TargetLink

API Reference

For TargetLink 5.1

Release 2020-B – November 2020

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

Contents

About this Reference

Contents

This reference provides information on:

- Application programming interface (API) commands of TargetLink: The API functions provide quick access to TargetLink commands as well as letting you automate specific TargetLink tasks.
- TargetLink API data types: They describe which values can be set for specified properties.
- Custom look-up functions: TargetLink provides a scripting mechanism for replacing TargetLink look-up functions with custom look-up functions.

Note

Not all API functions are documented. These API functions are subject to change without prior notice. Do not use or alter undocumented API functions as they might not be compatible or even supported in later versions.

Symbols

dSPACE user documentation uses the following symbols:

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

API Functions

Where to go from here

Information in this section

Overview of API Functions

Alphabetical List of API Functions

API Functions	Purpose
`ddv`	Retrieves the value of a DD Variable object.
`ds_error_check`	Returns number of messages of specified severity.
`ds_error_clear`	Clears messages in TargetLink's message system.
`ds_error_display`	Displays messages.
`ds_error_get`	Returns parameters of TargetLink's message system.
`ds_error_get('BatchModePrintMessage')`	Returns TargetLink's batch mode print status
`ds_error_get('GroupedIndices')`	Returns indices of registered messages grouped by type
`ds_error_get('DefaultExcludedMessages')`	Returns the message numbers which are initially excluded for display in the Message Browser
`ds_error_get('CurrentState')`	Returns all data of TargetLink's message system
`ds_error_get('AllMessages')`	Returns all messages in a message struct
`ds_error_get('EmptyMessageStruct')`	Returns an empty message struct (constructor function)
`ds_error_get('MessageStruct', propertyName, propertyValue, ...)`	Creates message struct and sets specified fields (constructor function)
`ds_error_get('BatchMode')`	Returns TargetLink's batch mode
`ds_error_get('Message', msgNum)`	Returns msgNum-th message from TargetLink's message system
`ds_error_log`	Writes messages to the logfile.
`ds_error_merge`	Merges messages.
`ds_error_msg`	Displays a message, and/or registers it in TargetLink's message system.
`ds_error_none`	Clears messages.
`ds_error_register`	Registers message in TargetLink's message system.
`ds_error_set`	Sets parameters of TargetLink's message system.

API Functions	Purpose
ds_msgdlg	Manages the TargetLink Message Browser.
ds_msgdlg('update', propertyName, propertyValue, ...)	Creates/updates Message Browser with new list of messages, clears previous display
ds_msgdlg('callback', propertyName, propertyValue, ...)	Invokes control callback
ds_msgdlg('show')	Shows Message Browser
ds_msgdlg('clear')	Clears messages in Message Browser
ds_msgdlg('close')	Hides Message Browser
ds_msgdlg('delete')	Deletes Message Browser
ds_msgdlg('find')	Finds Message Browser
dsdd_check_msg	Checks for errors and messages after the Data Dictionary has been accessed.
dsdd_compare_optionset	Compares the code generator options of a model with an option set.
dsdd_export_a2l_file	Exports an A2L file from the Data Dictionary.
dsdd_export_optionset	Transfers the option set properties to a model.
dsdd_free	Clears TargetLink's Data Dictionary repositories.
dsdd_get_block_path	Returns the Simulink or Stateflow path associated with a DD object.
dsdd_get_creator_options	Returns options used to create the specified Data Dictionary subsystem object.
dsdd_get_width	Returns the value of the Width property associated with a specified variable value.
dsdd_import_optionset	Creates a code generator option set in the TargetLink Data Dictionary
dsdd_manage_application	Manages DD Application objects.
dsdd_manage_application('GetApplication', propertyName, propertyValue, ...)	Gets application name for given model/subsystem
dsdd_manage_application('SetApplication', propertyName, propertyValue, ...)	Sets application name for given model/subsystem
dsdd_manage_application('UpdateConfig', propertyName, propertyValue, ...)	Updates application configuration (nested subsystems)
dsdd_manage_application('GetSubsystems', propertyName, propertyValue, ...)	Returns nested systems hierarchy of specified system
dsdd_manage_application('CheckSubsystems', propertyName, propertyValue, ...)	Checks if subsystems belonging to an application are compatible to target/compiler/basetypes/library function calls

API Functions	Purpose
`dsdd_manage_build`	Creates a DD Build object.
`dsdd_manage_build('Create', propertyName, propertyValue, ...)`	Creates a DD Build object with the specified name in the specified DD Application object tree as follows: ▪ Sets the platform-specific properties of the embedded `<Application>/<Build>/TargetInfo` object on the basis of the `TargetInfo.xml` file. ▪ Imports the descriptions of the base types from the `TargetConfig.xml` file into the `<Application>/<Build>/TargetInfo/BaseTypes` object. DD Build objects created with this function can be used for A2L file generation. The location of the `TargetInfo.xml` and `TargetConfig.xml` files can be specified directly or by means of the compiler and evaluation board names.
`dsdd_manage_build('ImportSymbolTable', propertyName, propertyValue, ...)`	Imports symbol table from MAP file to specified DD Build object

API Functions	Purpose
`dsdd_manage_project`	Manages DD project. DD project files are always loaded into DD0.
`dsdd_manage_project('Open', projectFile)`	Opens the DD project file. The file is looked for on the MATLAB search path. If it does not exist, it is created from a DD template file which is selected by the user.
`dsdd_manage_project('GetProjectFile', simulinkSystem)`	Gets the name of the project file associated with the current model or from the global preferences
`dsdd_manage_project('SetProjectFile', projectFile, simulinkSystem)`	Sets the name of the project file associated with the current model
`dsdd_manage_project('Close', propertyName, propertyValue, ...)`	Closes DD project
`dsdd_manage_project('Save')`	Saves DD0 to DD project file
`dsdd_manage_project('SaveAs', projectFile, propertyName, propertyValue, ...)`	Saves DD0 to specified file. If no file is specified, the command opens a Save File dialog. The new file becomes the current DD project file.
`dsdd_manage_project('MdlPostLoadFcn', simulinkSystem)`	Opens DD project file associated with specified model. This command is called in TargetLink models' PostLoadFcn callback.
`dsdd_manage_project('MdlPreSaveFcn', simulinkSystem)`	Saves DD project associated with model as specified with TargetLink's ProjectFileAutosave option. This command is called up in TargetLink models' PreSaveFcn callback.
`dsdd_manage_project('Check', projectFile)`	Validates the /Pool and /Config area of DD0, or the specified DD project file.
`dsdd_manage_project('ClearAll')`	Clears DD. Corresponds with DD MATLAB API ClearAll command.
`dsdd_manage_project('SaveCopyAs', projectFile)`	Saves DD0 to specified file in snapshot mode, which means that DD0 and all included subtrees are saved to one file. The snapshot DD file does not become the current DD project file.
`dsdd_validate`	Frontend to DD validation.

API Functions	Purpose
`dsddman`	Command-line interface to Data Dictionary Manager.
`dsddman('CloseView', propertyName, propertyValue, ...)`	Closes the custom output view
`dsddman('AddMessage', propertyName, propertyValue, ...)`	Adds message to the Message Browser
`dsddman('AddCustomMessage', propertyName, propertyValue, ...)`	Adds a message to custom output view
`dsddman('Refresh')`	Refreshes Data Dictionary Manager (except Model Browser and Difference Browser)
`dsddman('Open', file)`	Opens a DD file and displays its content
`dsddman('Compare', propertyName, propertyValue, ...)`	Compares two DD project files or DD workspaces.
`dsddman('IsGuiOpen')`	Checks whether the DD Manager UI is open
`dsddman('ReloadMenuExtensionSpecification')`	Reloads the files that contain the specifications of the menu extensions into the Data Dictionary Manager
`dsddman()`	Opens the Data Dictionary Manager UI
`dsddman('Edit', objectIdentifier)`	Opens dialog to edit specified object
`dsddman('Select', objectIdentifier)`	Selects specified object and displays it
`dsddman('GetSelected')`	Gets the selected object in the Data Dictionary Navigator
`dsddman('DemandCustomOutputView', propertyName, propertyValue, ...)`	Creates a custom output view
`dsddman('ClearView', propertyName, propertyValue, ...)`	Clears the custom output view
`get_tlsubsystems`	Returns identifiers of TargetLink subsystems in Simulink system.
`get_tlsystemID`	Returns the system ID of the specified TargetLink code generation unit (CGU).
`set_tlsystemID`	Sets the system ID of the TargetLink subsystem.

API Functions	Purpose
`tl_access_logdata`	Accesses logged simulation data on the TargetLink Data Server.
`tl_access_logdata('GetSimulationLabels')`	Gets labels of all simulations saved in RAM
`tl_access_logdata('GetLoggedSignal', propertyName, propertyValue, ...)`	Returns logged simulation data (struct, Timeseries or Simulink.TimeSeries object). The tl_access_logdata('GetLoggedSignal', ...) command can apply several filters to the logged simulation data and returns the cut set.
`tl_access_logdata('GetLoggedSignalInfo', propertyName, propertyValue, ...)`	Returns signal-specific information for logged simulation data. The tl_access_logdata('GetLoggedSignalInfo', ...) command can apply several filters to the logged simulation data and returns the cut set.
`tl_access_logdata('PlotSignal', propertyName, propertyValue, ...)`	Displays logged simulation data in TargetLink's Plot Overview Window
`tl_access_logdata('SetNumberOfBufferedSimulinkLogSamples', numberofsamples)`	Limits number of samples for Simulink log buffer used for logging in MIL simulation. This affects the performance and the memory consumption. A shorter number of samples decreases memory consumption but increases simulation duration. The number of TargetLink log samples is not limited by this command.
`tl_access_logdata('GetNumberOfBufferedSimulinkLogSamples')`	Gets current number of samples for Simulink log buffer
`tl_access_logdata('CalculateNumberOfBufferedSimulinkLogSamples', model)`	Calculates and sets recommended number of samples for Simulink log buffer for a specific model. The following model attributes affect the calculation: ▪ Number of TargetLink blocks ▪ Number of TargetLink subsystems in MIL mode ▪ Log settings of TargetLink blocks ▪ Global logging option
`tl_access_logdata('GetLastSimulationLabel')`	Gets label of last simulation
`tl_access_logdata('GetLoggedBlocks', simlabel)`	Gets paths of all TargetLink blocks with logged simulation data
`tl_access_logdata('GetSimulationInfo', simlabel)`	Gets simulation info (model, time, start time, stop time, lock, tlsubsystems).
`tl_access_logdata('DeleteSimulation', simlabel)`	Deletes simulation
`tl_access_logdata('SetSimulationLabel', simlabel, newlabel)`	Replaces existing simulation label(s) with new label(s)
`tl_access_logdata('SetSimulationLock', simlabel, lock)`	Sets simulation lock
`tl_access_logdata('SaveSimulation', simlabel, filename)`	Saves simulation to file
`tl_access_logdata('LoadSimulation', filename)`	Loads simulation from file
`tl_addsimframe`	Adds a TargetLink simulation frame to the TargetLink subsystem.

API Functions	Purpose
`tl_autoscaling`	Calculates (worst-case) ranges and scaling parameters for output, state and parameter variables of TargetLink blocks.
`tl_autoscaling('init', propertyName, propertyValue, ...)`	Initializes autoscaling data structure (ASDS)
`tl_autoscaling('propagateRanges', propertyName, propertyValue, ...)`	Propagates all known range limits: i.e., constrained limits and limits derived from valid scaling parameters
`tl_autoscaling('calculateRanges', propertyName, propertyValue, ...)`	Calculates worst-case ranges for all blocks within specified scope
`tl_autoscaling('inheritScaling', propertyName, propertyValue, ...)`	Inherits scaling parameters of all blocks marked as valid, as far as possible within specified scope
`tl_autoscaling('calculateScaling', propertyName, propertyValue, ...)`	Calculates scaling parameters based either on worst-case ranges or ranges determined via simulation
`tl_build_customcode_sfcn`	Generates and compiles the custom code S-function for the Custom Code block.
`tl_build_host`	Generates production code and builds a SIL simulation application for specified TargetLink subsystems.
`tl_build_standalone`	Generates a stand-alone S-function for specified TargetLink subsystems
`tl_build_target`	Generates production code and builds a PIL simulation application for specified TargetLink subsystems.
`tl_check_module_ownership`	Checks the specification of the module ownership for the given system.
`tl_check_usertypes`	Checks user types in Simulink model.
`tl_clean`	Deletes all files generated by TargetLink.
`tl_clear_system`	Clears the Simulink system (subsystem, model, or library) from TargetLink.
`tl_code_coverage`	Generates code coverage documents.
`tl_codesize`	Evaluates the RAM/ROM consumption of the generated code.
`tl_compare_fcn_signature`	Compares the current interface of the reference model with the reference.
`tl_compile_host`	Builds an S-function and/or simulation application for SIL simulation.
`tl_compile_target`	Builds the simulation application for the target EVB.
`tl_create_blacklist`	Due to a limitation of the Merge block, logging and overflow detection during MIL simulation are not possible for blocks that are connected to Merge block input ports. If these blocks are not directly connected to the Merge block, but through virtual blocks, subsystem borders or bus signals, TargetLink might not detect them before logging or overflow detection starts. In this case, the simulation stops with an error.

API Functions	Purpose
tl_demos	Opens and/or restores TargetLink demos.
tl_download	Loads production code applications to EVBs or MATLAB memory space.
tl_export_container	Exports a container for exchanging AUTOSAR data with SystemDesk
tl_export_files	Exports files generated by TargetLink to a separate directory.
tl_find	Searches for TargetLink blocks with specified property values.
tl_generate_code	Calls the Code Generator for the specified code generation units (CGUs).
tl_generate_customcode_tlc	Builds TLC scripts for TargetLink Custom Code blocks.
tl_generate_fmu	Generates a FMU for the specified TargetLink subsystem.
tl_generate_swc_model	Generates/updates TargetLink subsystem from description of software components.
tl_generate_vecu_implementation	Generates a V-ECU implementation for the specified TargetLink subsystem.
tl_get_block_config	Provides block configuration options not accessible via TargetLink API or UI.
tl_get_blocks	Compiles a list of TargetLink blocks belonging to a given class.
tl_get_checksum	Calculates the checksum of a Simulink model file or an arbitrary ASCII file.
tl_get_config_path	Returns the search path for configuration files and hook scripts.
tl_get_mlfcnobjects	Compiles a list of MATLAB code model elements of a specified class.
tl_get_sfobjects	Compiles a list of Stateflow objects belonging to a given class.
tl_get_sim_mode	Returns the simulation mode(s) of TargetLink subsystem(s) in a model.
tl_get_subsystem_info	Provides information about TargetLink subsystem.
tl_get_userdata	Extracts data embedded in the description string.
tl_get	Retrieves TargetLink properties of blocks and Stateflow objects.
tl_global_options	Returns global configuration options for TargetLink.
tl_manage_blockset	Provides information about TargetLink blocksets.
tl_pack_model	Bundles all files required to work with TargetLink model in a non-TargetLink environment
tl_pref	Returns and changes TargetLink preferences settings.
tl_prepare_system	Prepares the Simulink system (subsystem, model, or library) for TargetLink.

