one TargetLink subsystem. Now you can simulate and test the behavior of a single function that is part of the code. For details, refer to *How to Simulate a Separate Function in Test Mode* in the *TargetLink Production Code Generation Guide*.

Migrating to TargetLink 2.0

When migrating from TargetLink Release 1.3 to 2.0, you should be aware of the following aspects:

Migrating from TargetLink 1.3 or Earlier

When migrating from former TargetLink releases, for example, from TargetLink Release 1.2 to 2.0, you have to migrate step by step via the intervening TargetLink versions. Consequently, it is recommended to read the corresponding New Features And Migration Guide(s) beforehand. These guides contain information that is essential for migration.

If you want to migrate from TargetLink version 1.1 to TargetLink version 2.0, you have to follow the migration steps given in:

1. New Features and Migration of TargetLink version 1.2
2. New Features and Migration of TargetLink version 1.3
3. Finally, the migration steps described in this document.

Accessing new features and migration documents

You can find the PDF files of the New Features and Migration documents for previous releases in the \Doc\Print folder on the dSPACE CD or download them from http://www.dspace.de/goto?migration_tl.

The PDF files are named NewFeaturesAndMigrationxx.pdf, where xx stands for the version or release number.

Upgrading the Project Configuration File

The dSPACE Data Dictionary replaces the project configuration file (`default_cfg.m`) that was available in former TargetLink versions. The configuration data, for example, user-defined data types or variable classes, which was available in the project configuration file, is now accessible via the dSPACE Data Dictionary. The Data Dictionary Manager makes the information convenient to access and also allows modification.

Even if TargetLink 2.0 is able to import TL 1.x project configuration files, do not set up new projects in this file format. Use the dSPACE Data Dictionary instead. In TargetLink 2.0, a lot of configuration options have been added, which were not available in the `*_cfg.m` file, for example, templates for variables, access functions, look-up functions, to name just a few.

When you set up a new project, TargetLink creates a DD project file automatically. This file is based on a template that comes with the dSPACE Data Dictionary and TargetLink. It contains all the information necessary for modelling and generating production code.

Importing an old project configuration file

However, when you migrate from an old project, you usually need to import your existing TargetLink 1.x project configuration file. To do so, TargetLink provides an API command that converts all configuration data in a `*_cfg.m` file.

Other information such as the target simulation configurations or the code editor configuration can be accessed in the `tl_global_options.m` file. For details, refer to *Configuring Global Options* in the *TargetLink Advanced Practices Guide*.

Default variable classes and function classes

In TargetLink 2.0, the default variable classes and default function classes from TargetLink 1.3 project configuration files are represented by *VariableClassTemplates* and *FunctionClassTemplates* in the Data Dictionary. The templates give you much better control over defaults for generated variables and for functions.

The import of user-defined data types, variable classes, function classes, and default classes into the dSPACE Data Dictionary is fully automatic and does not require further work.

However, it is recommended to review the imported objects to take full advantage of the new TargetLink 2.0 features. For example, you can:

- Associate a variable class with an AccessFunctionTemplate to generate function calls whenever a variable is read/written
- Specify the module name where a data type is defined
- Specify constraints for a Typedef object
- Select additional optimization options for variable classes and function classes
- Select a code output template for a function class

How to Import a Project Configuration File into the dSPACE Data Dictionary

When you migrate a model created with TargetLink 1.3 to TargetLink 2.0, you need to import the project configuration file if it holds customized data.

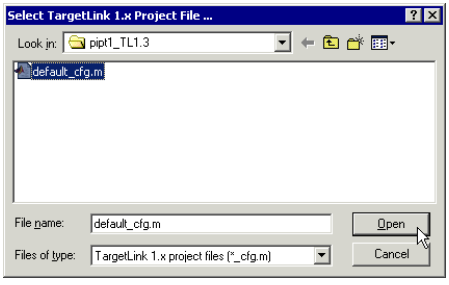
Precondition You must provide a project configuration file (*_cfg.m).

Method

To import a project configuration file in the Data Dictionary Manager

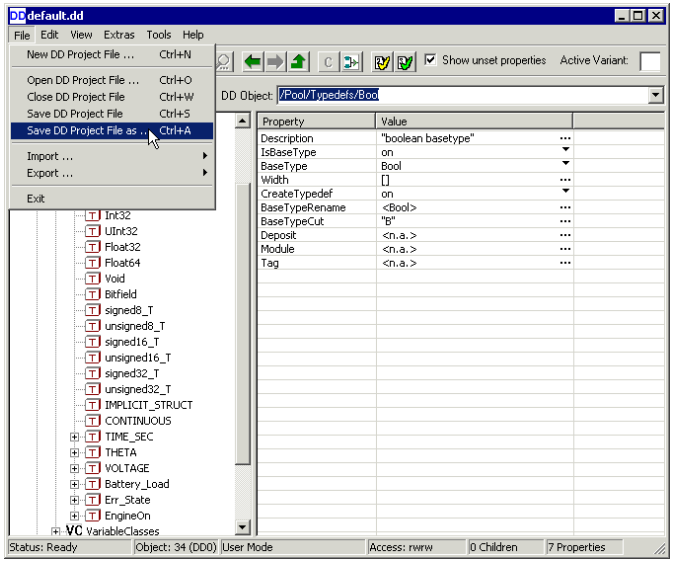
1. In MATLAB, change to the folder for your new TargetLink 2.0 project.
2. In the MATLAB Command Window, type **`dsddman`** to open the Data Dictionary Manager.

   The Data Dictionary Manager opens.
3. In the Tools menu, click **Import TL 1.3 Project File (`*_cfg.m`)**.
4. In the file selection dialog, select the desired TargetLink 1.3 project configuration file.



5. Click **Open** to convert the project configuration file to a new DD project file, which is stored in the working folder.

6 In the File menu, click **Save DD project file as** if you want to store the DD project file under another name.



Result You have now imported all the data from the project configuration file into the Data Dictionary Manager.

You can also use the `tl_upgrade_projectfile` API command to upgrade the project configuration file from TargetLink 1.x (`*_cfg.m`) to TargetLink 2.0 (`*.dd`).

How to Select a DD Project File Name

In TargetLink 1.x, the project name was configured with the API function `tl_set_project` and could be queried with `tl_get_project`. These functions are still supported but internally mapped to the new functions `dsdd_manage_project` and `dsdd_pref`.

To select a DD project file name

➤ In the MATLAB Command Window, type **`dsdd_pref`** to open a GUI where you can define the default DD project file name.

There is also an option that allows you to associate a model with a fixed DD project file name, regardless of the global default. To associate a model with a fixed DD project file name, open the TargetLink Main Dialog and go to the Options page. For detailed information refer to *Options Page (Main Dialog Block)* in the *TargetLink Block and Object Reference*.

Result This DD project file is automatically loaded when you open a Simulink model.

Upgrading TargetLink Models

Due to TargetLink's new features, such as the dSPACE Data Dictionary, support of OSEK-compliant operating systems, and new block properties, you need to upgrade your models that were created with TargetLink 1.3.

It is strongly recommended that your administrator upgrades all block libraries manually before you open any model with TargetLink 2.0 for the first time. If this is not done, problems can occur with regard to read/write rights as the libraries are usually maintained by an administrator and kept under version control. For details, refer to *How to Upgrade Libraries Manually* on page 51.

TargetLink model upgrade

The TargetLink model upgrade utility (`tl_upgrade`) is started automatically if you open a model whose model revision number does not match the current revision number. All TargetLink blocks and Stateflow objects are checked and updated accordingly if their data does not comply with the current TargetLink version. Custom Code S-functions and libraries are also rebuilt if you select the corresponding options in the Upgrade Options frame. For details, refer to *How to Upgrade Models from Previous TargetLink Versions Automatically* on page 49.

- If you do not upgrade, missing and incompliant data will cause numerous errors when you work with the non-upgraded model. This applies, for example, if you try to open a TargetLink block dialog, start code generation, or invoke any of the TargetLink tools.
- For user-defined libraries containing TargetLink blocks, the model upgrade is not invoked automatically, but must be started in the MATLAB Command Window manually. For details, refer to *How to Upgrade Libraries Manually* on page 51.