API Functions	Purpose
`tl_refmodel_to_subsystem`	Converts referenced models into subsystems configured for incremental code generation.
`tl_removesimframe`	Removes TargetLink simulation frame(s).
`tl_repair_busdata`	Checks and corrects the TargetLink data of bus port blocks.
`tl_set_sim_mode`	Switches the simulation mode(s) of specified TargetLink subsystems.
`tl_set`	Modifies TargetLink properties of blocks and Stateflow objects.
`tl_sim`	Starts the simulation for a TargetLink model with logging support for referenced models.
`tl_subsystem_to_refmodel`	Converts subsystems configured for incremental code generation into referenced models.
`tl_sync_system`	Synchronizes Simulink scaling data with TargetLink properties, or vice versa.
`tl_tl2rti`	Prepares simulation of production code in dSPACE prototyping environment.
`tlCodeCoverage`	Configures the code coverage analysis of generated production code and generates the code coverage report.
`tlCodeCoverage('Set', system, propertyName, propertyValue, ...)`	Configurates code coverage analysis
`tlCodeCoverage('Get', system, propertyName)`	Returns current value of specified code coverage configuration property
`tlCodeCoverage('Get', system)`	Returns current code coverage configuration
`tlCodeCoverage('GetCovTools')`	Gets supported code coverage tools
`tlCodeCoverage('MoveCTCFiles', propertyName, propertyValue, ...)`	Moves CTC files (*.sym and *.dat) created during build process and simulation to the specified folder. These files can later be used to create a combined report of code coverage analysis for multiple builds and simulations. By default, each time an application is (re)built, the existing CTC files are moved from the TargetLink build folder into the .\CTCRESULTS\TLBuild_<BuildTimeStamp> folder. See also: GenerateCombinedReport
`tlCodeCoverage('GenerateCombinedReport', propertyName, propertyValue, ...)`	Generates combined report from all CTC files (*.sym and *.dat) residing in and below the specified folders. See also: MoveCTCFiles
`tlCodeCoverage('GenerateReport', model, propertyName, propertyValue, ...)`	Generates report of code coverage analysis with selected code coverage tool
`tlCodeGenerationMetadata`	Saves and loads code generation metadata.
`tlCreateMATLABFunctionDDObjects`	Generates DD objects for specifying internal MATLAB code variables and MATLAB sub-functions.
`tlCustomizationFiles`	Creates customization files.

API Functions	Purpose
`tldoc`	Generates HTML or PDF documentation of the production code generated by TargetLink, and/or the simulation results.
`tldoc('Create', propertyName, propertyValue, ...)`	Creates new documentation that subsequent tldoc commands can add information to. The encoding of the HTML documentation files is UTF-8.
`tldoc('Overview', docFid, propertyName, propertyValue, ...)`	Adds general information to documentation
`tldoc('TargetLink Code Generation Units', docFid, propertyName, propertyValue, ...)`	Adds information about the specified TargetLink code generation units to documentation. Code generation units can be TargetLink subsystems, referenced models, or DD CodeGenerationUnit objects in Data Dictionary.
`tldoc('Simulation Results', docFid, propertyName, propertyValue, ...)`	Adds information about specified simulation results to documentation
`tldoc('Close', docFid, propertyName, propertyValue, ...)`	Closes generated documents
`tldoc('Convert', propertyName, propertyValue, ...)`	Converts generated documentation from HTML to PDF format.
`tlEnumDataType`	Imports a Simulink enumeration data type to the Data Dictionary.
`tlExecuteDDCustomCommand`	Executes a DD CustomCommand or CustomCommandGroup object.
`tlExtractSubsystem`	Generates a new independent TargetLink subsystem from a TargetLink subsystem part.
`tlFindDDReferences`	Returns TargetLink blocks, Stateflow objects and DD objects referencing a certain DD object.
`tlFindHook`	Searches for TargetLink hook scripts whose names match the specified pattern.
`tlGenerateSic`	Generates a dSPACE Simulink implementation container using the TargetLink Code Generator.
`tlGetArtifactLocation`	Returns the location of the code generation unit artifact.
`tlIsCodeGenerationInProgress`	Determines if the code generation process is active.
`tlMoveDDObject`	Moves or renames a DD object and adapts references to this object.
`tlOperationMode`	Gets or sets TargetLink's operation mode.
`tlProductionCodeSILCompiler`	Specifies the compiler to be used for building SIL simulation application.
`tlPromoteProperty`	Provides TargetLink properties in TargetLink blocks masks for Simulink promote mechanism.
`tlPropman`	Command-line interface to the TargetLink Property Manager.
`tlRebuildFixedPointLibrary`	Rebuilds the TargetLink Fixed-Point Library.
`tlRequirementInfo`	Manages requirement information at TargetLink model elements.

API Functions	Purpose
`tlSimInterface`	M interface for the TargetLink simulation engine.
`tlSimInterface('ConnectToSimPlatform', propertyName, propertyValue, ...)`	Connects to the specified simulation platform. Further actions relate to the production code simulation application running on the simulation platform.
`tlSimInterface('DisconnectFromSimPlatform', hPlatform)`	Disconnects from the specified simulation platform.
`tlSimInterface('Reset', hPlatform)`	Resets the production code simulation application.
`tlSimInterface('DisableResetBySimulationSFcn', hPlatform)`	Disables a reset of the production code application by a simulation S-function at simulation start (SIL/PIL). TargetLink automatically enables reset after simulation (SIL/PIL) finishes.
`tlSimInterface('EnableResetBySimulationSFcn', hPlatform)`	Enables a reset of the production code application by a simulation S-function previously disabled by the DisableResetBySimulationSFcn command.
`tlSimInterface('DisableRestartBySimulationSFcn', hPlatform)`	Disables a call of TargetLink subsystem-specific main restart functions (if any) at start of the simulation of the production code simulation application (SIL/PIL). TargetLink automatically enables a call of restart functions after simulation (SIL/PIL) finishes.
`tlSimInterface('EnableRestartBySimulationSFcn', hPlatform)`	Enables a call of TargetLink subsystem-specific main restart functions at start of the simulation of the production code simulation application (SIL/PIL) previously disabled by the DisableRestartBySimulationSFcn command.
`tlSimInterface('GetDDVarAddr', hPlatform, propertyName, propertyValue, ...)`	Obtains the address information of variables defined in the production code simulation application and specified by the DD identifier (handle or path) of their corresponding Variable objects in the Subsystem area.
`tlSimInterface('GetBlockVarAddr', hPlatform, propertyName, propertyValue, ...)`	Obtains the address information of variables defined in the production code simulation application and specified by the block it is generated from.
`tlSimInterface('GetMapVarAddr', hPlatform, propertyName, propertyValue, ...)`	Obtains the address information of variables defined in the production code simulation application and specified by their names and the name of the module they are defined in.
`tlSimInterface('GetFcnAddr', hPlatform, propertyName, propertyValue, ...)`	Obtains the address information of functions defined in the production code simulation application and specified by their names and the name of the module they are defined in.
`tlSimInterface('Read', hPlatform, propertyName, propertyValue, ...)`	Reads the values of variables specified by their info structures (returned by one of the address evaluation commands).

API Functions	Purpose
`tlSimInterface('Write', hPlatform, propertyName, propertyValue, ...)`	Writes values to variables specified by their info structures (returned by one of the address evaluation commands). If the limits (Min/Max components of the info structure) are set, this function issues a warning if the value to be written exceeds the given limits.
`tlSimInterface('CallFcn', hPlatform, propertyName, propertyValue, ...)`	Calls a function specified by its info structure (as returned by GetFcnAddr command). Only functions with the prototype void <fcnName>(void) can be called. Otherwise the application will crash.
`tlSimInterface('IsApplDownloaded', propertyName, propertyValue, ...)`	Checks whether production code simulation application was loaded to specified simulation platform
`tlSimParameterUpdate`	Modifies the online parameter automatically
`tlSimParameterUpdate('UpdateClasses', model, TLSubsystem, inputList, hPlatformHandle, propertyName, propertyValue, ...)`	Updates all values of variables of the specified classes
`tlSimParameterUpdate('UpdateVariables', model, TLSubsystem, inputList, hPlatformHandle, propertyName, propertyValue, ...)`	Updates all values of variables specified by DD variable paths
`tlSimulinkBusObject`	Processes Simulink.Bus objects in TargetLink.
`tlStartCallbackWithTimer`	starts a callback within a timer
`tlSyncSystemSignature`	Synchronizes a specified Data Dictionary Signature or Block object with a specified Simulink system.
`tlTransformerError`	provides functionality helpful for simulation of AUTOSAR transformer errors
`tlTransformerError('CreateSimulinkBusObject', propertyName, propertyValue, ...)`	Creates a Simulink.Bus object defining a bus needed for transformer error simulation. The following rule determines where this object is created: ▪ No model -> MATLAB Base Workspace ▪ Model without Simulink Data Dictionary -> MATLAB Base Workspace ▪ Model with Simulink Data Dictionary -> model's Simulink Data Dictionary If a file is specified, the bus definition is exported to it.
`tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...)`	Creates Simulink.Signal objects with the specified names as required for transformer error simulation. A suitable Simulink.Bus object is also created. The following rule determines where these objects are created: ▪ No model -> MATLAB Base Workspace ▪ Model without Simulink Data Dictionary -> MATLAB Base Workspace ▪ Model with Simulink Data Dictionary -> model's Simulink Data Dictionary If a file is specified, the definitions of the Simulink objects are exported to it.

API Functions	Purpose
`tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)`	Adds stimuli to the specified model for each specified Simulink.Signal object as needed for transformer error simulation. The stimuli of single Simulink.Signal objects are composed of Constant, Bus Creator and Data Store Write blocks. The Simulink.Signal objects are not created. To create them, call the CreateSimulinkSignalObjects command. If no model is specified, the stimulus blocks are added to the newly created model and can be copied to the final destination model, for example.
`tlTransformerError('PrepareModelForSimulation', propertyName, propertyValue, ...)`	Prepares the transformer error simulation for the specified AUTOSAR software components. This command performs the following steps: 1. Obtaining the names of the Simulink.Signal objects associated with the transformer errors to be simulated from the Data Dictionary. 2. Creating the required Simulink.Bus and Simulink.Signal objects. 3. Adding suitable stimulus blocks to the specified model that are needed to stimulate different values of the transformer errors. The names of the Simulink.Signal objects are taken from the TransformerErrorSignalLabel property set at the DD ComSpec objects below the DD ReceiverPort or DD SenderPort object whose ErrorHandling property is set to 'TransformerErrorHandling'.
`tlUpgrade`	Upgrades block diagrams containing TargetLink blocks to the current version.

Configuring TargetLink

Where to go from here

Information in this section

tl_global_options

tl_global_options

Purpose

Returns global configuration options for TargetLink.

Syntax

```
options = tl_global_options(request)
```

Input parameters

The following input parameters are available:

Parameter	Description
`request`	Specifies the requested part of TargetLink options. The following values are possible: ▪ `'codeopt'` - Available code generation targets (Generic ANSI C, TOM_XXX). ▪ `'rtosconfig'` - RTOS configurations. ▪ `'simconfig'` - Target simulation configurations. ▪ `'editor'` - Code editor configurations. ▪ `'gui'` - User interface options. ▪ `'all'` - Returns a MATLAB struct combining all global options. ▪ `'refresh'` - Refreshes the option cache and returns all global options.

Output parameters

The following output parameters are available:

Parameter	Description
`options`	Struct containing (part of) global TargetLink options

Example

```
% get all global options
allOptions = tl_global_options;
allOptions =tl_global_options('refresh');

% get all simulation configurations
simconfigList =tl_global_options('simconfig');
```

Related topics

Basics

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

tl_pref

tl_pref

Purpose

Returns and changes TargetLink preferences settings.

Syntax overview

The following syntaxes are available:

`tl_pref('gui')`

Opens TargetLink Preference Editor

`prefStruct = tl_pref('get')`

Returns all preferences as structure

`prefValue = tl_pref('get', prefName)`

Returns value of specified preference

`tl_pref('set', prefName, prefValue)`

Sets certain preference to new value

Input parameters

The following input parameters are available:

Parameter	Description
`prefName`	Preference identifier
`prefValue`	<NewValue>

Output parameters

The following output parameters are available:

Parameter	Description
`prefStruct`	Structure containing each preference as separate field
`prefValue`	Value of requested preference

Remarks

The following preferences are available:

- 'ProjectFile' - Specifies default DD file name.
- 'ProjectFileAutosave' - Controls the autosave mechanism for DD files.
  - 'on' - Saves the DD automatically when the model is saved.
  - 'interactive' - Opens the Save As dialog.
  - 'off' - Does not automatically save the DD file.
- 'Editor' - Specifies the editor to be used to edit code files.
- 'CodeCovProgressBar' - Shows the progress bar during code coverage tests.
- 'SyncSLScaling' - Synchronizes Simulink and Stateflow scaling properties with TargetLink data.
- 'Sync<what_to_sync>' - Synchronizes specific Simulink and Stateflow properties:
  - 'OutputScalingData' - Scaling data specified as a Simulink property of the block.
  - 'SignalScalingData' - Effective scaling data of signals that can result from inheritance.
  - 'SaturationFlags' - Saturation flags.
  - 'ConstrainedLimits' - Minimum and maximum values.
  - 'ParameterScalingData' - Scaling data of parameters.
  - 'SFObjectScalingData' - Scaling data specified as a Stateflow property of a Stateflow variable.
  - 'SFObjectCompiledScalingData' - Effective scaling data of Stateflow variables that can result from inheritance.
  - 'RtwData' - RTW settings as far as they can be associated with TargetLink settings.
- 'DialogProvider' - Specifies whether a double-click opens the TargetLink/Simulink dialog.
- 'DDTemplateDirectory' - Specifies the directory for user DD project file templates.
- 'DDFilterRuleDirectory' - Specifies the directory for DD Filter Rule files.

- 'DDMenuExtensionDirectory' - Specifies the directory for DD Menu Extension files.
- 'A2LStyleSheetDirectory' - Specifies the directory for A2L style sheet files.
- 'FixedPointLibrarySourcesDirectory' - Specifies the folder for fixed-point library sources.
- 'FixedPointLibraryBinariesDirectory' - Specifies the folder for fixed-point library binaries.

Related topics

Basics

Customizing the TargetLink Environment (📖 TargetLink Customization and Optimization Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

References

Customization Files (📖 TargetLink File Reference)
Fixed-Point Library Files (📖 TargetLink File Reference)
Preferences Editor (📖 TargetLink Tool and Utility Reference)
TargetLink Main Dialog Block (📖 TargetLink Model Element Reference)

tlCustomizationFiles

tlCustomizationFiles

Purpose

Creates customization files.

Syntax overview

The following syntaxes are available:

```
tlCustomizationFiles('Create', files, path, prefix, makeSubfolder)
```

Copies specified files to a specified directory and adds a specified prefix to all hook files. File extension of *.sam files is changed to *.m

```
tlCustomizationFiles('CopyTemplates', path, prefix)
```

Copies all customization template files as samples to a specified directory and adds a specified prefix to all hook files

```
nameList = tlCustomizationFiles('GetList')
```

Returns the names of all available customization files

Input parameters

The following input parameters are available:

Parameter	Description
`files`	Name of file (cell array of strings if there are multiple files)
`path`	Directory the files are copied to. If no path is specified, the current working directory is used.
`prefix`	File name prefix (only used for hook files)
`makeSubfolder`	If true, files will be saved in subfolders determining their kind

Output parameters

The following output parameters are available:

Parameter	Description
`nameList`	Cell array containing names of all available customization files

Example

```
% copy the file tl_pre_codegen_hook as M script into the current
% working directory
tlCustomizationFiles('Create','tl_pre_codegen_hook')

% copy all hook and configuration files as samples to a folder
% 'MyFolder' in the current working directory and add 'MyPrefix' as
% a prefix to all hook file names
tlCustomizationFiles('CopyTemplates','MyFolder','MyPrefix')
```

Remarks

You can specify special file names to generate certain files:

- 'A2LStyleSheets' - creates all style sheets needed for A2L export
- 'CodeOutputStyleDefinitionFile' - creates the style definition file for code output formatting
- 'CodeOutputStyleSheets' - creates all style sheets for code output formatting
- 'DDMenuExtension' - creates the DD menu extension file

To be able to use the created hook or configuration files you must ensure the following:

1. The path leading to the file is part of MATLAB's current search path and is listed in tl_get_config_path.m.
2. The path leading to tl_get_config_path.m is also part of MATLAB's search path.

To create tl_get_config_path.m, use tlCustomizationFiles('Create', 'tl_get_config_path', <path>).

Related topics

Basics

Basics on Code Formatting (📖 TargetLink Customization and Optimization Guide)
Basics on Using Hook Scripts (📖 TargetLink Customization and Optimization Guide)

HowTos

How to Create Customization Files via the Create Customization Files Dialog (📖 TargetLink Customization and Optimization Guide)
How to Edit the Style Definition File (📖 TargetLink Customization and Optimization Guide)

References

Customization Files (📖 TargetLink File Reference)

Setting TargetLink's Search Path for M-API Related Customization Files

tl_get_config_path

tl_get_config_path

Purpose

Returns the search path for configuration files and hook scripts.

Description

You have to derive this API function from its template via the `tlCustomizationFiles()` API function. Editing the `tl_get_config_path.m` file lets you do the following:

- Add directories to the TargetLink search path.
- Remove directories from the TargetLink search path.

Syntax

```
cfgPath = tl_get_config_path()
```

Output parameters

The following output parameters are available:

Parameter	Description
`cfgPath`	Cell array of directory names that constitute the search path

Example

```
tlConfigPath =fileparts(which('tl_get_config_path'));
cfgPath ={
    pwd
    tlConfigPath
'd:\CommonProjectFiles\HookScripts'
'd:\CommonProjectFiles\CodeFormatingFiles'
};
```

Remarks

This file must reside in the current working directory or be on the MATLAB search path.

The directories on the TargetLink search path containing configuration M files and hook scripts must be on the MATLAB search path.

Related topics

Basics

Basics on Replacing Simulink Blocks via Libmaps (📖 TargetLink Preparation and Simulation Guide)
Deriving Customization Files From Their Templates (📖 TargetLink Customization and Optimization Guide)

HowTos

How to Define TargetLink's Search Path for M-API-Related Customization Files (📖 TargetLink Customization and Optimization Guide)

References

Hook Scripts (📖 TargetLink File Reference)

Specifying Artifact Locations

Where to go from here

Information in this section

tlCodeGenerationMetadata

tlCodeGenerationMetadata

Purpose

Saves and loads code generation metadata.

Syntax overview

The following syntaxes are available:

```
[bError, msgStruct] = tlCodeGenerationMetadata('Save', propertyName, propertyValue, ...)
```

Saves CGU-specific code generation metadata, e.g., the DD Subsystem object, to the directory specified by the DD ProjectFolder and StructureFolder objects in /Pool/ArtifactsLocation.

```
[bError, msgStruct] = tlCodeGenerationMetadata('Load', propertyName, propertyValue, ...)
```

Loads CGU-specific code generation metadata, e.g., the saved DD Subsystem object, to the active Data Dictionary. The DD ProjectFolder and StructureFolder objects in /Pool/ArtifactsLocation are used to get the location of the directory containing the metadata.

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`CodeGenerationUnits`	List of identifiers of code generation units whose metadata is to be saved/loaded. The identifiers can be: ▪ A code generation unit name (for model-based CGUs) ▪ A DD path or DD handle (for DD-based CGUs)
`Model`	Name of the root model containing the model-based code generation unit whose code generation metadata is to be loaded. Default: none

Property	Description
`IncludeSubItems`	Specifies whether metadata for nested CGUs of the specified CGU is loaded as well. The following values are possible: ▪ `'on'` - Metadata is loaded. ▪ `'off'` - Metadata is not loaded. (default)
`Overwrite`	Specifies whether existing metadata is overwritten with the data in the file. The following values are possible: ▪ `'on'` - Metadata is overwritten. (default) ▪ `'off'` - Metadata is not overwritten.

Output parameters The following output parameters are available:

Parameter	Description
`bError`	Indicates whether an error occurred: ▪ `1` - An error occurred. ▪ `0` - No error occurred.
`msgStruct`	A structure containing message information. It has the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note'). ▪ number - Message number. ▪ title - Message title. ▪ msg - Message text. ▪ objectName - Simulink/Stateflow/DD object related to the message. ▪ objectHandle - Handle of the Simulink/Stateflow/DD object. ▪ module - Name of the module (M file) where the message occurred. ▪ fcn - Name of the subfunction in module (M file). ▪ line - Line in the module (M file) where the message occurred. ▪ clock - Date and time when the message occurred. ▪ confirmed - '1' if the user has confirmed the message. Otherwise, '0'. ▪ objectKind - The kind of the object related to the message.

tlGetArtifactLocation

tlGetArtifactLocation

Purpose

Returns the location of the code generation unit artifact.

Description

This function returns the location of the artifact of the specified type for the specified code generation unit. It gets the artifact location from the specification in the DD ProjectFolder and FolderStructure objects in /Pool/ArtifactsLocation. If the artifact type is not specified, the path to the project folder is returned.

Syntax

```
[artifactLocation, artifactRelPath, cguRootPath, bError, msgList] =
tlGetArtifactLocation(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	**Description**
`ArtifactType`	Type of the artifact whose location to return. Possible values are the names of the properties of the DD FolderStructure object (with the exception of the description property) For example: ProductionCodeFiles, StubCodeFiles, etc.
`CodeGenerationUnit`	Code generation unit for which to return an artifact location. Specified as follows: ▪ As a DD path or DD handle (for DD-based CGUs) ▪ As a code generation unit name (for model-based CGUs) ▪ As a DD path or DD handle (for the Subsystem object)
`NameMacrosStruct`	Structure specifying the values of the name macros to use. The struct component name corresponds to the name of the name macro. You can specify the value for the following name macros: ▪ $(Board) ▪ $(Compiler) The function automatically gets the other name macros, $(CGU) and $(Variant).

Output parameters

The following output parameters are available:

Parameter	Description
`artifactLocation`	Absolute path of the folder containing the code generation unit artifact.
`artifactRelPath`	Location of the code generation unit (CGU) artifact relative to the project folder.
`cguRootPath`	Path of the project folder of the specified code generation unit.
`bError`	Specifies whether an error occurred: ▪ `1` - An error occurred. ▪ `0` - No error occurred.
`msgList`	List of message structures.

Example

```
% Example 1
% Get the location of the production code files of the code generation unit.
% ModelBasedCGU
[prodCodeLocation,~,~,bError,msgStruct]=tlGetArtifactLocation('CodeGenerationUnit','ModelBasedCGU',....
'ArtifactType','ProductionCodeFiles');
% Register and display all messages, notes, warnings and errors, if any.
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display();
end
if bError
return;
end


% Example 2
% Get the location of the object files of the ModelBasedCGU code generation unit in the TestModel model
% generated for the active PIL simulation.
pilSimConfig             =tl_get_target_simconfig('TestModel');
nameMacroStruct.Board    = pilSimConfig.board;
nameMacroStruct.Compiler = pilSimConfig.cc;
[objFilesLocation,~,~,bError,msgStruct]=tlGetArtifactLocation('CodeGenerationUnit','ModelBasedCGU',....
'ArtifactType','ObjectFilesArchive',...
'NameMacrosStruct',nameMacroStruct);
% Register and display all messages, notes, warnings and errors, if any.
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display();
end
if bError
return;
end
```

Accessing Demo Models

tl_demos

tl_demos

Purpose

Opens and/or restores TargetLink demos.

Syntax overview

The following syntaxes are available:

```
tl_demos()
```

Opens an overview page presenting all available TargetLink demos

```
tl_demos(name)
```

Opens a particular TargetLink demo

```
tl_demos(name, '-restore')
```

Restores all files belonging to a particular demo and opens the demo

```
tl_demos('-restore')
```

Restores all TargetLink demos to their initial state

Input parameters

The following input parameters are available:

Parameter	Description
`name`	Name of the directory containing the demo files
`'-restore'`	Initiates a restore of demo files

Preparing Systems for TargetLink

Where to go from here

Information in this section

tl_prepare_system

tl_prepare_system

Purpose

Prepares the Simulink system (subsystem, model, or library) for TargetLink.

Syntax

```
[options, msgStruct] = tl_prepare_system(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`system`	Simulink system (subsystem, model, or library) to be prepared for TargetLink
`prepSystem`	Name of file the model is saved to after preparation. Placeholder '$' stands for name of original system. When used, name of file also applies to related libraries (with PrepareLibs = on) and referenced models (PrepareRefMdls = on). The property is ignored when Save = off. Default: $_tl

Property	Description
`save`	Saves model after preparation. Default: 'off'
`makeTLSubsystem`	Makes specified subsystem a TargetLink subsystem. Ignored for models, libraries, subsystems in libraries, and subsystems in TargetLink subsystems. Default: 'on'
`mapOutputScalingData`	Maps Simulink block output scaling data to TargetLink properties. Default defined in TargetLink preferences (property SyncOutputScalingData).
`mapSaturationFlags`	Maps Simulink SaturateOnIntegerOverflow block parameter to TargetLink output.checkmin and output.checkmax properties. Default defined in TargetLink preferences (property SyncSaturationFlags).
`mapConstrainedLimits`	Maps Simulink OutMin and OutMax block parameters to TargetLink output.min and output.max properties. Default defined in TargetLink preferences (property SyncConstrainedLimits).
`mapParameterScalingData`	Maps Simulink parameter scaling data to TargetLink properties. Default defined in TargetLink preferences (property SyncParameterScalingData).
`mapSignalScalingData`	Maps Simulink signal data (e.g., the blocks' compiled data types) to TargetLink properties. Default defined in TargetLink preferences (property SyncSignalScalingData). Ignored for libraries.
`mapSFObjectScalingData`	Maps Stateflow data object scaling data to TargetLink properties of Stateflow objects. Default defined in TargetLink preferences (property SyncSFObjectScalingData).
`mapSFObjectCompiledScalingData`	Maps Stateflow object compiled scaling data to TargetLink properties of Stateflow objects. Default defined in TargetLink preferences (property SyncSFObjectCompiledScalingData). Ignores libraries.
`mapRTWData`	Maps Simulink RTW data to TargetLink properties. Default defined in TL preferences (property SyncRTWData).
`enhancePorts`	Enhances ports at root level. Ignored when system is a library. Default: 'on'
`prepareLibs`	Prepares related user libraries. All specified options also apply to libraries. Default: 'off'
`prepareRefMdls`	Prepares related referenced models. All specified options also apply to referenced models. Default: 'off'
`restoreTLData`	Restores block data saved during TargetLink 2.x reconversion, or when system was cleared from TargetLink. Default: 'off'
`logFileName`	Writes messages to logfile. Default: tl_prepare_system.log
`verbose`	Be verbose in MATLAB Command Window. Default: 'off'

Property	Description
`ignoreErrors`	Proceeds with preparing if error. Default: 'off'
`enhancePortSubsystems`	Subsystem(s) in specified system whose ports are to be enhanced
`processInactiveVariants`	Processes inactive subsystem variants. Default: 'off'
`insertFcnBlock`	Inserts a TargetLink Function block in subsystems whose ports were enhanced (as specified with the enhancePortSubsystems option). Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`options`	Struct that contains options used
`msgStruct`	Struct that contains messages

Example

```
tl_prepare_system('system','model/controller','MakeTLSubsystem','on');

[tmp,msgStruct]=tl_prepare_system('system','model/controller','MakeTLSubsystem','on');
ds_error_register(msgStruct);
ds_msgdlg('update');
```

Remarks

If called without output arguments, the TargetLink Message Browser opens after preparation and displays messages that have been produced. If called without input and output arguments, the System Preparation dialog opens, enabling you to prepare Simulink systems interactively.

Related topics

References

tl_clear_system.. 50
tl_post_preparation_hook (📖 TargetLink File Reference)
tl_pre_preparation_hook (📖 TargetLink File Reference)
tl_sync_system.. 52

tl_clear_system

tl_clear_system

Purpose

Clears the Simulink system (subsystem, model, or library) from TargetLink.

Syntax

```
[options, msgStruct] = tl_clear_system(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`system`	Simulink system (subsystem, model, or library) to be cleared from TargetLink data
`clearedSystem`	Name of file the model is saved to after clearing. Placeholder '$' stands for name of the original system; when used, name of file also applies to cleared libraries (with ClearLibs = on) and referenced models (ClearRefMdls = on). Ignored when Save = 'off' Default: $_tl
`save`	Saves model after clearing. Default: 'off'
`remapOutputScalingData`	Remaps TargetLink properties to Simulink block output scaling data. Default defined in TargetLink preferences (property SyncOutputScalingData).
`remapSaturationFlags`	Remaps TargetLink output.checkmin and output.checkmax properties to Simulink SaturateOnIntegerOverflow block parameter. Default defined in TargetLink preferences (property SyncSaturationFlags).
`remapConstrainedLimits`	Remaps TargetLink output.min and output.max properties to Simulink OutMin and OutMax block parameters. Default defined in TargetLink preferences (property SyncConstrainedLimits).
`remapParamScalingData`	Remaps TargetLink properties to Simulink parameter scaling data. Default defined in TargetLink preferences (property SyncParameterScalingData).
`remapSFScalingData`	Remaps TargetLink properties of Stateflow objects to Stateflow data object scaling data. Default defined in TargetLink preferences (property SyncSFObjectScalingData).
`clearLibs`	Clears related user libraries. All specified options also apply to libraries. Default: 'off'

Property	Description
`clearRefMdls`	Clears related referenced models. All specified options also apply to referenced models. Default: 'off'
`processInactiveVariants`	Processes inactive subsystem variants. Default: 'off'
`removeSimFrameOnly`	Removes TargetLink simulation frames only. Remapping scaling data also possible. Default: 'on'
`saveTLData`	Saves TargetLink data to enable re-preparation without data loss. Does not delete TargetLink utility blocks. Default: 'off'
`logFileName`	Writes messages to logfile. Default: tl_clear_system.log
`verbose`	Be verbose in MATLAB Command Window. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`options`	Struct that contains options used for preparation
`msgStruct`	Struct that contains messages

Example

```
tl_clear_system('system','model/controller','saveTLdata','on');

[tmp,msgStruct]=tl_clear_system('system','model/controller','saveTLdata','on');
ds_error_register(msgStruct);
ds_error_display;
```

Remarks

If called without output arguments, the TargetLink Message Browser opens after the tool has finished and displays messages that have been produced. If called without input and output arguments, the Clear System Dialog opens, enabling you to clear Simulink systems interactively.

Related topics

Basics

Basics on Clearing TargetLink Data from Simulink Systems (📖 TargetLink Preparation and Simulation Guide)

HowTos

How to Clear All TargetLink Data (📖 TargetLink Preparation and Simulation Guide)

References

Clear System From TargetLink Dialog (📖 TargetLink Tool and Utility Reference)
System Preparation Hook Scripts (📖 TargetLink File Reference)
tl_prepare_system.. 47
tl_sync_system.. 52

tl_sync_system

tl_sync_system

Purpose

Synchronizes Simulink scaling data with TargetLink properties, or vice versa.

Syntax

```
[options, msgStruct] = tl_sync_system(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`system`	Simulink system (subsystem, model, or library) to be processed
`TL2SL`	If 'on', Simulink data is synchronized with TargetLink data If 'off', TargetLink data is synchronized with Simulink data. Default: 'off'
`syncOutputScalingData`	Synchronizes Simulink block output scaling data with TargetLink properties (TL2SL = on), or vice versa. Default defined in TargetLink preferences (property SyncOutputScalingData).

Property	Description
`syncSaturationFlags`	Synchronizes the Simulink SaturateOnIntegerOverflow block parameter with TargetLink output.checkmin and output.checkmax properties (TL2SL = on), or vice versa. Default defined in TargetLink preferences (property SyncSaturationFlags).
`syncConstrainedLimits`	Synchronizes Simulink OutMin and OutMax block parameters with TargetLink output.min and output.max properties (TL2SL = on), or vice versa. Default defined in TargetLink preferences (property SyncConstrainedLimits).
`syncParameterScalingData`	Synchronizes Simulink parameter scaling data with TargetLink properties (TL2SL = on), or vice versa. Default defined in TargetLink preferences (property SyncParameterScalingData).
`syncSignalScalingData`	Synchronizes TargetLink properties with Simulink signal data (e.g., the blocks' compiled data types). Default defined in TargetLink preferences (property SyncSignalScalingData). Ignored for libraries and with TL2SL = 'on'.
`syncSFObjectScalingData`	Synchronizes the Stateflow data object scaling data with TargetLink properties of Stateflow objects (TL2SL = on), or vice versa. Default defined in TargetLink preferences (property SyncSFObjectScalingData).
`syncSFObjectCompiledScalingData`	Synchronizes TargetLink properties of Stateflow objects with Stateflow object-compiled scaling data. Default defined in TargetLink preferences (property SyncSFObjectCompiledScalingData). Ignored for libraries and with TL2SL = 'on'.
`syncRTWData`	Synchronizes TargetLink properties with Simulink RTW data. Default defined in TL preferences (property SyncRTWData). Ignored with TL2SL = 'on'.
`syncLibs`	Enum that describes how library blocks should be processed. Possible values are: ▪ 'off' - no library block is processed. ▪ 'AllRelated' - Synchronizes related user libraries. All the specified options also apply to the libraries. ▪ 'SelectedInstances' - Synchronize scaling data of selected library instances and propagate it into the library. Default: 'off'
`libInstances`	List of library instances to be processed if 'syncLib' property is set to 'SelectedInstances'. Default: {}
`processInactiveVariants`	Processes inactive subsystem variants. Default: 'off'
`logFileName`	Writes messages to logfile. Default: tl_sync_system.log
`verbose`	Be verbose in the MATLAB Command Window. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`options`	Struct that contains options used for synchronization
`msgStruct`	Struct that contains messages

Example

```
tl_sync_system('system','model/controller');

[~, msgStruct]=tl_sync_system('system','model/controller');
ds_error_register(msgStruct);
ds_error_display;
```

Remarks

If called without output arguments, the TargetLink Message Browser opens after the tool has finished and displays messages that have been produced. If called without input and output arguments, the System Synchronization dialog opens, enabling you to synchronize data interactively.