How to Upgrade Models from Previous TargetLink Versions Automatically

To upgrade models created with previous versions of TargetLink and to avoid missing or incompliant data, your TargetLink models are updated automatically when you open them.

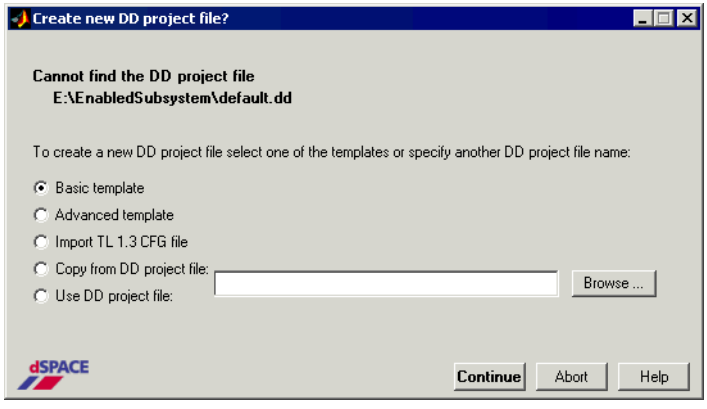
Precondition

You need a valid DD project file before model upgrading is performed. To facilitate the entire upgrade process, this file is created automatically if it does not exist. If a TargetLink 1.3 project configuration file exists, it is recommended to upgrade it beforehand.

To upgrade models from previous TargetLink versions automatically

1. Start MATLAB.
2. Open a model created with a previous TargetLink version.

   If no DD project file exists yet, the **Create new DD project file** dialog opens and you are prompted to create one.

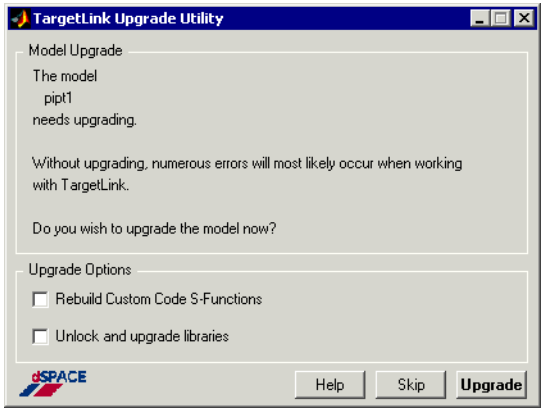


For small projects, the basic DD project file is sufficient. The advanced DD project file contains additional variable classes and templates which are usually needed for detailed TargetLink configurations in advanced projects.

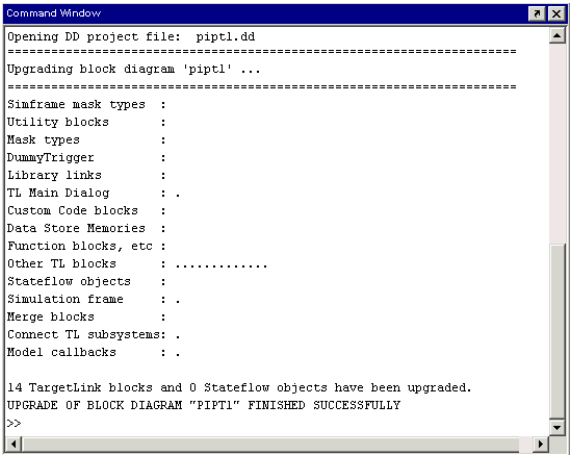
3 Select a template and click **Create.**

After the DD project file has been loaded, the Model Upgrade Utility Dialog opens.

4 In the **Upgrade Options** frame, select the desired options and click **Upgrade**.



Result Once the upgrade process is finished, the model opens automatically. A list of the TargetLink blocks and Stateflow objects that have been upgraded is displayed in the MATLAB Command Window.



You can invoke the model upgrade utility by entering **`tl_upgrade('<modelName>')`** in the MATLAB Command Window.

How to Upgrade Libraries Manually

You can upgrade libraries manually, for example, if you have not selected the **Unlock and upgrade libraries** option in the TargetLink Upgrade Utility dialog during the automatic TargetLink model upgrade.

Restrictions

If a library link points to another block in the same library, the conversion utility does not upgrade this instance of the block. For details, refer to *General Limitations* in the *TargetLink Production Code Generation Guide*.

Precondition

If the library to be upgraded is already open, it must be unlocked before it can be upgraded. To do so, click Edit - Unlock library in the Simulink File menu.

To upgrade a library manually

➤ In the MATLAB command Window, type **`tl_upgrade(<mylib>)`**.

Result

The library is upgraded.

Converting Data from the DD Project File to the Info File

The dSPACE Data Dictionary replaces the Info file (`<subsystem>info.m`) that was available in former TargetLink versions. During code generation, all code generation related data is written to the dSPACE Data Dictionary. You can access the data created via the Data Dictionary Manager or the DD MATLAB API.

Creating an Info file

In TargetLink 2.0, you can still use API utilities written for TargetLink 1.3, provided that the migration steps described in this document are carried out. If your utilities are based on the old TargetLink Info file format, you can create such a file from the Data Dictionary data.

- The Info file cannot represent all the information in the dSPACE Data Dictionary. It is recommended to upgrade Info file based utilities to the Data Dictionary to take full advantage of TargetLink.
- In future versions, the Info file format will be discontinued.

How to Convert Data from the DD Project File to the Info File

Sometimes, you still need to work with the old, autonomous Info file format, because you cannot adapt the interface of tools that are used in a post-processing process and are based on the M script format. If so, you can convert data from the DD project file.

Precondition

The code generation process must be finished successfully.

Method

To convert data from the DD project file to the Info file

➤ In the MATLAB Command Window, type **`tl_generate_info_file`**.

All relevant data is converted to the Info file format. The name(s) of the Info file(s) is derived from the name(s) of the generated subsystem(s) in the DD project file. The data in the DD project file is not influenced by this command.

If the Info file is always needed, place the `tl_generate_info_file` command in the post-code generation hook function.

Result You can now use the Info file as usual.

Migration Features

The following features can influence the behavior of your model or the code generation process. You should therefore note these changes when migrating from TargetLink 1.3 to TargetLink 2.0.

Enhanced initial value checks

TargetLink 2.0 performs additional checks to ensure that signals can be represented by the specified data type and scaling. If an initial value of a block output or parameter is outside of the implemented range, an error occurs. Make sure that your model does not contain such invalid initializations.

You can calculate the implemented range from the data type and the scaling. For example, if the data type is UInt8 (integer values 0..255), the scaling is set to 2^-1 and the offset is set to 10, the implemented range is 10 ... 136.5. If the initial value is within this range, there is no message; otherwise a message is output.

Empty root functions

In TargetLink 2.0, empty root functions are not generated. To maintain an empty root function, you have to place a Function block in the system and select a non-default function class, for example, GLOBAL_FCN.

Base data types

Up to TargetLink 1.3, the generated code included `tl_types.h`, a header file that contained definitions such as those of base data types, i.e., the mapping of the platform-independent data types Int8, Int16, etc. to the C data types signed char, short int, etc. These header files were static and were always supplied in a form that fitted the targets supported by TargetLink.

To allow you to define your own type names or even the entire type definition file, a new concept was introduced. In TargetLink 2.0, the type definition file is generated. You can specify its name; the default is `tl_basetypes.h`. To generate the file and other target-dependent code patterns, the Code Generator requires information which it acquires from the `TargetConfig.xml` configuration file. TargetLink provides these configuration files for the supported targets. The targetDir folder containing the files is specified in `tl_global_options.m` in the codeopt structure.

The following example shows an excerpt of the tl_global_options.m file:

```
function codeopt = i_GetCodeopt(hostCompiler)
i = 1;
codeopt(i).name      = 'Generic ANSI-C';
codeopt(i).targetDLL = 'AnsiTarget.dll';
codeopt(i).target    = 'Generic';
codeopt(i).cc        = '';
codeopt(i).targetDir = ['SrcFiles\I86\' hostCompiler];
```

For details, refer to *Configuring Global Options* in the *Advanced Practices Guide*.