Related topics

References

tl_clear_system	50
tl_prepare_system	47

tl_addsimframe

tl_addsimframe

Purpose

Adds a TargetLink simulation frame to the TargetLink subsystem.

Description

This function inserts a TargetLink subsystem into a TargetLink simulation frame for SIL/PIL simulation. If the specified system is not a TargetLink subsystem, the function aborts and displays an error. If the subsystem already resides in a simulation frame, the function exits without action. A message is displayed which informs the user that a simulation frame was established. If invoked without output arguments, messages are displayed in the TargetLink Message Browser.

Syntax

```
[msgStruct, hFrameBlock, hTLSubsystem] = tl_addsimframe(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`system`	TargetLink subsystem to be placed into simulation frame. Default: []

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	TargetLink message struct
`hFrameBlock`	Handle of frame subsystem (i.e., subsystem visible to the user)
`hTLSubsystem`	Handle of TargetLink subsystem placed into simulation frame; not equal to TargetLink subsystem specified as input

Example

```
% remove all TargetLink simulation frames in model TL_Fuelsys, and have the Message Browser
% display messages about which simulation frames were removed
tl_removesimframe('system','TL_Fuelsys');

% re-add simulation frames to all TL subsystems in model, and have messages displayed in Message Browser
hTLSubsystems =get_tlsubsystems('TL_Fuelsys');
fori=1:numel(hTLSubsystems)
    msgStruct =tl_addsimframe('subsystem',hTLSubsystems{i});
ds_error_register(msgStruct);
end
ds_msgdlg('update','title','Adding Targetlink Simulation Frame');
ds_msgdlg('show');
```

Related topics

References

tl_removesimframe.. 56

tl_removesimframe

tl_removesimframe

Purpose

Removes TargetLink simulation frame(s).

Description

This function removes the simulation frames of all the TargetLink subsystems contained in the specified system. The TargetLink subsystem IDs remain unchanged. If no TargetLink simulation frames exist in the system, the function exits without action. For each removed frame, a message is generated. If the function is invoked without output parameters, the messages are displayed in the TargetLink Message Browser.

Syntax

```
msgStruct = tl_removesimframe(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`system`	Simulink system to be processed. Default: []
`verbose`	Be verbose in MATLAB Command Window. Default: '0'

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	TargetLink message struct

Example

```
% remove all TargetLink simulation frames in model TL_Fuelsys, and have the Message Browser
% display messages about which simulation frames were removed
tl_removesimframe('system','TL_Fuelsys');

% remove TargetLink simulation frame from subsystem myModel/Subsystem
msgStruct =tl_removesimframe('system','myModel/Subsystem');
ifisempty(msgStruct)
disp('There was no simulation frame that could be removed.);
end
```

Related topics

References

tlEnumDataType

tlEnumDataType

Purpose

Imports a Simulink enumeration data type to the Data Dictionary.

Syntax overview

The following syntaxes are available:

```
[hDDTypedef, hDDEnumTemplate, msgStruct] = tlEnumDataType('CreateDDTypedef', propertyName,
propertyValue, ...)
```

Creates a DD Typedef object and, optionally, a DD EnumTemplate object for a Simulink enumeration data type. The DD EnumTemplate object provides an assignment between the DD Typedef object and the Simulink enumeration data type and can be used by the TargetLink Code Generator.

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`SLEnumType`	Name of the Simulink enumeration data type. (mandatory)
`TypedefGroup`	Specifies the DD TypedefGroup object to which the Simulink enumeration data type is imported. Default: /Pool/Typedefs/Enums

Property	Description
`Verbose`	If 'on', produces verbose output in the MATLAB Command Window. Default: 'off'
`CreateTemplate`	Specifies that a DD EnumTemplate object is created which keeps the assignment between the Simulink enumeration data type and the DD Typedef object. Default: 'on'
`Overwrite`	Specifies if existing DD objects are overwritten or merged. The following values are possible: ▪ 'Merge' - Do not modify existing DD objects. Display an error message if an existing DD object conflicts with the imported Simulink enumeration data type. ▪ 'Overwrite' - Overwrite existing DD objects. Default: 'Merge'

Output parameters The following output parameters are available:

Parameter	Description
`hDDTypedef`	Handle of DD Typedef object to which the Simulink enumeration data type was imported
`hDDEnumTemplate`	Handle of the DD EnumTemplate object that provides the assignment between the Simulink enumeration data type and the DD Typedef object
`msgStruct`	Message struct containing all error and success messages. If not used , the TargetLink Message Browser shows these messages.

Example

```
Simulink.defineIntEnumType('myColor',{'red','green','blue'},[102030],'DefaultValue','blue','Description','My base
colors');

% Imports the 'myColor' Simulink enumeration data type to the myColor DD Typedef object in /Pool/Typedefs/myEnumTypes:
[hDDTypedef, hDDEnumTemplate, msg]=tlEnumDataType('CreateDDTypedef','SLEnumType','myColor','TypedefGroup','myEnumTypes')
```

tl_create_blacklist

tl_create_blacklist

Purpose Due to a limitation of the Merge block, logging and overflow detection during MIL simulation are not possible for blocks that are connected to Merge block input ports. If these blocks are not directly connected to the Merge block, but through virtual blocks, subsystem borders or bus signals, TargetLink might not detect them before logging or overflow detection starts. In this case, the simulation stops with an error.

Syntax overview

The following syntaxes are available:

```
hSrcPorts = tl_create_blacklist('firstStep', model, propertyName, propertyValue, ...)
```

Creates a list of all ports connected to Merge blocks. Since the Simulink model is not compiled for the search, the list might be incomplete.

```
tl_create_blacklist('create', model)
```

Creates a file called <model>_blacklist.txt. This file contains paths of the blocks that cannot be logged because they are connected to Merge block input ports. If the <model>_blacklist.txt file already exists, it is overwritten.

```
tl_create_blacklist('refresh', model)
```

Updates the list of blocks in the automatic section of <model>_blacklist.txt that cannot be logged because they are connected to Merge block input ports. If the <model>_blacklist.txt file does not exist, it is created. The manual section in <model>_blacklist.txt created via the 'add' command is not modified.

```
tl_create_blacklist('add', model, propertyName, propertyValue, ...)
```

Adds a port to manual section in <model>_blacklist.txt

```
tl_create_blacklist('remove', model, propertyName, propertyValue, ...)
```

Removes a port from the manual section in <model>_blacklist.txt

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`block`	Block path/handle
`port`	Index of port
`MergeBlocks`	List of Merge blocks for which source ports are to be searched. If not specified, the function searches the source ports for all Merge blocks contained in the model.

Input parameters

The following input parameters are available:

Parameter	Description
`model`	Name or handle of the model

Output parameters

The following output parameters are available:

Parameter	Description
`hSrcPorts`	List of all source ports connected to Merge blocks

Example

```
% Creates a new blacklist file for the 'pipt1' model:
tl_create_blacklist('create','pipt1')

% Updates the blacklist file for the 'pipt1' model:
tl_create_blacklist('refresh','pipt1')

% Excludes the 'pipt1/picontroller/Subsystem/picontroller/Kp' block from MIL handling
tl_create_blacklist('add','pipt1','block','pipt1/picontroller/Subsystem/picontroller/Kp')

% Includes port 1 of the 'pipt1/picontroller/Subsystem/picontroller/Kp' block in MIL handling
tl_create_blacklist('remove','pipt1','block','pipt1/picontroller/Subsystem/picontroller/Kp','port',1)

% Gets all source ports of all Merge blocks
hSrcPorts =tl_create_blacklist('firstStep','MyModel')

% Gets all source ports of Merge blocks specified in the hMergeList list
hSrcPorts =tl_create_blacklist('firstStep','MyModel','MergeBlocks', hMergeList)
```

Related topics

Basics

Basics on Logging (📖 TargetLink Preparation and Simulation Guide)
Basics on Logging Signals in MIL Simulation Mode (📖 TargetLink Preparation and Simulation Guide)
Overflow Detection (📖 TargetLink Preparation and Simulation Guide)
Verifying the Controller Design (MIL Simulation Mode) (📖 TargetLink Preparation and Simulation Guide)

Upgrading TargetLink Systems

tlUpgrade

tlUpgrade

Purpose

Upgrades block diagrams containing TargetLink blocks to the current version.

Description

This function upgrades TargetLink models created with an earlier TargetLink version to the version that you are currently using. The need for an upgrade can result from the following cases:

- Different TargetLink version
- Different TargetLink and MATLAB versions

All the model elements that changed are replaced by the ones matching the MATLAB and TargetLink versions you are currently using.

Apart from upgrading your models, you can also use this function to check all the model's TargetLink-relevant data for consistency:

- If you set the CheckModel property to 'ReportIssues', this function reports issues with the model's TargetLink-relevant data for you to evaluate.
- If you set the CheckModel property to 'FixIssues', this function reports and fixes issues with the model's TargetLink-relevant data. This usually means that the affected block properties are set to their default values, because TargetLink can check your model only formally.

Syntax

```
[errFlag, msgStruct] = tlUpgrade(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Model to be upgraded Default: current model

Property	Description
`CheckModel`	Performs a complete check of all TargetLink-relevant data and settings. The possible values for this property are: ▪ 'off' - Performs no complete check. ▪ 'ReportIssues' - Reports all the issues found by the complete check. ▪ 'FixIssues' - Fixes all the issues found by the complete check. Default: 'off'
`TLVersion`	Specifies the TargetLink version used to edit the model last time. Usally the version is stored at the model and there is no need to utilize this argument. It is intended for the exceptional case that the version number is missing.

Output parameters

The following output parameters are available:

Parameter	Description
`errFlag`	Indicates whether upgrade was successful or not
`msgStruct`	Structure with messages

Related topics

Basics

Basics on Migrating Between TargetLink Versions (📖 TargetLink New Features and Migration Guide)
Basics on Simulation Modes and Preconditions (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

Autoscaling Variables

tl_autoscaling

Where to go from here

Information in this section

tl_autoscaling

Purpose

Calculates (worst-case) ranges and scaling parameters for output, state and parameter variables of TargetLink blocks.

Syntax overview

The following syntaxes are available:

```
asds = tl_autoscaling('init', propertyName, propertyValue, ...)
```

Initializes autoscaling data structure (ASDS)

```
asds = tl_autoscaling('propagateRanges', propertyName, propertyValue, ...)
```

Propagates all known range limits: i.e., constrained limits and limits derived from valid scaling parameters

```
asds = tl_autoscaling('calculateRanges', propertyName, propertyValue, ...)
```

Calculates worst-case ranges for all blocks within specified scope

```
asds = tl_autoscaling('inheritScaling', propertyName, propertyValue, ...)
```

Inherits scaling parameters of all blocks marked as valid, as far as possible within specified scope

```
asds = tl_autoscaling('calculateScaling', propertyName, propertyValue, ...)
```

Calculates scaling parameters based either on worst-case ranges or ranges determined via simulation

Example

```
% initialize the autoscaling data structure and refreshe the netlist
asds =tl_autoscaling('init','subsystem',<subsystem>);

% calculate worst-case ranges for <subsystem>
asds =tl_autoscaling('calculateRanges','subsystem',<subsystem>);

% autoscale blocks in the subsystem by using ranges of the existing data structure
asds =tl_autoscaling('calculateScaling','subsystem',<subsystem>,'useexistingasds',<asds>);
```

Remarks

You can use the following colors:

- 'none'
- 'Black'
- 'White'
- 'Red'
- 'Green'
- 'Blue'
- 'Cyan'
- 'Magenta'
- 'Yellow'
- 'Gray'
- 'Light Blue'
- 'Orange'
- 'Dark Green'

Related topics

Basics

Transforming Floating-Point Code to Fixed-Point Code (Scaling Variables) (TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (TargetLink Interoperation and Exchange Guide)

References

Gain Block (TargetLink Model Element Reference)
tl_set.. 87

tl_autoscaling('init', propertyName, propertyValue, ...)

Purpose

Initializes autoscaling data structure (ASDS)

Description

This command is part of the tl_autoscaling function.

Syntax

```
asds = tl_autoscaling('init', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'subsystem'`	Name or handle of subsystem the command is to be performed for.
`'useExistingASDS'`	Given autoscaling data structure is used for specified command
`'subsystemOnly'`	All blocks below current subsystem level are ignored. Default: 'on'
`'rangeColor'`	Color of all blocks with given or calculated ranges. Default: 'none'
`'scalingColor'`	Color of all blocks with valid or calculated scaling parameters. Default: 'none'
`'loopColor'`	Color of all blocks that are part of an unsolvable loop. Default: 'none'

Output parameters

The following output parameters are available:

Parameter	Description
`asds`	Autoscaling data structure contains information about netlist, loops, etc.

tl_autoscaling('propagateRanges', propertyName, propertyValue, ...)

Purpose

Propagates all known range limits: i.e., constrained limits and limits derived from valid scaling parameters

Description

This command is part of the tl_autoscaling function.

Syntax

```
asds = tl_autoscaling('propagateRanges', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'subsystem'`	Name or handle of subsystem the command is to be performed for.
`'useExistingASDS'`	Given autoscaling data structure is used for specified command
`'useWorstCaseRanges'`	Autoscaling performed with worst-case ranges, otherwise simulated ranges are used. Default: 'on'
`'subsystemOnly'`	All blocks below current subsystem level are ignored. Default: 'on'
`'rangeColor'`	Color of all blocks with given or calculated ranges. Default: 'none'
`'scalingColor'`	Color of all blocks with valid or calculated scaling parameters. Default: 'none'
`'loopColor'`	Color of all blocks that are part of an unsolvable loop. Default: 'none'

Output parameters

The following output parameters are available:

Parameter	Description
`asds`	Autoscaling data structure contains information about netlist, loops, etc.

tl_autoscaling('calculateRanges', propertyName, propertyValue, ...)

Purpose

Calculates worst-case ranges for all blocks within specified scope

Description

This command is part of the tl_autoscaling function.

Syntax

```
asds = tl_autoscaling('calculateRanges', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'subsystem'`	Name or handle of subsystem the command is to be performed for.
`'useExistingASDS'`	Given autoscaling data structure is used for specified command
`'useWorstCaseRanges'`	Autoscaling performed with worst-case ranges, otherwise simulated ranges are used. Default: 'on'
`'subsystemOnly'`	All blocks below current subsystem level are ignored. Default: 'on'
`'rangeColor'`	Color of all blocks with given or calculated ranges. Default: 'none'
`'scalingColor'`	Color of all blocks with valid or calculated scaling parameters. Default: 'none'
`'loopColor'`	Color of all blocks that are part of an unsolvable loop. Default: 'none'

Output parameters

The following output parameters are available:

Parameter	Description
`asds`	Autoscaling data structure contains information about netlist, loops, etc.

tl_autoscaling('inheritScaling', propertyName, propertyValue, ...)

Purpose

Inherits scaling parameters of all blocks marked as valid, as far as possible within specified scope

Description

This command is part of the tl_autoscaling function.

Syntax

```
asds = tl_autoscaling('inheritScaling', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'loopColor'`	Color of all blocks that are part of an unsolvable loop. Default: 'none'
`'scalingColor'`	Color of all blocks with valid or calculated scaling parameters. Default: 'none'
`'rangeColor'`	Color of all blocks with given or calculated ranges. Default: 'none'
`'subsystemOnly'`	All blocks below current subsystem level are ignored. Default: 'on'
`'useExistingASDS'`	Given autoscaling data structure is used for specified command
`'subsystem'`	Name or handle of subsystem the command is to be performed for.

Output parameters

The following output parameters are available:

Parameter	Description
`asds`	Autoscaling data structure contains information about netlist, loops, etc.

tl_autoscaling('calculateScaling', propertyName, propertyValue, ...)

Purpose

Calculates scaling parameters based either on worst-case ranges or ranges determined via simulation

Description

This command is part of the tl_autoscaling function.

Syntax

```
asds = tl_autoscaling('calculateScaling', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'minParaWidth'`	Data type of parameter is determined so that its word length is as small as possible. Default: 'off'

Property	Description
`'selectDatatypeSign'`	Sign of parameter's data type is optimized: e.g., Uint16 instead of Int16. Default: 'off'
`'scaleStates'`	Parameters of all blocks in the scope are scaled. Default: 'on'
`'scaleParameters'`	States of all blocks in the scope are scaled. Default: 'off'
`'scaleOutputs'`	Outputs of all blocks in the scope are scaled if ranges are available. Default: 'on'
`'defaultWidth'`	Width of default data types. Default: 16 bit
`'loopColor'`	Color of all blocks that are part of an unsolvable loop. Default: 'none'
`'scalingColor'`	Color of all blocks with valid or calculated scaling parameters. Default: 'none'
`'rangeColor'`	Color of all blocks with given or calculated ranges. Default: 'none'
`'subsystemOnly'`	All blocks below current subsystem level are ignored. Default: 'on'
`'useExistingASDS'`	Given autoscaling data structure is used for specified command
`'subsystem'`	Name or handle of subsystem the command is to be performed for.
`'useWorstCaseRanges'`	Autoscaling performed with worst-case ranges, otherwise simulated ranges are used. Default: 'on'
`'scaleDDVariables'`	DD variable specified as block output variable is scaled if './LocalScaling' is specified and ranges are available. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`asds`	Autoscaling data structure contains information about netlist, loops, etc.

Getting Block or Subsystem Parameters

Where to go from here

Information in this section

tl_find

tl_find

Purpose

Searches for TargetLink blocks with specified property values.

Description

This function checks all TargetLink blocks in the specified system to see whether they fulfill the search criteria. A criterion is considered to be satisfied if the property has the given value. Only blocks that satisfy all of the criteria are incorporated into the return value.

Syntax

```
[hBlockList, errorFlag, msg] = tl_find(system, property_1, value_1, ..., property_n, value_n)
```

Input parameters

The following input parameters are available:

Parameter	Description
`system`	Simulink identifier of model or subsystem to be used as starting point of search
`property_1`	First property used as search criterion
`value_1`	Value of first property
`...`	...
`property_n`	n-th property used as search criterion
`value_n`	Value of n-th property

Output parameters

The following output parameters are available:

Parameter	Description
`hBlockList`	Column matrix with handles of found blocks, empty matrix if no blocks were found
`errorFlag`	Flag showing success or failure. The following error flag values are defined: ▪ 0 - No error ▪ 1- 1st input parameter not a Simulink subsystem ▪ 2 - Uneven number of property/property values ▪ 3 - Empty or nonstring property identifier found
`msg`	Error message, empty on success

Example

```
% search pipt1 for all gain blocks whose gain type is Int8
hInt8Gains =tl_find('pipt1','gain.type','Int8');
```

Related topics

Basics

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

tlFindDDReferences

tlFindDDReferences

Purpose

Returns TargetLink blocks, Stateflow objects and DD objects referencing a certain DD object.

Syntax

```
[reference, skipped] = tlFindDDReferences(ddObjectPath, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`System`	Simulink identifier of the model or subsystem to be searched through
`IncludeReferencedModels`	If set to 'on', the search includes referenced models. Default: 'on'
`IncludeDataDictionary`	If set to 'on', the search includes the current Data Dictionary workspace. Default: 'on'

Input parameters

The following input parameters are available:

Parameter	Description
`ddObjectPath`	Path of DD object

Output parameters

The following output parameters are available:

Parameter	Description
`reference`	Structure array of found references, containing the following fields: ▪ object - Path of object ▪ objectKind - Object kind ('slblock' | 'sfobject' | 'ddobject') ▪ objectHandle - Numerical object handle ▪ property - Name of the property

Parameter	Description
`skipped`	Structure array of skipped objects. The search routine records read-only objects in the model and the Data Dictionary, and also excluded objects as specified by the search options IncludeReferencedModels and IncludeDdObjects. The structure consists of the following fields: ▪ object - Path of object ▪ objectKind - objectKind('slblock' | 'slhandle' | 'sfobject' | 'ddobject') ▪ objectHandle

Related topics

HowTos

How to Find Data Dictionary Object References (📖 TargetLink Data Dictionary Basic Concepts Guide)

References

Find References (📖 TargetLink Data Dictionary Manager Reference)
tlMoveDDObject... 88

get_tlsystemID

get_tlsystemID

Purpose

Returns the system ID of the specified TargetLink code generation unit (CGU).

Syntax

```
tlSystemID = get_tlsystemID(system)
```

Input parameters

The following input parameters are available:

Parameter	Description
`system`	Simulink identifier of TargetLink subsystem, subsystem configured for incremental code generation or referenced model

Output parameters

The following output parameters are available:

Parameter	Description
`tlSystemID`	TargetLink system ID

Example

```
% get a system ID via a Simulink handle obtained with get_tlsubsystems
hTLSys =get_tlsubsystems('pipt1', true);
tlSystemID =get_tlsystemID(hTLSys)
```

Remarks

The system ID is a unique identifier for the TargetLink subsystem. You can use it to form unique code identifiers via name macros ($I, $S).

Related topics

References

tl_get_sfobjects

tl_get_sfobjects

Purpose

Compiles a list of Stateflow objects belonging to a given class.

Description

This function looks for Stateflow objects in Simulink system(s) that are relevant for TargetLink.

Syntax

```
[hSFObjectList, objClassList] = tl_get_sfobjects(systemList, objList)
```

Input parameters

The following input parameters are available:

Parameter	Description
`systemList`	Simulink system(s) to be searched through for Stateflow objects
`objList`	List of Stateflow object types or the string 'all' for searching all Stateflow objects. If this parameter has been omitted all Stateflow objects types are searched for.

Output parameters

The following output parameters are available:

Parameter	Description
`hSFObjectList`	Column matrix with handles of found Stateflow objects, empty matrix if no objects were found
`objClassList`	Cell array containing associated Stateflow object types

Example

```
% get all Stateflow objects in model1 and model2
hSFObjList =tl_get_sfobjects({'model1''model2'});

% get all SF Outputs in model or subsystem whose handle is hSys
hSFObjList =tl_get_sfobjects(hSys,'SFOutput');

% get all SF Inputs and SF Outputs in model or block whose handle is h
hSFObjList =tl_get_sfobjects(h,{'SFOutput''SFInput'});

% use second return value to create a Stateflow object statistic
[hSFObj, objTypes]=tl_get_sfobjects(bdroot);
foundTypes =unique(objTypes);
for m =1:numel(foundTypes),
    type = foundTypes{m};
    sfid =sort(hSFObj(strcmp(objTypes, type)));
    csv =sprintf('%d, ', sfid);
fprintf('%3d %s(s) found (Stateflow identifiers: %s)\n',...
numel(sfid), type,csv(1:end-2));
end
```

Remarks

To obtain a list of Stateflow object types use the following command:
`tl_manage_blockset('GetSFTypes')`

Related topics

Basics

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

tl_get_blocks

tl_get_blocks

Purpose

Compiles a list of TargetLink blocks belonging to a given class.

Syntax

```
[hBlockList, blockTypeList] = tl_get_blocks(systemList, objClass)
```

Input parameters

The following input parameters are available:

Parameter	Description
`systemList`	Simulink systems to be searched through for TargetLink blocks
`objClass`	List of TargetLink block types or one of the following classes of block types: ▪ All - All blocks supported by TargetLink ▪ AllInclSubsystems - Like 'All', but with subsystems included ▪ TargetLink - Blocks with TargetLink settings ▪ Simulink - Supported blocks without TargetLink settings ▪ TLSim - TargetLink blocks with simulation functionality ▪ TLUtility - TargetLink utility blocks

Output parameters

The following output parameters are available:

Parameter	Description
`hBlockList`	Column matrix with handles of found blocks, empty matrix if no blocks were found
`blockTypeList`	Cell array containing associated block types

Example

```
% get all TargetLink blocks in model1 and model2
hTLBLocks =tl_get_blocks({'model1','model2'})

% get all TargetLink Inports in model or subsystem whose handle is hSys
hTLBlocks =tl_get_blocks(hSys,'TL_Inport')

% get all TargetLink In- and Outports  in model or subsystem whose handle is hSys
hTLBlocks =tl_get_blocks(hSys,{'TL_Inport','TL_Outport'})

% use second return value to create a statistic over TargetLink simulation blocks
[hTLBlocks, blockTypes]=tl_get_blocks(bdroot,'TLSim');
[foundTypes,~, typeIdxOfBlock]=unique(blockTypes);
for m =1:numel(foundTypes)
    type = foundTypes{m};
    numBlocks =sum(m == typeIdxOfBlock);
fprintf('%3d blocks of type %s\n', numBlocks, type);
end
```

Remarks

To obtain a list of TargetLink block types use this command: tl_manage_blockset('GetTLBlockTypes').

get_tlsubsystems

get_tlsubsystems

Purpose

Returns identifiers of TargetLink subsystems in Simulink system.

Syntax

```
tlSubsystems = get_tlsubsystems(model, bHandle)
```

Input parameters

The following input parameters are available:

Parameter	Description
`model`	Simulink system
`bHandle`	▪ True - Returns subsystems as vector of handles ▪ False/Omitted - Returns cell array of subsystem paths

Output parameters

The following output parameters are available:

Parameter	Description
`tlSubsystems`	Identifiers of TargetLink subsystems

Related topics

Basics

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

tl_get

tl_get

Purpose

Retrieves TargetLink properties of blocks and Stateflow objects.

Syntax

```
[value, errorFlag, msg] = tl_get(hBlock, propertyName)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hBlock`	(List of) Simulink identifier(s) of Simulink block(s)/Stateflow object(s)
`propertyName`	Property whose value to retrieve

Output parameters

The following output parameters are available:

Parameter	Description
`value`	Retrieved property value, empty on error
`errorFlag`	Non-zero number in case of failure, zero on success
`msg`	Contains error message, empty on success

Example

```
% get variable classes of all gain variables in model 'pipt1'
hBlocks =tl_get_blocks('pipt1','TL_Gain');
ifisempty(hBlocks)
    varClassList={};
elseifnumel(hBlocks)==1
    varClassList ={tl_get(hBlocks,'gain.class')};
else
    varClassList =tl_get(hBlocks,'gain.class');
end

% get data type of bus signal with label MyBus.SubBus.element_a
numElem =tl_get(hBusPort,'numoutputs');
sigNameList =arrayfun(@(idx)tl_get(hBusPort,['output(',num2str(idx)').signalname']),1:numElem,...
'UniformOutput', false);
idxOut =num2str(find(strcmp('MyBus.SubBus.element_a', sigNameList));
type =tl_get(hBusPort,['output(', idxOut,').type']);
```

Remarks

1. If you invoke tl_get with only one right-hand argument, the command returns a cell array of strings with all the available property identifiers for the selected block.
2. The imaginary property BlockDataStruct combines all TargetLink properties in a single structure.

Related topics

Basics

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

References

tl_sim.. 134

tl_get_subsystem_info

tl_get_subsystem_info

Purpose

Provides information about TargetLink subsystem.

Description

Provides name and path information for TargetLink subsystem.

Syntax

```
[info, msg] = tl_get_subsystem_info(sys)
```

Input parameters

The following input parameters are available:

Parameter	Description
`sys`	Simulink identifier of the TargetLink subsystem or any block that resides in a TargetLink subsystem. TargetLink simulation frames are considered, which means that it suffices to specify the visible simulation frame subsystem in order to specify the underlying TargetLink subsystem.

Output parameters

The following output parameters are available:

Parameter	Description
`info`	Struct containing name and path information: ▪ tlSubsystemName - Name of the TargetLink subsystem. ▪ tlSubsystemParent - Parent of the TargetLink subsystem. ▪ tlSubsystemPath - Path of the TargetLink subsystem. ▪ milSubsystemPath - Path of the TargetLink subsystem's MIL subsystem. ▪ simMode - Current simulation mode of the TargetLink subsystem. ▪ sfcnName - Name of the S-function which implements the TargetLink subsystem.
`msg`	Error message (empty string on success)

Example

```
[info,msg]=tl_get_subsystem_info(hBlock);
fprintf('The block is in TL subsystem %s\n',info.tlSubsystemName);
```

Related topics

Basics

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

References

tlExtractSubsystem 116

tl_get_userdata

tl_get_userdata

Purpose

Extracts data embedded in the description string.

Description

User-defined data can be embedded in the block description of a TargetLink block as an M-script code snippet. The code snippet must be enclosed by the keyword $TLUSERDATA$ to separate it from the regular description text. This function evaluates the user data script in a separate local workspace, and the resulting variables are returned as fields of a MATLAB struct.

Note that this functionality is designed to augment the Custom Look-up Function feature. Do not use it for other purposes. This feature might be subject to change in future TargetLink versions.

Syntax

```
userDataStruct = tl_get_userdata(hBlock)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hBlock`	Simulink identifier of TargetLink block that has user data

Output parameters

The following output parameters are available:

Parameter	Description
`userDataStruct`	Structure containing all variables created by user data code

Example

```
Suppose a block comment contains
    description text
    $TLUSERDATA$
    inputMode =1;
    myVar ='value';
    $TLUSERDATA$

... then this command
>> userData =tl_get_userdata(hBlock);

... returns
    userData =
        inputMode:1
            myVar:'value'
```

Related topics

Basics

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

tl_manage_blockset

tl_manage_blockset

Purpose

Provides information about TargetLink blocksets.

Syntax overview

The following syntaxes are available:

```
blockName = tl_manage_blockset('BlockType2Name', blockType)
```
Returns name of block representing the specified block type in the blockset library

```
blockType = tl_manage_blockset('Name2BlockType', name)
```
Returns type of block with the specified name

```
blockDef = tl_manage_blockset('GetBlockDef', blockType)
```
Provides information about certain block type (TL, SL, SF, EML)

```
blockDef = tl_manage_blockset('GetTLBlockDef', slBlockType)
```
Provides information about certain block (expects a Simulink block type as its input argument)

```
[blockVars, blockVarKinds] = tl_manage_blockset('GetBlockVariables', blockType)
```
Returns list of variables (kinds) associated with block type

`blockTypeList = tl_manage_blockset('GetValidBlockTypes')`

Block types known to TargetLink Code Generator

`blockTypeList = tl_manage_blockset('Get<group>Types')`

Returns list of block types belonging to <group>.
<group> stands for one of the following:
- TLBlock
- TLSimBlock
- TLUtitlityBlock
- SLBlock
- SF

Input parameters

The following input parameters are available:

Parameter	Description
`name`	Name of block in the blockset library
`blockType`	TargetLink block type
`slBlockType`	Simulink block type

Output parameters

The following output parameters are available:

Parameter	Description
`blockName`	Name of block in blockset library
`blockType`	Block type identifier
`blockTypeList`	List of block types
`blockDef`	Structure with information about certain block type
`blockVars`	Names of substructures describing block variables
`blockVarKinds`	Kind of block variables

Example

```
% list all the variables of the TargetLink FIR Filter block including their types
>>[var, kind]=tl_manage_blockset('GetBlockVariables','TL_FIRFilter')

var =
'coeff''input''output'

kind =
'parameter''input''output'

% get all the TargetLink Utility blocks
utilBlockTypes =tl_manage_blockset('GetTLUtilityBlockTypes');
```

Setting Block or Subsystem Parameters

Where to go from here

Information in this section

set_tlsystemID

set_tlsystemID

Purpose

Sets the system ID of the TargetLink subsystem.

Syntax

```
bSuccess = set_tlsystemID(system, tlSystemID)
```

Input parameters

The following input parameters are available:

Parameter	Description
`system`	Simulink identifier of TargetLink subsystem
`tlSystemID`	TargetLink system ID (string consists of letters, digits, and underscores)

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	Possible values: ▪ 'True' - Success ▪ 'False' - Failure

Example

```
% set the system ID of the picontroller subsystem in the PIPT1 demo model to sys1
bSuccess =set_tlsystemID('pipt1/picontroller/Subsystem/picontroller','sys1')
```

Remarks

The system ID is a unique identifier for the TargetLink subsystem. You can use it to form unique code identifiers via name macros ($I, $S).

Related topics

Basics

Basics on Using Name Macros (TargetLink Customization and Optimization Guide)

References

get_tlsystemID.. 73
TargetLink Main Dialog Block (TargetLink Model Element Reference)
tl_get_subsystem_info.. 79

tl_repair_busdata

tl_repair_busdata

Purpose

Checks and corrects the TargetLink data of bus port blocks.

Description

Checks whether data of bus port blocks is appropriate for current bus structure

Syntax

```
msgStruct = tl_repair_busdata(hBusports, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`ReportIndexChanges`	If 'on', generated report contains all signal changes including index switches. Default: 'on'
`ShowReport`	If 'on', report is opened. Default: 'on'
`PropagateChangesToLibraries`	If 'on', associated libraries are unlocked and block data is updated in library. Default: 'on'

Input parameters

The following input parameters are available:

Parameter	Description
`hBusports`	List of bus port blocks

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	TargetLink message struct

Example

```
hBusports =tl_get_blocks(<model>,{'TL_BusInport','TL_BusOutport'});
tl_repair_busdata(hBusports,...
'ShowReport','on',...
'PropagateChangesToLibraries','on',...
'ReportIndexChanges','on');
```

Remarks

All blocks must be contained in the same model.

tl_set

tl_set

Purpose

Modifies TargetLink properties of blocks and Stateflow objects.