When you select a target optimization module under Code Generator options in the TargetLink Main Dialog, the right target folder is set automatically.

For the Generic ANSI-C code generation option, the specification of the target folder has to be adapted to the target you want to generate code for.

The target folder of the host compiler (32 bit) is the default, because TargetLink does not know your intended target processor if Generic ANSI-C code is selected. Note that the base type definitions for the host compiler will normally not be appropriate for your embedded target.

To generate code for a different target, you have to insert a new codeopt element in `tl_global_options.m`. For details, refer to *How to Add a Code Generator Configuration* in the *Advanced Practices Guide*.

Function reuse

In TargetLink 2.0, subsystems can be selected for function reuse by setting the **Make function reusable** flag in the Function block of the subsystem. The Function block must reside in a separate library and must have a fixed function name as in TargetLink 1.3. As a consequence, multiple occurrences of the reused subsystem in a TargetLink system is no longer necessary. The **Make function reusable** flag also allows the reuse of a subsystem that is used only once in the model. For details, refer to *Function Reuse* in the *Advanced Practices Guide*.

System interface for reused functions

TargetLink 2.0 requires a fully specified function interface using TargetLink InPort and OutPort blocks if you want to generate reused functions. For details, refer to *Preparing Functions for Reuse* in the *Advanced Practices Guide*.

If no TargetLink InPort block was used in TargetLink 1.3, the scaling and data types of interface input variables of a reused subsystem were inherited from the signal source blocks. This could lead to unpredictable results, as the implementation of the reused subsystem depends on blocks outside of the system.

Reuse of external functions

Functions which were not coded by TargetLink can be called by specifying the EXTER_FCN function class in the Function block. In TargetLink 1.3, you could simply specify the same function in several subsystems if you wanted to reuse it. In TargetLink 2.0, you have to use the same mechanism for reusing external functions as for functions that are not external. Otherwise, there are problems with inconsistent subsystems. That means the subsystems must be instances of the same library system, and you have to select the **Make function reusable** checkbox in the Function dialog.

Global variables at function interfaces

In TargetLink 2.0, the Code Generator implements function interfaces with global variables by default. That means if you select the *default* variable class for TargetLink inports or outports, you get global variables. In TargetLink 1.3, function parameters were generated. To specify a TargetLink InPort or OutPort as a function parameter, use one of the predefined classes:

- For TargetLink InPorts: FCN_ARG, or FCN_PARAM

- For TargetLink OutPorts: FCN_REF_ARG, FCN_REF_PARAM, or FCN_RETURN

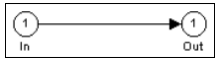
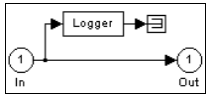
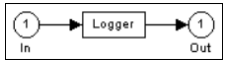
For details, refer to *How to Specify the Function Interface via Function Parameters* in the *TargetLink Advanced Practices Guide*.

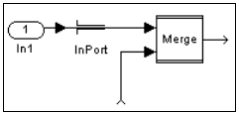
Subsystem interface implementation

TargetLink Inport and Outport blocks at the borders of a subsystem are treated as explicit function interface definitions. Even if they are specified as default ports, interface variables may remain in the code, though they seem to be superfluous. An optimized interface can be achieved by removing the blocks.

Changed block properties

Virtual TargetLink inports and outports In TargetLink 2.0, you can select if an InPort or OutPort block is virtual or nonvirtual. By default, TargetLink uses virtual inports and outports. TargetLink InPort and OutPort blocks are inserted at boundaries of TargetLink subsystems to define a clear interface for the generated C functions. They specify the names, data types, variable classes and scalings of the input and output signals. In addition, virtual TargetLink InPort and OutPort blocks enable signal label and data type propagation from the outside into the TargetLink subsystem. In previous TargetLink versions, the TargetLink InPort and OutPort blocks were always nonvirtual blocks. If this behavior is needed, you can clear the **Virtual port** checkbox on the **Logging & Autoscaling** page of the respective TargetLink InPorts and OutPort block. The table summarizes the characteristics of the different options for TargetLink InPorts and OutPorts.

	Virtual (Default for TargetLink 2.0)	**Virtual with Data Logging Enabled**	**Nonvirtual (Default for TargetLink 1.3.0)**
			
Internal implementation of TargetLink InPort/OutPort	Direct **virtual** connection	Direct **virtual** connection with branch to data logger	Direct **nonvirtual** connection with data logger
Overflow warning	No	Yes	Yes
Simulink data type propagation	Yes	Yes	No, signal labels cannot be propagated over nonvirtual connections

	Virtual (Default for TargetLink 2.0)	Virtual with Data Logging Enabled	Nonvirtual (Default for TargetLink 1.3.0)
Simulink signal label propagation	Yes	Yes	No, signal labels cannot be propagated over nonvirtual connections
	Same simulation behavior as in Simulink; nested InPorts are merged in one variable; Initial values must be equal.	Same simulation behavior as in Simulink; nested InPorts are merged in one variable; Initial values must be equal.	Simulation behavior may be different from Simulink; nested InPorts are not merged in one variable; initial values may be different.
Nested enabled subsystems	Supported	Not supported; *This case is not regarded as a relevant limitation, because normally the output of a Merge block is logged, not its inputs*	Supported
Merge Block directly connected to TargetLink InPort/OutPort 			

There is no general recommendation as to which solution is preferred.

Freeze flag In TargetLink 2.0, the `output.freeze` block property is a scalar flag and only interpreted by the autoscaling tool to determine if the offset can be adjusted. In former TargetLink versions, the freeze flag could be set for each element of the block output individually.

Custom Code blocks

In former TargetLink versions, a common section of a Custom Code block was placed in the ancestor function of the respective subsystems. In TargetLink 2.0, the common sections are moved inside the limits of atomic subsystems. For specific models, the old behaviour can be achieved by partitioning the Custom Code into the block-specific part and the model-common part. The new Custom Code block with the common code parts must be located in the top most subsystem belonging to the required model part.

Data store memory

In TargetLink 2.0, you can specify the name of the C code variable independently of the data store tag. The meaning of the output.name property has changed. As for other blocks, this property represents now the variable name in the C code. The following table lists the modified properties of the Data Store blocks:

Block name	Property	Property name in TL 1.x	Property name in TL 2.0
Data Store Memory	Variable name	-/-	output.name
Data Store Memory	Data store tag	output.name	datastorename
Data Store Read	Data store tag	output.name	datastorename
Data Store Write	Data store tag	dsmname	datastorename

Renamed block types

Block type identifier The string *FXPCG_* in the TargetLink 1.x block type names is replaced by *TL_*. The new block types are derived from the corresponding Simulink block type identifiers with the prefix *TL*, for example, *TL_Sum* instead of *FXPCG_SUM*. There are some blocks where this naming convention does not apply, for example:

- TL_Lookup1D
- TL_LogicalOperator
- TL_Interpolation

The TargetLink 1.x block type names are still accepted by TargetLink API functions. However, it is recommended to upgrade user-written M scripts to the new block type names.

To get a list of all supported block types, invoke the following command in the MATLAB Command Window:

```
tl_manage_blockset('GetValidBlocktypes')
```

You can also upgrade a TargetLink 1.x block type to the corresponding TargetLink 2.0 type. To do so, enter the following command in the MATLAB Command Window:

```
tl20Type = tl_manage_blockset('UpgradeBlocktype',tl13Type)
```

Simulation frame tag The tag 'FXPCG Subsystem' of the TargetLink simulation frame is renamed 'MIL Subsystem'.

The upgrade routine changes the tag for user models automatically.

Changed feedthrough behavior of Flip-Flops

The direct feedthrough behavior of the D Flip-Flop and the J/K Flip-Flop has changed. The D-Input and J/K Inputs do not have direct feedthrough in TargetLink 2.0. This corresponds to the Simulink implementation of the Flip-Flop blocks in MATLAB R12.1, which is different from MATLAB R11. For details, refer to *J-K Flip-Flop Block* and to *D Flip-Flop Block* in the *TargetLink Block and Object Reference*.