Syntax

```
[errorFlag, msg] = tl_set(hBlock, property_1, value_1, ..., property_n, value_n, propertyName,
propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`AllowLibraryBlockModification`	Autorized tl_set to modify library blocks, i.e. if it is called from a callback script. Default: 'off'

Input parameters

The following input parameters are available:

Parameter	Description
`hBlock`	(List of) Simulink identifier(s) of Simulink block(s)/Stateflow object(s)
`property_1`	1st property whose value should be modified
`value_1`	New property value
`...`	...
`property_n`	n-th property whose value should be modified
`value_n`	New property value

Output parameters

The following output parameters are available:

Parameter	Description
`errorFlag`	Non-zero number in case of failure, zero on success
`msg`	Contains error message, empty on success. If more than one object is processed, a cell array is used to deliver messages.

Example

```
% in model pipt1 set the variable class of all gain parameter variables to CAL
hGain =tl_get_blocks('pipt1','TL_Gain');
tl_set(hGain,'gain.class','CAL');

% transfer TargetLink settings from busport BP_A to busport BP_B
blockDataStruct =tl_get(hBP_A,'BlockDataStruct');
tl_set(hBP_B,'BlockDataStruct', blockDataStruct);

% reset scaling properties of all elements of a bus signal
for m =1:tl_get(hBusPort,'numoutputs')
tl_set(hBusPort,['output(',num2str(m),').lsb'],1,['output(',num2str(m),').offset'],0);
end
```

Remarks

The imaginary BlockDataStruct property combines all TargetLink properties in a single structure. When you set properties via BlockDataStruct, the following particularities apply:

- The structure might be incomplete.
- If the structure contains read-only properties, these properties are ignored.

If you use a bus port block, you have to use indices to specify the property of the bus signal that is to be modified or read: tl_set(<BusPortIdentifier>, output(2).type, 'Float32')

Related topics

Basics

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

References

tl_get.. 78

tlMoveDDObject

tlMoveDDObject

Purpose

Moves or renames a DD object and adapts references to this object.

Syntax

```
errMsg = tlMoveDDObject(oldName, newName, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`system`	Model or subsystem where references should be adapted. Default: []
`adaptOnly`	Adapt references only. This implies that the object was already moved or renamed. Default 'off'
`includeReferencedModels`	Specifies whether referenced models are involved in adaptating references. Default: 'on'
`includeLibraries`	Specifies whether libraries are involved in adaptating references. Default: 'on'

Input parameters

The following input parameters are available:

Parameter	Description
`oldName`	Path or handle of DD object to be renamed
`newName`	New path or name for DD object

Output parameters

The following output parameters are available:

Parameter	Description
`errMsg`	Returns a message structure, empty if no error occurs

Related topics

References

tlFindDDReferences.. 72

tl_get_block_config

tl_get_block_config

Purpose

Provides block configuration options not accessible via TargetLink API or UI.

Description

This function is mainly intended to configure user-written look-up functions, e.g., the data format for the output of the PreLook-Up Index Search block. By default, an empty matrix is returned, i.e. there are no block-specific options. You can modify the block-specific configuration options according to your project's needs.

Syntax

```
configOptions = tl_get_block_config(block)
```

Input parameters

The following input parameters are available:

Parameter	Description
`block`	Simulink identifier of block whose configuration options are to be retrieved

Output parameters

The following output parameters are available:

Parameter	Description
`configOptions`	Blocktype-specific configuration options

tlPromoteProperty

tlPromoteProperty

Purpose

Provides TargetLink properties in TargetLink blocks masks for Simulink promote mechanism.

Syntax overview

The following syntaxes are available:

`msgStruct = tlPromoteProperty('add', hBlock, tlProp, paramName, prompt)`

Adds mask parameter for TargetLink property.

`msgStruct = tlPromoteProperty('remove', hBlock, tlProp)`

Removes mask parameter for TargetLink property.

`msgStruct = tlPromoteProperty('clear', hBlock)`

Clears all mask parameter for TargetLink properties.

Input parameters

The following input parameters are available:

Parameter	Description
`hBlock`	Model element
`tlProp`	TargetLink property (i.e. 'output.type')
`paramName`	Name of new mask parameter (fix for Simulink parameter)
`prompt`	Prompt of new mask parameter

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	TargetLink message structure

Remarks

If called without input and output arguments, the TargetLink Mask Parameters dialog opens, enabling you to prepare TargetLink properties in TargetLink blocks interactively.

Related topics

Basics

Basics on Adding Mask Parameters for TargetLink Block Properties (📖 TargetLink Preparation and Simulation Guide)

Decomposing Models

Where to go from here

Information in this section

tl_compare_fcn_signature

tl_compare_fcn_signature

Purpose

Compares the current interface of the reference model with the reference.

Description

This function compares the current interface of the referenced model with the given reference. The interface of the referenced model is given as a DD Subsystem object created during production code generation for this model. The reference is given as a DD file containing the saved DD Subsystem object created during the reference production code generator run for this model. The interface of the referenced model matches the given reference if the description of the root functions in the specified DD Subsystem object and the specified reference DD files are identical. Any remaining DD objects are not taken into account. If desired, an arbitrary function can be compared (not the root functions only). In this case, the name of the function must be specified explicitly. The results of the comparison are written to the log file <Subsystem>_fcn_compare.log

Syntax

```
tl_compare_fcn_signature(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`ReferenceDDFile`	Path of reference DD file, absolute or relative
`Subsystem`	Name of DD Subsystem object
`FcnName`	Name of function to be compared. Default: all root functions in the specified Subsystem object

Example

```
% Compare the signature of the root functions in Subsystem object TL_SubSysD_MdlRef
% residing in current DD and the referenced DD save in the file TL_SubSysD_MdlRef_reference.dd
tl_compare_fcn_signature('ReferenceDDFile','TL_SubSysD_MdlRef_reference.dd'...
'Subsystem','TL_SubSysD_MdlRef');
```

Remarks

You can create the referenced DD file during the file distribution process by calling the tl_distribute_refmodel_files API function with its CopyRefDDFile property set to 'on'.

tl_refmodel_to_subsystem

tl_refmodel_to_subsystem

Purpose

Converts referenced models into subsystems configured for incremental code generation.

Description

This function replaces the specified Model blocks with subsystems configured for incremental code generation. The contents of the subsystems configured for incremental code generation are equal to the contents of the models referenced by the Model block. The name of the created subsystem configured for incremental code generation is set as follows:

- Equal to the name of the model if the referenced model is not reusable
- Equal to the name of the Model block if the referenced model is reusable

The referenced models remain unchanged. To identify the resulting subsystems as converted from a referenced model, their blocks are marked by double lines that make them look like the Model block.

Syntax

```
bSuccess = tl_refmodel_to_subsystem(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`RefModelBlock`	Simulink path(s) of Model block(s) to be replaced by incremental subsystem(s)

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	1 - Successfully converted referenced model(s) into subsystem(s). 0 - Error

Example

```
% Convert the Model block myAtomicSub1 in the TL subsystem
% pipt1_ref/picontroller/Subsystem/picontroller into a subsystem configured for incremental code
% generation of the same name.
tl_refmodel_to_subsystem('RefModelBlock',...
'pipt1_ref/picontroller/Subsystem/picontroller/myAtomicSub1');

% Convert the Model blocks r_AirflowCalculation and r_SensorCorrection in the TL subsystem
% mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller into subsystems configured for incremental code
% generation
tl_refmodel_to_subsystem('RefModelBlock',...
{'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_AirflowCalculation',...
'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_SensorCorrection'});
```

Remarks

When a referenced model is converted into a subsystem, the variables in its model workspace are not considered: i.e., they are not imported into the parent model's workspace.

Related topics

HowTos

How to Disable Model References (TargetLink Customization and Optimization Guide)

References

tl_subsystem_to_refmodel

tl_subsystem_to_refmodel

Purpose

Converts subsystems configured for incremental code generation into referenced models.

Description

This function creates new models, copies the contents of the specified atomic subsystems configured for incremental code generation into the respective models, replaces the subsystems with Model blocks and creates references from the Model blocks to the corresponding models.

This function checks whether a Function block with the following properties exists on the top-most level of the subsystem:

- The $N, $M and $B name macros are not used in the names of the Step, Init, Start and Restart functions and the module name.
- The 'Make function reusable' option is not set.
- The subsystem ID is set.
- The 'Scaling invariant' option is not set.

In function-call triggered subsystems it checks whether the 'States when enabling' option of the TriggerPort is set to 'Reset' or 'Held'.

In addition, TargetLink prepares the model, i.e., necessary callbacks and model parameters are set.

Syntax

```
bSuccess = tl_subsystem_to_refmodel(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Subsystems`	List of subsystems to be converted
`RefModelNames`	List of names for referenced models created from specified subsystems. If empty, subsystem names are used.

Property	Description
`UseExistingModel`	If 'off', an existing model is replaced with the contents of the subsystem. If 'on', an already existing referenced model will be used. If not specified and batch mode is not set, you are asked for your preferences. Must be set in batch mode, otherwise an error is displayed.
`RefModelsSimulationMode`	Simulink simulation mode of the created referenced models. Possible values are: ▪ Normal - The model is executed interpretively, as if it were an atomic subsystem implemented directly within the parent model. ▪ Accelerator - An S-function is created for the model. Then, the model is executed by running the S-function. Default: Same as in original referenced model, if applicable. Otherwise Normal.

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	1 - Successfully converted subsystem(s) into referenced model(s). 0 - Error

Example

```
% Convert the subsystem 'pipt1/picontroller/Subsystem/picontroller/myIncrSubSys' to referenced model named Ref_Model1.
% If a model Ref_Model1 already exists, replace its contents with the contents of the subsystem myIncrSubSys
tl_subsystem_to_refmodel('Subsystems','pipt1/picontroller/Subsystem/picontroller/myIncrSubSys',...
'RefModelNames','Ref_Model1',...
'UseExistingModel',0)


% Convert the subsystems 'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_SensorCorrection' and
% 'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_AirflowCalculation' to models named
r_SensorCorrection
% and r_AirflowCalculation
tl_subsystem_to_refmodel('Subsystems',...
{'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_SensorCorrection',...
'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_AirflowCalculation'});
```

Remarks

Apart from the properties that are set explicitly for the system to be converted, the properties of the parent model are applied to the newly created ones.

Related topics

HowTos

How to Restore Model References (📖 TargetLink Customization and Optimization Guide)

References

tl_refmodel_to_subsystem.. 93

Property Manager

tlPropman

tlPropman

Purpose

Command-line interface to the TargetLink Property Manager.

Syntax overview

The following syntaxes are available:

`tlPropman('Start')`

Starts the Property Manager.

`tlPropman('Load', Model)`

Loads a model into the Property Manager.

`tlPropman('Select', ModelElement)`

Selects a model element in the Property Manager's navigation tree.

`tlPropman('Unload', Model)`

Unloads a model from the Property Manager.

`tlPropman('Exit')`

Exits the Property Manager.

`tlPropman('ImportViewSets', FileName, OverwriteBehavior)`

Imports view sets.

Input parameters

The following input parameters are available:

Parameter	Description
`Model`	The Simulink model.
`Subsystem`	System/Subsystem to be selected.
`ModelElement`	Model element to be selected.
`FileName`	Name of the file to import.
`OverwriteBehavior`	Behavior of the import if a view set already exists.

Generating and Compiling Code

Where to go from here

Information in this section

tl_build_customcode_sfcn

tl_build_customcode_sfcn

Purpose

Generates and compiles the custom code S-function for the Custom Code block.

Description

This function generates and compiles the S-function for the specified TargetLink Custom Code block. In addition, the TLC file needed to run the S-function with the Simulink Coder (formerly Real-Time Workshop) is generated.

Custom code S-functions are only needed for Type I Custom Code blocks.

The S-function is compiled and linked with the current MEX compiler.

Syntax

```
[bSuccess, sfcnName] = tl_build_customcode_sfcn(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Block`	TargetLink Custom Code block to generate S-function for. Default: []
`CompileOnly`	Compiles S-function (no generation). Default: 'off'
`GenerateOnly`	Generates S-function (no compilation). Default: 'off'
`GenerateTLCFile`	Generates associated TLC file. Default: 'on'
`BlockData`	Block's TargetLink data struct (only used if block property is undefined). Default: []

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	True on success, false on error
`sfcnName`	Name of S-function (derived from block's custom code file)

Related topics

Basics

Basics on Defining the Interface of Custom Code Blocks (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

References

Custom Code Block (📖 TargetLink Model Element Reference)
tl_build_host.......... 101
tl_build_standalone.......... 118
tl_build_target.......... 103

tl_build_host

tl_build_host

Purpose

Generates production code and builds a SIL simulation application for specified TargetLink subsystems.

Description

This function carries out all the steps required to perform a SIL simulation for specified TargetLink subsystem(s):

- Generating production code
- Compiling generated production code and linking it to the simulation application
- Compiling and linking a simulation S-function for each TargetLink subsystem
- Setting each of the TargetLink subsystems to the SIL simulation mode

All the specified TargetLink subsystems must reside in one root model.