Initial condition of the Unit Delay block

The `output.initialcondition` property of the Unit Delay block has been renamed `state.initialcondition`. The state variable can now explicitly be specified in the block dialog.

During model upgrade from TargetLink version 1.x to 2.0, the block data is automatically upgraded.

You have to modify user-written API scripts accordingly.

For details on the Unit Delay block, refer to *Unit Delay Block* in the *TargetLink Block and Object Reference*.

Optimization of look-up blocks

The PreLook-Up Index Search and the Interpolation (n-D) Using PreLook-Up blocks are needed to split a look-up function into the index search and interpolation function. This can be done if a common axis with the same input signal was used at multiple Look-Up blocks. The two blocks provide the same behavior as in TargetLink 1.3, that is, the generated code for the interpolation routine can be reused for several search routines.

Unlike in TargetLink 1.3, reuse of interpolation routines for 1-D and 2-D Look-Up Table blocks is not automatically generated.

For details, refer to *How to Optimize Table Searches and Reuse Search Results* in the *Advanced Practices Guide*.

Getting properties of Stateflow objects

The `tl_get` function now returns the default value of a Stateflow property that was not set. In previous TargetLink versions, an empty matrix [] was returned. For details, refer to *tl_get* in the *TargetLink API Reference*.

TargetLink Autoscaling tool

The TargetLink Autoscaling tool calculates the scaling parameters for the variables of blocks of a specific model part. The scaling parameters are:

- Data type
- LSB
- Offset

The scaling parameters of a block are calculated with respect to the scaling parameters of neighboring blocks already calculated. The following input data is required for calculation:

- A variable's range, to be determined via:
  - Worst-case range calculation, or
  - Simulation
- Scaling parameters known in advance (optional)

The Scaling Page in the TargetLink 1.3 Main Dialog has been removed. There is a separate GUI for the Autoscaling tool, which can be invoked from the Tools page of the TargetLink Main Dialog or via the Tool Selector block.

Due to the significant enhancements of the Autoscaling tool, the API function `tl_autoscaling` has also changed. For details, refer to *tl_autoscaling* on page 69.

For details, refer to *Autoscaling Basics* in the *TargetLink Production Code Generation Guide* and to *Autoscaling Tool* in the *TargetLink Tool and Utility Reference*.

EXTERN_GLOBAL interface for SIL and PIL

The interfaces of the generated production code are often specified as EXTERN_GLOBAL variables because the input and output signals are provided in global variables from external code. To simulate such code in SIL and PIL in TargetLink 1.3, the variable definitions had to be provided in user-written files. The files then had to be linked to the simulation application by specifying them in Addfile blocks. In TargetLink 2.0, simulation frame generation automatically generates files with all the necessary variable definitions for the EXTERN_GLOBAL interfaces. Thus, it is not necessary to specify user-written files via Addfile blocks to avoid multiple definitions of variables.

ASAM-MCD 2MC File Generator

With ASAM-MCD 2MC, TargetLink supports a standard commonly used for describing access to ECU-internal data by calibration systems. In TargetLink 2.0, the following new features, enhancements and changes are supported.

A2L export via DD Manager

In TargetLink 2.0, you can use the Data Dictionary Manager or the Data Dictionary MATLAB API to export ASAM-MCD 2MC (ASAP2) files. Exporting A2L files via the Data Dictionary Manager works independently of TargetLink and does not even require a MATLAB installation. You can use XSL style sheets to customize the layout of the generated A2L files and add project-specific or module-specific information to the A2L file. The standard XSL style sheets are located in `%DSPACE_ROOT%\dsdd\A2L\Stylesheets`.

Limitations of the supported mechanisms

The ASAM-MCD 2MC (ASAP2) file generation mechanism of TargetLink 1.3, which was based on the `generate_ASAP2` API command and the template file mechanism, is still supported. However, the following limitations apply:

- The M file `if_data_asap1b_<ASAP1bInterfaceName>.m` used to generate the IF_DATA entry at CHARACTERISTIC, MEASUREMENT, AXIS_PTS is not evaluated. The corresponding XSL Style Sheet `a2l_if_data_<ASAP1bInterfaceName>.xsl` is used instead.

  If you have used such an M file to generate an IF_DATA entry, you have to migrate it to the corresponding XSL style sheet. For details, refer to *Specifying the ASAM-MCD 1b Interface* in the *dSPACE Data Dictionary Manager Reference*. With TargetLink 2.0, new predefined ASAP1b interfaces are available for:

  - CANAPE_EXT
  - DCI_GME1
  - ETK

  To use the CANape Extension, select both CCP and CANAPE_EXT in the ASAP2 template file variable *interfaceTypes*:

```
interfaceTypes = {
   'CCP'
   'CANAPE_EXT'};
```

- In TargetLink 2.0, a conversion table can be added via the Data Dictionary Scaling Object. The TargetLink 1.3 mechanism of adding a conversion table via the *tabList* vector of structures is no longer supported.
- The predefined ASAP2 template file variable *stInfo* is now a vector of structures instead of a simple structure as in TargetLink1.3. There is one `stInfo` element for each sampling rate in the system. If you use this variable in your template file, a loop through the variable is performed:

```
$tlASAP2('BeginLoop','stInfo', stInfo)
   /begin SOURCE
   …
   /end SOURCE
$tlASAP2('EndLoop')
```

MAPs and CURVEs axis and input quantity

In TargetLink 2.0, look-up table axes are generated as shared axes (COM_AXIS) by default. If such an axis is shared by more than one MAP or CURVE, the input quantity (if known) is generated only if it is identical for all MAPs and CURVEs. Otherwise, no information about the input quantity is specified at the AXIS_PTS entry (NO_INPUT_QUANTITY is generated). If available, the input quantity information is always generated at the AXIS_DESC entry of the MAP/CURVE CHARACTERISTIC.

For details, refer to *Preparing ASAM-MCD 2MC File Generation* in the *TargetLink Advanced Practices Guide* and to *ASAM-MCD 2MC File Format* in *dSPACE Data Dictionary ASAP2 Import and Export*.

Configuration Files

The configuration of TargetLink, that is, the available configuration files has changed since TargetLink 1.3. Some configuration files are obsolete, and new files and mechanisms have been introduced.

Project configuration file

TargetLink 1.x project configuration files (`*_cfg.m`) must be converted to the corresponding Data Dictionary project file. For details, refer to *Upgrading the Project Configuration File* on page 43 and *get_tloptions* on page 68.

sltype2tltype

The `sltype2tltype` configuration file is obsolete. If Simulink data types other than double are used for block outputs or block parameters, they are converted to corresponding TargetLink data types. The following table shows how Simulink data types are mapped to TargetLink data types:

Simulink Type	TargetLink Type
boolean	Bool
uint8	UInt8
int8	Int8
uint16	UInt16
int16	Int16
uint32	UInt32
int32	Int32
single	Float32
double	<DefaultDataType> (*)

The <DefaultDataType> can be specified in the Data Dictionary Manager, for example, `/Config/TargetLink/DefaultDataType='Int16'` or `'Float32'`.

Specifying the communication interface

The communication interface between your development PC and the evaluation board (EVB) for processor-in-the-loop (PIL) simulation is specified in an XML configuration file. The name of the file can be defined for each simulation configuration in the `tl_global_options.m` file, as shown below:

```
simconfig(i).name   = 'Cmd565/Diab50';
simconfig(i).board   = 'CMD565';
simconfig(i).cc   = 'DIAB50';
simconfig(i).myC   = 'MPC5xx';
simconfig(i).config   = 'TLSerPort.xml';
simconfig(i).makefile   = 'MakeFile.mk';
simconfig(i).compilerRootEnv   = 'MPC5XX_DIAB50_ROOT';
```

During the installation of TargetLink, you are asked for the COM port interface to be used for communication with the EVB. This information is written to the `%DSPACE%\Matlab\TL\config\tlsim\TLSerPort.xml` file, which is used as the default for all simulation configurations.

You can change the communication interface later by editing the COMPORT tag in the `TLSerPort.xml` file. It is also possible to create your own XML files, each of which specifies another comport or baud rate, for example. Thus, a specific XML file can be used for each target EVB. For details, refer to *Files Related to the Host-Target-Communication* on page 85 in the *TargetLink File and Message Reference*.

tl_get_block_config

The `forceNonVirtual` flag for TargetLink inports and outports in the block-specific configuration file `tl_get_block_config.m` is obsolete. In this TargetLink version, you can specify each TargetLink inport and outport as virtual or nonvirtual individually. For details, refer to *tl_get_block_config* in the *TargetLink API Reference*.

Changes to API Commands

For this version of TargetLink, some new API commands were added to TargetLink's API, while other API commands were renamed. In addition, there are some API functions that have changed or been given new enhanced options. This guide only describes the modifications that are relevant for migration from TargetLink 1.3 to TargetLink 2.0. For a complete description of all options, refer to the *TargetLink API Reference*.

For downward compatibility, the API functions used in TargetLink 1.3 are still available although it is strongly recommended that you migrate your utilities to the new API commands.

New and renamed API functions

The following list shows the renamed and new API commands.

TargetLink 2.0	TargetLink 1.3	Renamed	New
ds_error_check	tl_error_check	x	-/-
ds_error_clear	tl_error_clear	x	-/-
ds_error_display	tl_error_display	x	-/-
ds_error_get	tl_error_get	x	-/-
ds_error_log	tl_error_log	x	-/-
ds_error_merge	tl_error_merge	x	-/-
ds_error_msg	tl_error_msg	x	-/-
ds_error_none	tl_error_none	x	-/-
ds_error_register	tl_error_register	x	-/-
ds_error_set	tl_error_set	x	-/-
ds_msgdlg	tl_msgdlg	x	-/-
sllib2tllib	-/-	-/-	x
tl_code_coverage	-/-	-/-	x
tl_extract_subsystem	-/-	-/-	x
tl_get_blocks	get_tlblocks	x	-/-
tl_get_subsystem_info	get_tlsubsystem_info	x	-/-

TargetLink 2.0	TargetLink 1.3	Renamed	New
tl_global_options	get_tloptions	x	-/-
tl_pref	-/-	-/-	x
tllib2sllib	-/-	-/-	x
tl_build_standalone	-/-	-/-	x
tl_build_custom_code_sfcn	build_customcode_sfcn	x	-/-

For details, refer to *API Commands* in the *TargetLink API Reference*.

Changed API functionality

get_tloptions The `get_tloptions` function is obsolete. One part of it has been mapped to the new function `tl_global_options`. This file includes all options that are not project-specific. The project-specific configurations like variable classes and user-defined data types are now included in the dSPACE Data Dictionary.

In TargetLink 1.3, the `get_tloptions` function returned a MATLAB struct with the content of the project configuration file (`*_cfg.m`). In TargetLink 2.0, only the data from `tl_global_options.m` is returned. The following table shows the MATLAB struct returned by `get_tloptions` in TargetLink 1.3 and 2.0:

TargetLink 1.3	TargetLink 2.0	Remark
`options.codeopt`	`options.codeopt`	-/-
-/-	`options.rtosconfig`	-/-
`options.simconfig`	`options.simconfig`	-/-
`options.baseTypeNames`	-/-	Base types are defined as Typedef objects in `/Pool/Typedefs` in the dSPACE Data Dictionary
`options.userTypes`	-/-	User-defined data types are defined as Typedef objects in `/Pool/Typedefs` in the dSPACE Data Dictionary
`options.class`	-/-	Variable classes are defined in the Data Dictionary node `/Pool/VariableClasses`
`options.defaultClass`	-/-	Default variable classes are defined in the Data Dictionary node `/Pool/Templates/VariableClasses`
`options.functionClass`	-/-	Function classes are defined in the Data Dictionary node `/Pool/FunctionClasses`

TargetLink 1.3	TargetLink 2.0	Remark
`options.defaultfunctionClass`	-/-	Default function classes are defined in the Data Dictionary node `/Pool/Templates/FunctionClasses`
`options.editor`	`options.editor`	-/-
`options.gui`	`options.gui`	-/-

tl_pre_codegen_hook The MATLAB variable `tlSystemFile` is no longer supported in the `tl_pre_codegen_hook` function because the names of the generated C code modules are not known before code generation. If the root level of the TargetLink subsystem is completely virtual, there might be no code file at all for the root system.

tl_pre_conversion_hook / tl_post_conversion_hook The name of the variable which denotes the subsystem to be converted has changed. Use the `hSubSystem` identifier instead of `model` in your hook functions.

tl_autoscaling With TargetLink 2.0, autoscaling can not only be based on simulated ranges (as in TargetLink 1.3), but also on calculated worst-case ranges. In addition, you can autoscale single subsystems, not only models. For details, refer to *Worst-Case Autoscaling* on page 19.

The `LogFileName` property has been removed because all messages are written to the Message Browser, which also allows to create log files.

Because of the above changes, the API function `tl_autoscaling` has been revised. You can now perform several operations using an additional argument `action` which can take the values:

- `Init`
- `PropagateRanges`
- `CalculateRanges`
- `InheritScaling`
- `CalculateScaling`

The syntax was modified for this purpose as shown below:

Syntax 2.0: `tl_autoscaling(action, PropertyName, PropertyValue)`

Syntax 1.3: `tl_autoscaling(PropertyName, PropertyValue)`

In addition, some of the property names were changed. The changes are emphasized in the following table:

TargetLink 1.3 Property Name	TargetLink 2.0 Property Name
Model	*Subsystem*
ScaleOutputs	ScaleOutputs
ScaleStates	ScaleStates
ScaleParameters	ScaleParameters
SelectDatatypeSign	SelectDatatypeSign
MinimizeWordWidth	*MinParaWidth*
LogFileName	**----**

For details, refer to *Autoscaling Basics* in the *TargetLink Production Code Generation Guide* and to *Autoscaling Tool* in the *TargetLink Tool and Utility Reference*.

Last-Minute Information

This chapter provides information on changes and enhancements that were made after the TargetLink documentation was completed.

TargetLink Production Code Generation Guide

For information on last-minute changes concerning production code generation, refer to *Changes on the TargetLink Production Code Generation Guide* on page 73.

TargetLink Advanced Practices Guide

For information on last-minute changes concerning advanced practices, refer to *Changes on the TargetLink Advanced Practices Guide* on page 74.

TargetLink Multirate Modeling Guide

For information on last-minute information concerning multirate modeling, refer to *Changes on the TargetLink Multirate Modeling Guide* on page 75.

dSPACE Data Dictionary Basic Concepts Guide

For information on last-minute changes concerning the dSPACE Data Dictionary, refer to *Changes on the dSPACE Data Dictionary Basic Concepts Guide* on page 82.

TargetLink File and Message Reference

For information on last-minute changes concerning files and messages, refer to *Changes on the TargetLink File and Message Reference* on page 83.

For a list of open problems in the release version of TargetLink 2.0, refer to the Known Problems List %DSPACE_ROOT%\Doc\Print\TL_2_0_KnownProblemsList.pdf.

You can download the updated Known Problems List from http://www.dspace.de/goto?TargetLinkDocumentationUpdate. Please check the support area of the dSPACE homepage regularly for the latest TargetLink news.If you encounter any problem when using TargetLink, please send an email to support.tl@dspace.de.

Changes on the TargetLink Production Code Generation Guide

This chapter contains information on changes that were made after the TargetLink Production Code Generation Guide was completed.

Simulation start time must be 0

In Simulink, the simulation starts at the time x and ends at the time y. By default, x is 0, and y is 10 seconds.

You are not allowed to enter a start time x other than 0. If you do, the simulation is interrupted, and an error message displays.