Syntax

```
tl_build_host(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of root model. Default: current model
`TLSubsystems`	Name(s) of TargetLink subsystem(s). Default: all TargetLink subsystems in root model
`DDCodeGenerationUnits`	List of DD CodeGenerationUnit objects or DD CodeGenerationUnitGroup objects that production code is to be generated for, specified as list of DD paths or DD handles. Specified DD objects must exist in DD0.
`DDModules`	List of DD Module objects or DD ModuleGroup objects that production code is to be generated for, specified as list of DD paths or DD handles. Specified DD objects must exist in DD0.
`IncludeSubItems`	Generates code for nested systems: i.e., subsystems configured for incremental code generation and referenced models. Default: 'off'

Property	Description
`AllCodeGenerationUnits`	Generates code for all TargetLink subsystems, including nested systems (see above). Includes DD CodeGenerationUnit objects if the following applies: ▪ They are defined in DD0 ▪ They reference one or more DD Module objects whose DD CodeGenerationBasis property is set to 'DDBased' and whose DD ExcludeFromCodegeneration property is set to 'off' If this option is set to 'on', the specification of the code generation units via the options TLSubsystems, DDCodeGenerationUnits and DDModules is ignored. Default: 'off'

Example

```
% Build SIL simulation application for all TargetLink
% subsystems in the current model
tl_build_host
% Carry out an error check afterwards:
if ds_error_check
disp('There was an error while executing TL_BUILD_HOST');
end

% Build SIL simulation for specific TargetLink subsystems,
% ign_inj_control and throttle_control of the model ECUmdl
tl_build_host(...
'Model','ECUmdl',...
'TlSubsystems',{'ign_inj_control','throttle_control'});
% Carry out an error check afterwards:
if ds_error_check
disp('There was an error while executing TL_BUILD_HOST');
end
```

Related topics

Basics

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

HowTos

How to Start the Build Process for All Subsystems (📖 TargetLink Preparation and Simulation Guide)
How to Start the Build Process for Selected Subsystems (📖 TargetLink Preparation and Simulation Guide)

References

Code Generation Hook Scripts (📖 TargetLink File Reference)

tl_build_target

tl_build_target

Purpose

Generates production code and builds a PIL simulation application for specified TargetLink subsystems.

Description

This function carries out all the steps required to perform a PIL simulation for specified TargetLink subsystem(s):

- Generating production code
- Compiling the generated production code and linking it to the simulation application
- Compiling and linking a simulation S-function for each TargetLink subsystem
- Downloading the production code application to the evaluation board selected in the Main Dialog
- Setting each of the TargetLink subsystems to the PIL simulation mode

All the specified TargetLink subsystems must reside in one root model.

Syntax

```
tl_build_target(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of root model. Default: current model
`TlSubsystems`	Names of TargetLink subsystems. Default: all TargetLink subsystems in root model
`DDCodeGenerationUnits`	List of DD CodeGenerationUnit objects or DD CodeGenerationUnitGroup objects that production code is to be generated for, specified as list of the DD paths or DD handles. Specified DD objects must exist in DD0.
`DDModules`	List of DD Module objects or DD ModuleGroup objects that production code is to be generated for, specified as list of the DD paths or DD handles. Specified DD objects must exist in DD0.
`IncludeSubItems`	Generates code for nested systems: i.e., subsystems configured for incremental code generation and referenced models
`AllCodeGenerationUnits`	Generates code for all TargetLink subsystems, including nested systems (see above). Includes DD CodeGenerationUnit objects if the following applies: ▪ They are defined in DD0 ▪ They reference one or more DD Module objects whose DD CodeGenerationBasis property is set to 'DDBased' and whose DD ExcludeFromCodegeneration property is set to 'off' If this option is set to 'on', the specification of the code generation units via the options TLSubsystems, DDCodeGenerationUnits and DDModules is ignored.

Example

```
% Build PIL simulation application for all TargetLink
% subsystems in the current model and download it to the EVB
tl_build_target
% Carry out an error check afterwards:
if ds_error_check
disp('There was an error while executing TL_BUILD_TARGET');
end

% Build PIL simulation for specific TargetLink subsystems,
% ign_inj_control and throttle_control of the model ECUmdl
tl_build_target(...
'Model','ECUmdl',...
'TlSubsystems',{'ign_inj_control','throttle_control'});
% Carry out an error check afterwards:
if ds_error_check
disp('There was an error while executing TL_BUILD_TARGET');
end
```

Related topics

Basics

Basics on the Build Process (📖 TargetLink Preparation and Simulation Guide)
Basics on the Download Process (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

HowTos

How to Start the Build Process for All Subsystems (📖 TargetLink Preparation and Simulation Guide)
How to Start the Build Process for Selected Subsystems (📖 TargetLink Preparation and Simulation Guide)

References

Code Generation Hook Scripts (📖 TargetLink File Reference)

tl_codesize

tl_codesize

Purpose

Evaluates the RAM/ROM consumption of the generated code.

Syntax

```
tl_codesize(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`model`	Name of the model whose code to evaluate. Default value: current model

Property	Description
`simconfig`	Name of target simulation configuration for an evaluation board. Default: simulation configuration of the current model.
`showcodesizefile`	Shows code size information file in current editor. Default: 'on'

Example

```
% check the code size of the current model
tl_codesize;

% check the code size of the model ECUmdl for an SH2 EVB with Renesas compiler  Ver. 5.1
tl_codesize('model','ECUmdl','SimConfig','SH2eEVB/Hit51');

% evaluate code size for model 'poscontrol'; don't open editor
csiFile =tl_codesize('Model','poscontrol','ShowCodesizeFile','off');
```

Related topics

Basics

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

HowTos

How to Determine the Code Size (📖 TargetLink Preparation and Simulation Guide)

References

TargetLink Main Dialog Block (📖 TargetLink Model Element Reference)

tl_compile_host

tl_compile_host

Purpose

Builds an S-function and/or simulation application for SIL simulation.

Description

This function invokes the MEX compiler to compile:

- The simulation application for SIL simulation

- The simulation S-function(s) for SIL/PIL simulation
- The standalone S-function(s)

Depending on the simulation mode, it generates the following compilations:
- SIL simulation mode (TL_CODE_HOST) - S-function(s) for SIL simulation and the SIL simulation application
- PIL simulation mode (TL_CODE_TARGET) - S-function(s) for PIL simulation
- Standalone execution mode (TL_CODE_SFCN) - Standalone S-function(s)

The function retrieves the compiler and target options from the model. They can be adjusted with the tl_set() command before invoking this function.

Syntax

```
tl_compile_host(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of Simulink model containing TargetLink subsystems to compile and link S-function(s) and/or simulation application for. Default: current model
`TlSubsystems`	Names of TargetLink subsystems to compile and link S-functions and/or simulation application for. Default: all TargetLink subsystems in model
`DDCodeGenerationUnits`	List of DD CodeGenerationUnit objects or DD CodeGenerationUnitGroup objects specified as list of DD paths/DD handles. The specified DD objects must exist in DD0.
`SimMode`	Simulation mode: ▪ 'TL_CODE_HOST' - Compiles the simulation application for SIL simulation and, optionally (see CompileSFcn), SIL simulation S-Function(s) ▪ 'TL_CODE_TARGET' - Compiles PIL simulation S-function(s) ▪ 'TL_CODE_SFCN' - Compiles standalone S-function(s) Default: 'TL_CODE_HOST'
`CompileSFcn`	If 'on' and SimMode = 'TL_CODE_HOST', also compiles SIL simulation S-function(s). Default: 'on'
`GenerateGlobalSymbols`	If 'on', creates, compiles, and links a C-module tlsim_<application>_globals.c. Contains definitions of symbols needed for simulation (such as interface variables) that are specified as extern global and therefore not defined in generated code. Set to 'off' if these definitions are made in a user module linked to simulation application to avoid double symbol definitions. Default: 'on'

Property	Description
`RebuildAll`	If 'on', rebuilds all files, also existing OBJ files. Default: 'on'

Example

```
% compile production code simulation application and the S-function(s)
% for current model
tl_compile_host

% compile production code simulation S-functions
% for two TargetLink subsystems in 'ECUmdl'
tl_compile_host(...
'Model','ECUmdl',...
'TlSubsystems',{'ign_inj_control','throttle_control'},...
'SimMode','TL_CODE_TARGET');
% don't forget error checking!
if ds_error_check
disp('There was an error in TL_COMPILE_HOST');
end
```

Related topics

Basics

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

HowTos

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

References

TargetLink Main Dialog Block (📖 TargetLink Model Element Reference)
tl_compile_target.. 109

tl_compile_target

tl_compile_target

Purpose

Builds the simulation application for the target EVB.

Description

This function invokes the target compiler to build the PIL simulation application for the target EVB (evaluation board). It retrieves all compiler and target options from the model. They can be adjusted with the tl_set() command before invoking this function.

Syntax

```
tl_compile_target(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of Simulink model that contains TargetLink subsystems to compile and link simulation application for. Default: current model
`TlSubsystems`	Names of TargetLink subsystems to compile and link a simulation application for. Default: all TargetLink subsystems in model
`DDCodeGenerationUnits`	List of DD CodeGenerationUnit objects or DD CodeGenerationUnitGroup objects, specified as list of DD paths/DD handles. The specified DD objects must exist in DD0.
`RebuildAll`	If 'on', rebuilds all files, also existing OBJ files. Default: 'on'
`GenerateGlobalSymbols`	If 'on', creates, compiles, and links a C-module tlsim_<application>_globals.c. Contains definitions of symbols needed for simulation (such as interface variables) that are specified as extern global and therefore not defined in generated code. Set to 'off' if these definitions are made in a user module linked to simulation application to avoid double symbol definitions. Default: 'on'

Example

```
% compile PIL simulation application for current model
tl_compile_target

% compile PIL simulation application for all TargetLink subsystems in
% model 'ECUmdl'
tl_compile_target('Model','ECUmdl');
% error checking
if ds_error_check
disp('There was an error in TL_COMPILE_TARGET');
end

% compile PIL simulation application for TargetLink subsystem 'ign_inj_control' in
% the model 'ECUMdl'
tl_compile_target('Model','ECUmdl',...
'TLSubsystems','ign_inj_control');
% error checking
if ds_error_check
disp('There was an error in TL_COMPILE_TARGET');
end
```

Related topics

Basics

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

HowTos

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

References

Build Hook Scripts (📖 TargetLink File Reference)
TargetLink Main Dialog Block (📖 TargetLink Model Element Reference)
tl_compile_host........ 106
tl_generate_code........ 111

tl_generate_code

tl_generate_code

Purpose

Calls the Code Generator for the specified code generation units (CGUs).

Description

This function calls the Code Generator for the specified CGUs. The specified model CGUs (i.e., TargetLink subsystems and incremental subsystems) must reside in one Simulink model. The specified Data Dictionary CGUs must reside in the DD0 workspace of the current Data Dictionary.

If desired, frame files needed for building a SIL/PIL simulation are generated in addition to the production code files.

Syntax

```
tl_generate_code(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of Simulink root model containing the model CGUs that production code is to be generated for. Default: current model
`TlSubsystems`	Names of TargetLink subsystems that production code is to be generated for. Default: all TargetLink subsystems in the root model
`IncrSubsystems`	Simulink paths or full model element paths of subsystems configured for the incremental code generation that production code is to be generated for. Full model element path must be specified for an icremental subsystem if one of the following conditions apply: 1) It resides in a model with ModelArguments 2) For it a reused function is to be generated

Property	Description
`ReferencedModels`	Names or full model element paths of referenced models that production code is to be generated for. These models must meet the following conditions: ▪ They do not contain a TargetLink subsystem ▪ They do not contain the Main Dialog Block ▪ They are configured for incremental code generation These models must not be identical with the root model specified via the 'Model' property. The Code Generator options are taken from the root model. The code for the specified referenced models is generated in the same order as specified in the ReferencedModels property. The full model element path must be specified for a model if one of the following conditions apply: 1) It contains ModelArguments 2) For it a reused function is to be generated
`DDCodeGenerationUnits`	List of DD CodeGenerationUnit objects or DD CodeGenerationUnitGroup objects that production code is to be generated for, specified either as list of their DD paths or list of their DD handles. Specified DD objects must exist in DD0.
`DDModules`	List of DD Module objects or DD ModuleGroup objects that production code is to be generated for, specified either as list of their DD paths or list of their DD handles. Specified DD objects must exist in DD0 and their CodeGenerationBasis property must be DDBased.
`IncludeSubItems`	Generates code for specified model CGU's nested systems: i.e., subsystems configured for incremental code generation and for referenced models. If activated, the nested systems' hierarchy is taken into account. Default: 'off'
`AllCodeGenerationUnits`	Generates code for all TargetLink subsystems, including nested systems (see above) and all DD CodeGenerationUnit objects in DD0 that reference at least one DDBased Module not excluded from code generation. If root model is not specified and no model is open, code is generated only for DD CodeGenerationUnit objects. If set to 'on', specification of code generation unit via the options TLSubsystems, FcnSubsystem, ReferencedModels, DDCodeGenerationUnits and DDModules is ignored. Default: 'off'
`SimMode`	Simulation mode: ▪ 'none' - Only production code will be generated ▪ 'TL_CODE_HOST' - Frame files for the SIL simulation are also generated ▪ 'TL_CODE_TARGET' - Frame files for the PIL simulation are also generated ▪ 'TL_CODE_SFCN' - Stand-alone S-function is also generated Default: 'none'
`CheckReferencedModels`	Checks whether models specified by ReferencedModels property are referenced from root model via Model Reference blocks (directly or indirectly, by means of Model Referenced blocks of referenced models). Default: 'off'

Property	Description
`CodeCoverageLevel`	Possible values are 0, 1 and 2: ▪ 0 - Code is NOT instrumented for code coverage tests ▪ 1 - Code is instrumented for statement coverage tests ▪ 2 - Code is instrumented for decision coverage tests Default: 0
`Validate`	Performs a level 4 validation of specified list of DD objects. If list is empty, DD objects are not validated. Default: {'/config', '/Pool', /<Application>}
`CheckCgUnitFullInstancePathSpecification`	Specifies if all parts of the full model element path of the referenced model or incremental subsystem should be checked. By default only the first and last parts are checked. Both must denote a valid Simulink identifier of a subsystem or model. The first part must additionally starts in the root model. Default: off

Example

```
% Generate code for all TargetLink subsystems in the current model.
tl_generate_code;
if ds_error_check,
return;
end

% Generate code for all CGUs, including the incremental systems,
% in the current model.
tl_generate_code('IncludeSubItems','on');
if ds_error_check,
return;
end

% Generate production code and SIL simulation frame
% for two TargetLink subsystems in the ECUmdl model.
tl_generate_code(...
'Model','ECUmdl',...
'TlSubsystems',{'ign_inj_control','throttle_control'},...
'SimMode','TL_CODE_HOST');
if ds_error_check
return;
end

% Generate production code for a subsystem configured
% for incremental code generation.
tl_generate_code(...
'FcnSubsystems','ECUmdl/Subsystem/throttle_control/Subsystem/incrSubsystem');
if ds_error_check,
return;
end

% Validate only /Pool/Variables and /Pool/Typedefs before code generation.
tl_generate_code(...
'Validate',{'/Pool/Variables','/Pool/Typedefs'});
if ds_error_check,
return;
end
```

Related topics

Basics

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

HowTos

How to Generate Production Code for Selected TargetLink Subsystems (📖 TargetLink Preparation and Simulation Guide)

References

Code Generation Hook Scripts (📖 TargetLink File Reference)
CodeCoverageLevel (📖 TargetLink Model Element Reference)
TargetLink Main Dialog Block (📖 TargetLink Model Element Reference)

tl_get_checksum

tl_get_checksum

Purpose

Calculates the checksum of a Simulink model file or an arbitrary ASCII file.

Syntax

```
[checksum, errorFlag, msg] = tl_get_checksum(objectKind, fileName)
```

Input parameters

The following input parameters are available:

Parameter	Description
`objectKind`	Kind of object whose checksum should be calculated: ▪ 'model' - Calculates the checksum of a Simulink model file ▪ 'txt' - Calculates the checksum of an arbitrary ASCII file
`fileName`	Name of file whose checksum should be calculated

Output parameters

The following output parameters are available:

Parameter	Description
`checksum`	Checksum of the selected file
`errorFlag`	Flag indicating success or failure: ▪ 0 - No error ▪ 1 - Invalid object kind ▪ 2 - Filename is not a string or an empty string ▪ 3 - File is not accessible ▪ 4 - Error executing tl_get_cecksum.exe
`msg`	Error or warning message, empty string on success

Example

```
% calculate the checksum of the tl_pre_codegen_hook script
chkSum =tl_get_checksum('txt','tl_pre_codegen_hook.m')

% calculate the checksum of the pipt1 demo model
chkSum =tl_get_checksum('Model','pipt1.mdl')
```

Related topics

Basics

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

tlIsCodeGenerationInProgress

tlIsCodeGenerationInProgress

Purpose

Determines if the code generation process is active.

Syntax

```
bCgInProgress = tlIsCodeGenerationInProgress()
```

Output parameters

The following output parameters are available:

Parameter	Description
`bCgInProgress`	Indicates whether code generation is in progress: ▪ `'true'` – Code generation is in progress. ▪ `'false'` – No code generation is in progress.

tlExtractSubsystem

tlExtractSubsystem

Purpose

Generates a new independent TargetLink subsystem from a TargetLink subsystem part.