Unit Delay Reset Enabled block

Concerning the Unit Delay Reset Enable block there are the following limitations:

- The outputs of a Unit Delay Reset Enabled block cannot be merged.
- The Unit Delay Reset Enable block does not support input signals with the Simulink's Boolean data type for Input u and Input IV.

Custom Code blocks

The variable classes for interface variables of Custom Code blocks are restricted. They must not specify const variables, macros, or function parameters of any kind.

Changes on the TargetLink Advanced Practices Guide

This chapter contains information on changes that were made after the TargetLink Advanced Practices Guide was completed.

Stateflow

Nested definition of graphical functions is not supported.

Inport and Outport blocks

If the signals are not merged due to the dataflow, using the same variable for Inport and Outport signals may lead to undesired behaviour. There is no check for detecting error situations.

Look-up tables

For look-up tables, dynamic structure components are not possible.

File restart functions

A file restart function may become empty when the respective variables are eliminated and thus all initializations are removed. The empty file restart function is currently not removed.

Changes on the TargetLink Multirate Modeling Guide

This chapter contains information on changes that were made after the TargetLink Multirate Modeling Guide was completed.

CounterAlarm block dialog

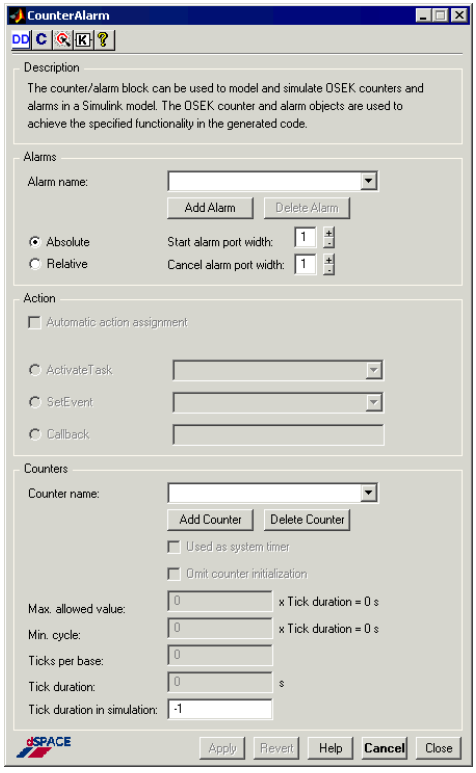
The dialog of the CounterAlarm block has been changed. Refer to *CounterAlarm Block Dialog* on page 76.

Limitations on multirate modeling

For additional information on limitations, refer to *Limitations on Multirate Modeling* on page 77.

CounterAlarm Block Dialog

The dialog of the CounterAlarm block has been changed. The following illustration shows the updated dialog:



In the updated dialog, the order of the Alarm, Action and Counter frames has been changed. Additionally, the updated dialog contains the **Used as system timer** checkbox. Selecting this checkbox lets you specify to use the specified counter as the system timer.

If you select the Used as system timer checkbox, the tick duration must not be -1.

Limitations on Multirate Modeling

In the section below you will find information on limitations that were changed or added after the TargetLink Multirate Modeling Guide was completed.

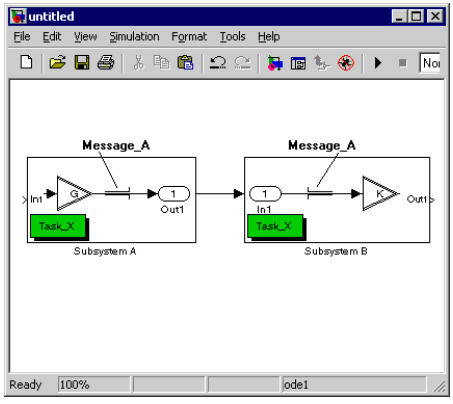
No ITC within a root step function

You cannot specify intertask communication for subsystems belonging to the same root step function.

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

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

Consider the following example model:



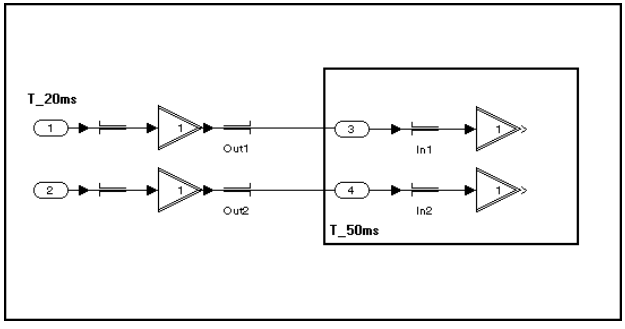
In this example, the bounds of subsystem A and subsystem B are root step function borders.

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

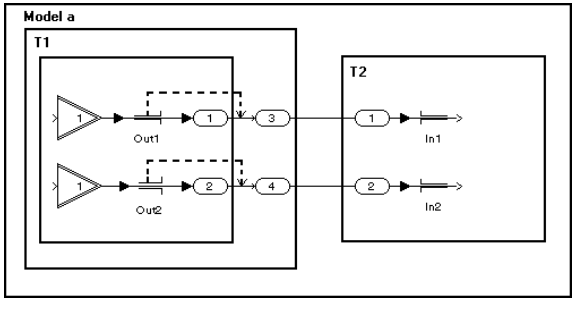
ITC at TargetLink InPort or OutPort blocks

The specification of intertask communication is ignored if the TargetLink InPort or OutPort blocks are not at task borders and thus are not recognized as task interface specifying blocks.

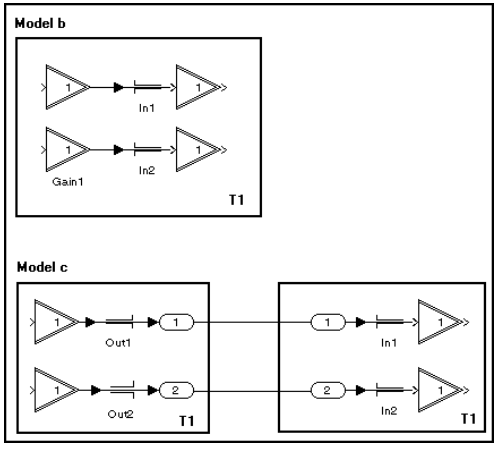
TargetLink InPort and OutPort blocks must be used as task interface specifying blocks if they are used to specify intertask communication. A task interface specifying block is a TargetLink InPort (OutPort) block with a one-to-one connection to a Simulink Inport (Outport) block. Additionally, the block sending (receiving) the signal resides in a different task. In the illustration below, In1 and In2 are task interface specifying blocks. Out1 and Out2 are not.



TargetLink InPort and OutPort blocks must be placed at the border of a task if they are used to specify intertask communication. In model a in the illustration below, In1 and In2 are at the border of a task. Out1 and Out2 are not. You can move Out1 and Out2 to the task border as shown below.



In models b and c in the following illustration, the TargetLink InPort and OutPort blocks are not at a task border because the sending and receiving real blocks reside in the same task.



Several solutions are possible:

- Specify intertask communication at the task interface specifying blocks (In1 and In2 in model a).
- If you want to specify intertask communication, move the TargetLink InPort (OutPort) blocks to the task border as shown in model a.
- Delete the specification of intertask communication if the sending and receiving real blocks are in the same task.

Protecting function-call triggered subsystems as critical section

You have to protect function-call triggered subsystems as critical sections if all of the following preconditions are fulfilled:

- The function-call triggered subsystem is triggered by several tasks/ISRs which can interrupt each other.
- The code resulting from the function-call triggerd subsystem uses global or static variables.

Generic RTOS simulation uses OSEK emulator

To execute a generic RTOS simulation in SIL and MIL mode, TargetLink must use a scheduler to schedule the different tasks containing the generated code. Since the generic RTOS simulation should cover a wide range of existing RTOSs, it cannot take in to account all specific features such as scheduling behavior, and so on. Therefore, the SIL and PIL simulation behavior might differ slightly from the real-time behavior in the RTOS you use. TargetLink uses the same simulation scheduler for generic RTOS simulation as for OSEK simulation. Since OSEK simulation provides some error and warning messages to detect modeling errors, it is possible that some of these OSEK warnings are displayed during generic RTOS simulation, too. This applies only to a few particular models.

SIL/PIL simulation not possible for externally triggered counters

You cannot perform SIL or PIL simulations for a model to which all of the following points apply:

- A counter is used in several CounterAlarm blocks.
- The counter is triggered externally.
- The counter is not used as the system timer.

This does not affect production code generation.

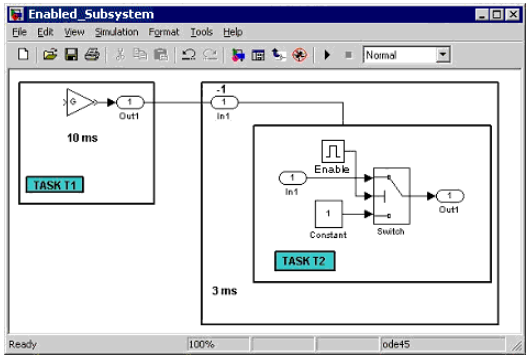
StartAlarm and CancelAlarm source

The trigger sources for the CounterAlarm block's StartAlarm and CancelAlarm inports must reside in the same TargetLink subsystem as the CounterAlarm block.

Show enable port is not allowed in special cases

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

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



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

No message called default

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

Execution time and stack consumption of multirate models

If you perform simulations for multirate models, both the execution time and stack consumption measured include the time and stack needed by the OSEK emulator. As a result, the time and stack consumption measurements are higher than actually required.

$S name macro

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

Sample time specification

A large sample time specification may lead to an error message claiming downsampling problems.

Changes on the dSPACE Data Dictionary Basic Concepts Guide

This chapter contains information on changes that were made after the dSPACE Data Dictionary Basic Concepts Guide was completed.

Admin password for DD files

If the current DD project file is not password protected and another DD project file is loaded, where the admin password is set, this password is inherited by the current DD project file. That means a user cannot get access rights for protected objects from the loaded file without knowing the admin password.

Without knowing the admin password you cannot delete write-protected DD objects. The only option to delete these objects is to clear the whole data dictionary.

Multiselection

Multiselection of objects in the Data Dictionary object tree of the Data Dictionary Manager is supported for only the Cut, Paste and Delete actions. With multiselection, Drag & Drop is not possible.

Components of variable objects

In the Data Dictionary Manager, it is possible to create struct components via the context menu of a variable object, even if the variable's base type is not struct.

Changes on the TargetLink File and Message Reference

You can download the updated document from the dSPACE Web site at http://www.dspace.de/goto?TargetLinkDocumentationUpdate and copy it into dSPACE HelpDesk for TargetLink 2.0.

To update dSPACE HelpDesk for TargetLink 2.0

To update TargetLink Error Messages

1. Close dSPACE HelpDesk for TargetLink 2.0.
2. Delete the `dSPACEHelpDesk.chw` file from the `%DSPACE_ROOT%/doc/online` folder.
3. Download the updated PDF and CHM files from http://www.dspace.de/goto?TargetLinkDocumentationUpdate.
4. Copy the PDF file to `%DSPACE_ROOT%/doc/print`, and the CHM file to `%DSPACE_ROOT%/doc/online`.
5. Open dSPACE HelpDesk for TargetLink 2.0.

Result This document is updated in dSPACE HelpDesk for TargetLink 2.0.

Key Features of the MATLAB R14 Compatibility Update for TargetLink 2.0

The MATLAB R14 Compatibility Update for TargetLink 2.0 enables you to use TargetLink 2.0 together with MATLAB and Simulink on the basis of The MathWorks Release 14. For simplicity, the term "MATLAB R14" is used to refer to The MathWorks Release 14 throughout the rest of this document. MATLAB R14 provides several new features. This document describes how to use the new features when working with TargetLink. For details on the new MATLAB features, refer to the MATLAB R14 documentation. Note that TargetLink does not support all new MATLAB features. For details on the limitations that apply when working with the MATLAB R14 Compatibility Update for TargetLink 2.0, refer to *Limitations in the MATLAB R14 Compatibility Update for TargetLink 2.0* on page 89.

When you work with the MATLAB R14 Compatibility Update for TargetLink 2.0, it is assumed that you have knowledge in handling TargetLink, and that you know the TargetLink Production Code Generation Guide and the TargetLink Advanced Practices Guide. It is also assumed that you know the MATLAB R14 Release Notes for MATLAB, Simulink, and Stateflow.

General changes

The MATLAB R14 Compatibility Update for TargetLink 2.0 has several implementation enhancements and changes to enable TargetLink 2.0 to be used with MATLAB R14. Refer to *General Enhancements and Changes* on page 87.

Supported Simulink features

The MATLAB R14 Compatibility Update for TargetLink 2.0 supports some of the new Simulink features. Refer to *Supported Simulink Features* on page 88.

General Enhancements and Changes

The MATLAB R14 Compatibility Update for TargetLink 2.0 provides the following general enhancements and changes.

TargetLink 2.0 support of MATLAB R14

After you install the MATLAB R14 Compatibility Update for TargetLink 2.0, TargetLink 2.0 supports MATLAB R14. For details on limitations, refer to *Limitations in the MATLAB R14 Compatibility Update for TargetLink 2.0* on page 89.

Support of TargetLink release

The MATLAB R14 Compatibility Update for TargetLink 2.0 supports exclusively the following dSPACE products:

- TargetLink 2.0

Compatibility of TargetLink models

TargetLink models created with TargetLink 2.0 or earlier provide full functionality when you work with the MATLAB R14 Compatibility Update for TargetLink 2.0. Nevertheless, you should be aware of the limitations described in *Limitations in the MATLAB R14 Compatibility Update for TargetLink 2.0* on page 89.

If you save a TargetLink model with the MATLAB R14 Compatibility Update for TargetLink 2.0, you can no longer use it with earlier TargetLink releases.

Supported Simulink Features

The MATLAB R14 Compatibility Update for TargetLink 2.0 supports the following new Simulink features:

- Rate Transition block
- Bus-capable nonvirtual blocks

Rate Transition block

The Rate Transition block has been modified. It now automatically distinguishes the type of transition between the source and destination block.

The MATLAB R14 Compatibility Update for TargetLink 2.0 supports the Rate Transition block.

Bus-capable nonvirtual blocks

Simulink now propagates busses through some nonvirtual blocks.

Propagated busses are supported by MATLAB R14 Compatibility Update for TargetLink 2.0 with some blocks, but not with others. The MATLAB R14 Compatibility Update for TargetLink 2.0 supports busses propagated through the following nonvirtual blocks.

- Merge
- Rate Transition
- Zero-Order Hold

Limitations that apply to the Simulink blocks apply to the corresponding TargetLink blocks as well.

Limitations in the MATLAB R14 Compatibility Update for TargetLink 2.0

You should note some limitations when you migrate from TargetLink 2.0 to the MATLAB R14 Compatibility Update for TargetLink 2.0.

Simulink 6.0

There are some new features of Simulink 6.0 which are not supported by this MATLAB R14 Compatibility Update for TargetLink 2.0. Refer to *Unsupported Features of Simulink 6.0* on page 91.

Unsupported new Simulink blocks

There are some new Simulink blocks which are not supported by this MATLAB R14 Compatibility Update for TargetLink 2.0. Refer to *Unsupported New Simulink Blocks* on page 95.

Stateflow 6.0

There are some limitations concerning Stateflow. Refer to *Unsupported Features of Stateflow 6.0* on page 96.

Limitations in the MATLAB R14 Compatibility Update for TargetLink 2.0

Software environment and compatibility

When you migrate to the MATLAB R14 Compatibility Update for TargetLink 2.0, there are some limitations in the software environment and compatibility. Refer to *Limitations in Software Environment and Compatibility* on page 98.

Graphical User Interface

There are some limitations in the appearance of TargetLink dialogs. Refer to *Limitations in the Graphical User Interface* on page 99.

Unsupported Features of Simulink 6.0

There is no support for the following new features of Simulink 6.0:

- Fixed-Point blockset
- Model referencing
- Bus-capable nonvirtual blocks
- Duplicate input ports
- Zero- and one-based indexing
- User-specifiable sampling times
- Enhanced Discrete-Time Integrator block
- Initial output handling
- States in function-call triggered subsystems
- State reset in conditionally executed subsystems
- Output reset in function-call triggered subsystems

Fixed-Point blockset

In MATLAB R14, fixed-point capabilities in MATLAB and Simulink have been extended. Two new products, Fixed-Point Toolbox and Simulink Fixed Point, take the place of the Fixed-Point blockset.

The MATLAB R14 Compatibility Update for TargetLink 2.0 provides only very restricted support for the new fixed-point capabilities.

For further information, refer to the *Data Types* application note.

Model referencing

With Simulink in MATLAB R14, you can include models in other models as blocks, which is called model referencing. This allows you to split large models into a number of smaller models saved in separate files.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support the model referencing feature and features based on it.

Bus-capable nonvirtual blocks

Simulink now propagates busses through some nonvirtual blocks.

Propagated busses are supported by MATLAB R14 Compatibility Update for TargetLink 2.0 with some blocks, but not with others. The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support busses propagated through the following nonvirtual blocks:

- Switch
- Multiport Switch
- Unit Delay

Duplicate input ports

With Simulink 6.0, models can contain several instances of the same Inport block. The duplicates have the same properties as the original Inport block they represent.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support duplicate input ports.

Zero- and one-based indexing

Simulink 6.0 introduces zero-based indexing for some Simulink blocks.

The indices of the following blocks now can optionally start at 0 or 1:

- For Iterator
- Assignment
- Selector
- Multiport Switch

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support zero-based indexing.

User-specifiable sampling times

Simulink 6.0 lets you specify a nondefault sampling time at most blocks.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support user-specifiable sampling times.

Enhanced Discrete-Time Integrator block

Simulink 6.0 adds new parameters and new values to existing parameters of the Discrete-Time Integrator block.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support:

- The 'Accumulation' values of the **Integrator Method**
- **Gain value**
- **Show state port**

To avoid problems with the Discrete-Time Integrator block, do not use these values or parameters.

Initial output handling

Simulink 6.0 introduces a new method of handling initial output values. In contrast to previous releases, a Simulink block computes its initial output value only when a connected block requests it.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support the new initial output handling but adopts the output initialization method of the previous Simulink releases. That is, the generated code represents blocks that write their initial output values to the connected blocks when the model is started.

With conditionally executed subsystems especially, the difference between Simulink's and TargetLink's initial output handling may result in deviations of the SIL simulation from the MIL simulation.

To avoid differences between MIL and SIL simulations, enter initial values at the Outport blocks of conditionally executed subsystems.

States in function-call triggered subsystems

Simulink 6.0 introduces 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, the MATLAB R14 Compatibility Update for TargetLink 2.0 does not support the 'held' and 'reset' settings of the States when enabling property.

To avoid problems with function-call triggered subsystems, select 'inherit'.

State reset in conditionally executed subsystems

The Simulink state reset behavior has changed from the MATLAB version R13 to R13 (Service Pack 1) and is not yet supported by the MATLAB R14 Compatibility Update for TargetLink 2.0.

Differences may show up especially with conditionally executed subsystems that are activated by multiple callers.

When one, some, or all of the calling subsystems send an enable trigger to a conditionally executed subsystem, Simulink checks the sequence of the previous disable triggers from all of the calling subsystems. The states of the called subsystem are reset only if all the calling subsystems were set to disabled, or already had been set to disabled, in the time step before the enable trigger to the conditionally executed subsystem.

To avoid problems with conditionally executed subsystems:

- With all conditionally executed subsystems except function-call triggered subsystems, set the **States when enabling** property of the called subsystem to 'held'.

  Only with function-call triggered subsystems, set the **States when enabling** property of the called subsystem to 'inherit' (refer to *States in function-call triggered subsystems* on page 93) and make sure that the **States when enabling** property of the calling subsystems is set to 'held'.
- Try to make sure that the conditionally executed subsystems are called only via a single path.

Output reset in function-call triggered subsystems

Simulink 6.0 introduces the **Output when disabled** property for Outport blocks in function-call triggered subsystems. This lets you influence the way the output is dealt with when a subsystem is disabled.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support the 'reset' setting of the **Output when disabled** property.

To avoid problems, set the **Output when disabled** property of the Outport block in the function-call triggered subsystem to 'held'.

Unsupported New Simulink Blocks

There is no support for the following new Simulink blocks:

- Bus Assignment block
- Embedded MATLAB Function block

Bus Assignment block

Simulink 6.0 introduces the Bus Assignment block, which allows single signal values to be assigned to signals in the bus.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support the Bus Assignment block.

Embedded MATLAB Function block

Simulink 6.0 introduces the Embedded MATLAB Function block, which lets you include embedded MATLAB functions in Simulink models.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support the Embedded MATLAB Function block.

Unsupported Features of Stateflow 6.0

There is no support for the following new features of Stateflow 6.0:

- Type casting with cast and type
- Inheritance of data type and size
- Data sized by expression
- Matrix inputs and outputs for graphical functions
- Embedded MATLAB functions
- Stateflow boxes data
- BIND actions

Type casting with cast and type

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support Stateflow's new `cast` and `type` operators.

Where possible, convert cast operations into the following form:

```
y = uint16(x+3)
```

Inheritance of data type and size

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support Stateflow's data type and size inheritance from Simulink.

When working with TargetLink, you have to enter the data type and size of Stateflow input and output explicitly.

Data sized by expression

You can use expressions in the Size property for Stateflow data. The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support this.

Matrix inputs and outputs for graphical functions

Stateflow 6.0 introduces vectors and two-dimensional matrices as data sizes of inputs and outputs for Stateflow graphical functions.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support vectors or two-dimensional matrices for data sizes.

Embedded MATLAB functions

Stateflow 6.0. now lets you include embedded MATLAB functions in Stateflow charts.

As TargetLink cannot generate code for M scripts, the MATLAB R14 Compatibility Update for TargetLink 2.0 does not support embedded MATLAB functions. Use graphical functions instead.

Stateflow boxes data

Data can now be defined in Stateflow box objects.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support data defined in box objects.

Instead of defining data in Stateflow boxes, define them on a higher level, for example, in the surrounding Stateflow chart.

BIND actions

A Bind action ties specified data and events to a state, called a binding state. If a function-call event is tied to a state, the associated function-call triggered subsystem is included. This means that entering the binding state enables the function-call triggered subsystem and exiting the binding state disables it.

The MATLAB R14 Compatibility Update for TargetLink 2.0 does not support Bind actions.

Limitations in Software Environment and Compatibility

The following limitations apply to the software environment and compatibility.

Unsupported MATLAB versions

After you install the MATLAB R14 Compatibility Update for TargetLink2.0, your TargetLink installation no longer supports MATLAB R12.1 and MATLAB R13.x (R13.0.1, R13SP1 and R13SP1+). If you still need to use MATLAB R12.1 or MATLAB R13.x, you must create another installation and switch between them using the dSPACE Installation Manager.

The MATLAB R14 Compatibility Update for TargetLink 2.0 supports exclusively the following dSPACE products:

- TargetLink 2.0

Limitations in the Graphical User Interface

The following limitations apply to the graphical user interface.

Display errors in TargetLink dialogs

Due to a problem in MATLAB R14, the following display errors can occur in TargetLink dialogs which are implemented as MATLAB Handle Graphics dialogs:

- With Windows XP, disabled edit fields look enabled.
- With Windows XP, the background color of dialogs is incorrect.
- Control names appear slightly outside of frames.
- Checkboxes have no right border.
- Frames might be slightly displaced on dialog pages.
- Text might be truncated at the top.
- The alignment of controls might be slightly displaced.

You can avoid some problems that occur with Windows XP by selecting “Windows Classic” from the Theme list on Windows’ Display Properties dialog.

These display errors do not impact the TargetLink dialogs’ functionality. All controls work correctly.