Description

This function copies one of the following Simulink systems/blocks into a destination system:

- Simulink subsystem
- Simulink Model block
- Stateflow Chart block
- Referenced model

The system/block is embedded into a TargetLink simulation frame to make it ready for simulation and code generation. This lets you extract one part of a large model in order to perform tests with smaller units.

Syntax

```
tlExtractSubsystem(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Source`	Simulink identifier of subsystem, Stateflow chart, model reference block or referenced model
`Destination`	Identifier of destination system (model or subsystem) into which the generated TargetLink subsystem is placed
`Name`	Name of the TargetLink subsystem generated from the extracted system
`Overwrite`	Overwrites existing block with new generated TargetLink subsystem
`AddTestFrame`	Creates subsystems for signal stimuli generation and signal recording and connects them to the generated TargetLink subsystem
`GetLabels`	Names outports of the signal stimuli generator according to propagated signal labels of source model
`AddSimFrame`	Adds simulation frame to generated TargetLink subsystem

Preparing and Performing Simulations

Where to go from here

Information in this section

tl_build_standalone

tl_build_standalone

Purpose

Generates a stand-alone S-function for specified TargetLink subsystems

Description

This function generates code for specified TargetLink subsystems and builds a stand-alone S-function for them. It prepares the destination model for stand-alone simulations with the generated production code. Stand-alone means that the application can be run in a non-TargetLink environment, such as RCP systems.

On success, this function performs the following steps:

- Pastes a subsystem that contains the stand-alone S-functions which run the production code into the destination model
- Removes existing TargetLink Main Dialog block from destination model

- Replaces TargetLink Sink blocks by Simulink Scope blocks
- Removes all TargetLink-related callbacks

This results in a model which runs TargetLink production code but does not need the TargetLink environment.

This process cannot be reverted.

Syntax

```
tl_build_standalone(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of source model that contains TargetLink subsystems to generate stand-alone S-function for. Default: current model
`TlSubsystems`	Names of TargetLink subsystems. Default: all TargetLink subsystems in the model
`DestModel`	Name of destination model where subsystems to be replaced reside. Default: source model
`DestBlocks`	Simulink paths of blocks to be replaced by stand-alone S-function subsystems
`IncludeSubItems`	If 'on', generates code for nested systems (subsystems configured for incremental code generation and referenced models). Default: 'off'
`GenerateGlobalSymbols`	If 'on', creates, compiles, and links a C-module tlstandalone_<tlSubsystem>_globals.c. Contains definitions of symbols needed for simulation (such as interface variables) that are specified as extern global and therefore not defined in generated code. Set to 'off' if these definitions are made in a user module which is to be compiled and linked together with generated code to avoid double symbol definitions. Default: 'on'
`SkipCodeGeneration`	If 'on', skips the production code generation step. The user have to ensure in this case that a required production code was generated and that its description (DD Subsystem object, DD Application object) exists in the currently open DataDictionary. Default: 'off'

Example

```
% Build stand-alone S-function(s) for all TargetLink
% subsystems in the current model, replace the TargetLink subsystems
% with stand-alone S-functions, and remove all TargetLink
% dependencies
tl_build_standalone
% don't forget error checking!
if ds_error_check
disp('There was an error in TL_BUILD_STANDALONE');
end


% Apply this to two subsystems in model 'ECUmdl', but paste the
% stand-alone S-functions into 'ECUmdl_RCP'. 'ECUmdl' will remain
% untouched.
tl_build_standalone(...
'Model','ECUmdl',...
'TlSubsystems',{'ign_inj_control','throttle_control'},...
'DestModel','ECUmdl_RCP');
% don't forget error checking!
if ds_error_check
disp('There was an error in TL_BUILD_STANDALONE');
end
```

Related topics

Basics

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

HowTos

How to Generate Production Code for Selected TargetLink Subsystems (📖 TargetLink Preparation and Simulation Guide)

tlCodeCoverage

Where to go from here

Information in this section

tlCodeCoverage

Purpose

Configures the code coverage analysis of generated production code and generates the code coverage report.

Syntax overview

The following syntaxes are available:

```
tlCodeCoverage('Set', system, propertyName, propertyValue, ...)
```

Configurates code coverage analysis

```
ccConfigValue = tlCodeCoverage('Get', system, propertyName)
```

Returns current value of specified code coverage configuration property

```
ccConfigStruct = tlCodeCoverage('Get', system)
```

Returns current code coverage configuration

```
tlCodeCoverage('GenerateReport', model, propertyName, propertyValue, ...)
```

Generates report of code coverage analysis with selected code coverage tool

```
covTools = tlCodeCoverage('GetCovTools')
```

Gets supported code coverage tools

```
tlCodeCoverage('MoveCTCFiles', propertyName, propertyValue, ...)
```

Moves CTC files (*.sym and *.dat) created during build process and simulation to the specified folder. These files can later be used to create a combined report of code coverage analysis for multiple builds and simulations. By default, each time an application is (re)built, the existing CTC files are moved from the TargetLink build folder into the .\CTCRESULTS\TLBuild_<BuildTimeStamp> folder.
See also: GenerateCombinedReport

```
tlCodeCoverage('GenerateCombinedReport', propertyName, propertyValue, ...)
```

Generates combined report from all CTC files (*.sym and *.dat) residing in and below the specified folders.
See also: MoveCTCFiles

Example

```
% Set the CTC code coverage for model poscontrol and all incremental subsystems
tlCodeCoverage('Set','poscontrol','Tool','CTC','MethodByName','MulticonditionCoverage','ApplyForAll',true);

% Build the SIL application and start the SIL simulation
tl_build_host('Model','poscontrol');
sim('poscontrol');

% Generate code coverage report and open it in your HTML browser
tlCodeCoverage('GenerateReport','poscontrol','Tool','CTC','OutputDirectory','.\MyCTCReport','Show',true);

% Turn off the code coverage analysis
tlCodeCoverage('Set','poscontrol','Tool','CTC','MethodByName','NoCodeCoverage','ApplyForAll',true);
```

Related topics

HowTos

How to Measure Code Coverage via Third-Party Tools (Testwell CTC) (TargetLink Preparation and Simulation Guide)

tlCodeCoverage('Set', system, propertyName, propertyValue, ...)

Purpose

Configurates code coverage analysis

Description

This command is part of the tlCodeCoverage function.

Syntax

```
tlCodeCoverage('Set', system, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Tool'`	Name of supported code coverage tool. The following tools are supported: ▪ TL - TargetLink built-in ▪ CTC - CTC Code Coverage Analyzer for C/C++ Note: Third-party tools are not installed with TargetLink installation and must be installed separately.
`'MethodByLevel'`	The code coverage method, specified by level. Possible values depend on the tool used For TargetLink's code coverage tool these levels are possible: ▪ 0 - No code coverage ▪ 1 - Statement coverage ▪ 2 - Decision coverage (default) For CTC's code coverage tool these levels are possible: ▪ 0 - No code coverage ▪ 1 - Function coverage ▪ 2 - Decision coverage ▪ 3 - Multicondition coverage (default) Equivalent to the 'MethodByName' property.
`'MethodByName'`	The code coverage method, specified by name. Possible values depend on the tool used For TargetLink's code coverage tool these names are possible: ▪ NoCodeCoverage ▪ StatementCoverage ▪ DecisionCoverage (default) For CTC's code coverage tool these names are possible: ▪ NoCodeCoverage ▪ FunctionCoverage ▪ DecisionCoverage ▪ MulticonditionCoverage (default) Equivalent to the 'MethodByLevel' property.
`'ApplyForAll'`	▪ true - Applies the code coverage configuration specified at the root model to nested CGUs ▪ false - Does not apply the code coverage configuration specified at the root model to nested CGUs Only for the CTC Code Coverage Analyzer tool Default: true

Input parameters

The following input parameters are available:

Parameter	Description
`system`	Simulink identifier of model-based CGU to analyze code coverage for

tlCodeCoverage('Get', system, propertyName)

Purpose Returns current value of specified code coverage configuration property

Description This command is part of the tlCodeCoverage function.

Syntax

```
ccConfigValue = tlCodeCoverage('Get', system, propertyName)
```

Input parameters The following input parameters are available:

Parameter	Description
`system`	Simulink identifier of model-based CGU to analyze code coverage for
`propertyName`	Name of property whose value is to be returned

Output parameters The following output parameters are available:

Parameter	Description
`ccConfigValue`	Value of specified code coverage configuration property

tlCodeCoverage('Get', system)

Purpose Returns current code coverage configuration

Description This command is part of the tlCodeCoverage function.

Syntax

```
ccConfigStruct = tlCodeCoverage('Get', system)
```

Input parameters

The following input parameters are available:

Parameter	Description
`system`	Simulink identifier of model-based CGU to analyze code coverage for

Output parameters

The following output parameters are available:

Parameter	Description
`ccConfigStruct`	Structure with information about the specified code coverage configuration. It contains the following fields: ▪ tool - Name of selected code coverage tool ▪ methodByLevel - Code coverage method as a number ▪ methodByName - Code coverage method as a name ▪ applyForAll - true, if the code coverage specification set at the root model applies for all code generation units in the model

tlCodeCoverage('GenerateReport', model, propertyName, propertyValue, ...)

Purpose

Generates report of code coverage analysis with selected code coverage tool

Description

This command is part of the tlCodeCoverage function.

Syntax

```
tlCodeCoverage('GenerateReport', model, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'SimMode'`	Simulation mode to generate code coverage report for. Possible values: ▪ SIL - Code coverage report is generated for SIL simulation ▪ PIL - Code coverage report is generated for PIL simulation Default: SIL
`'Simulations'`	Names of simulations that code coverage report is to be generated for. 'All' if all available simulations are to be taken into account. Default: All

Property	Description
`'OutputDirectory'`	Directory to generate CTC code coverage report in. Default: .\CTCHTML
`'Tool'`	Name of supported code coverage tool. The following tools are supported: ▪ TL - TargetLink built-in ▪ CTC - CTC Code Coverage Analyzer for C/C++ Note: Third-party tools are not installed with TargetLink installation and must be installed separately.
`'Show'`	If true, opens generated code coverage report in the system browser. Default: false

Input parameters

The following input parameters are available:

Parameter	Description
`model`	Simulink identifier of root model to generate code coverage report for

tlCodeCoverage('GetCovTools')

Purpose

Gets supported code coverage tools

Description

This command is part of the tlCodeCoverage function.

Syntax

```
covTools = tlCodeCoverage('GetCovTools')
```

Output parameters

The following output parameters are available:

Parameter	Description
`covTools`	Option struct of the supported code coverage tools. It contains the following fields: ▪ name - Tool name ▪ validlevelnames - Valid level names ▪ validlevels - Valid level numbers ▪ defaultlevel - Default tool's level as a number

tlCodeCoverage('MoveCTCFiles', propertyName, propertyValue, ...)

Purpose

Moves CTC files (*.sym and *.dat) created during build process and simulation to the specified folder. These files can later be used to create a combined report of code coverage analysis for multiple builds and simulations. By default, each time an application is (re)built, the existing CTC files are moved from the TargetLink build folder into the .\CTCRESULTS\TLBuild_<BuildTimeStamp> folder.

See also: GenerateCombinedReport

Description

This command is part of the tlCodeCoverage function.

Syntax

```
tlCodeCoverage('MoveCTCFiles', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'SourceDirectory'`	Directory to move the CTC files from.
`'DestinationDirectory'`	Directory to move the CTC files to. Default: .\CTCRESULTS\TLBuild_<TimeStamp>

tlCodeCoverage('GenerateCombinedReport', propertyName, propertyValue, ...)

Purpose

Generates combined report from all CTC files (*.sym and *.dat) residing in and below the specified folders.

See also: MoveCTCFiles

Description

This command is part of the tlCodeCoverage function.

Syntax

```
tlCodeCoverage('GenerateCombinedReport', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'OutputDirectory'`	Directory to generate CTC code coverage report in. Default: .\CTCHTML
`'Show'`	If true, opens generated code coverage report in the system browser. Default: false
`'SourceDirectories'`	Directories containig CTC files to generate report from.

tl_code_coverage

tl_code_coverage

Purpose

Generates code coverage documents.

Syntax overview

The following syntaxes are available:

```
tl_code_coverage(DocumentScope, CodeCoverageData)
```
Generates report with code coverage test results and opens it in browser

```
reportFile = tl_code_coverage(Scope, CodeCoverageData)
```
Generates report with code coverage test results and returns its name

```
tl_code_coverage('RemoveAllCCMarks')
```
Removes all code coverage macros from production code

Input parameters

The following input parameters are available:

Parameter	Description
`DocumentScope`	The following document scopes are available: ▪ Overview - Summarizes code coverage test results ▪ Report - Adds a list of C source files and associated coverage measurements as an additional report to the overview
`Scope`	The following document scopes are available: ▪ GenerateOverview - Summarizes code coverage test results ▪ GenerateReport - Adds a list of C source files and associated coverage measurements as an additional report to the overview
`CodeCoverageData`	DD reference

Output parameters

The following output parameters are available:

Parameter	Description
`reportFile`	File name of generated report

Related topics

Basics

Measuring the Code Coverage of Generated Code (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

tl_download

tl_download

Purpose

Loads production code applications to EVBs or MATLAB memory space.

Description

This function loads the simulation application into the MATLAB memory space (SIL simulation) or to the evaluation board (PIL simulation).

To load the simulation application to an evaluation board, you have to specify the name of the simulation configuration describing the simulation platform to load the simulation application to. Available simulation configurations are listed in the TargetLink Preferences Editor that you can open by invoking tl_pref in the MATLAB Command Window.

Default: The simulation configuration selected on the Simulation page of the TargetLink Main Dialog.

Syntax

```
tl_download(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Model the simulation application to be downloaded was generated for. Default: current model
`SimConfig`	Target simulation configuration. Default: simulation configuration selected in model's Main Dialog
`SimMode`	Simulation mode, possible values: ▪ 'TL_CODE_HOST' - Loads simulation application to MATLAB memory space ▪ 'TL_CODE_TARGET' - Loads simulation application to EVB Default: 'TL_CODE_TARGET'

Example

```
% Download a production code PIL application for model VCFP
tl_download('Model','vcfp','SimulationMode','TL_CODE_TARGET');
```

Related topics

Basics

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

HowTos

How to Download Simulation Applications (📖 TargetLink Preparation and Simulation Guide)

References

Build Hook Scripts (📖 TargetLink File Reference)
TargetLink Main Dialog Block (📖 TargetLink Model Element Reference)
tl_compile_target.. 109

tl_get_sim_mode

tl_get_sim_mode

Purpose

Returns the simulation mode(s) of TargetLink subsystem(s) in a model.

Description

Retrieves the simulation modes of TargetLink subsystems in a model. If called with fewer than two output arguments, the function displays error messages in a modal dialog. Otherwise, error messages are returned in a message struct.

Syntax

```
[simMode, msgStruct] = tl_get_sim_mode(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`model`	Simulink model. Default: current model, as returned by the ds_get_current_model function
`tlSubsystems`	Cell array with names of TargetLink subsystems in specified Simulink model. Default: names of all TargetLink subsystems in model

Output parameters

The following output parameters are available:

Parameter	Description
`simMode`	Simulation mode, returned as string or cell array of strings, depending on number of TargetLink subsystems. The following values are supported: ▪ TL_BLOCKS_HOST - MIL simulation mode ▪ TL_CODE_HOST - SIL simulation mode ▪ TL_CODE_TARGET - PIL simulation mode ▪ UNKNOWN - Simulation mode could not be evaluated
`msgStruct`	If an error occurs, a TargetLink message struct with error messages. Otherwise, an empty matrix.

Related topics

Basics

Basics on Simulation Modes and Preconditions (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

References

tl_set_sim_mode

tl_set_sim_mode

Purpose

Switches the simulation mode(s) of specified TargetLink subsystems.

Description

Switches the simulation modes of the specified TargetLink subsystems. If called without output arguments, the function displays error messages in a modal dialog. Otherwise, error messages are returned in a message struct.

Syntax

```
msgStruct = tl_set_sim_mode(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as 📖 propertyName/propertyValue pairs, refer to the following table:

Property	Description
`model`	Simulink model which contains TargetLink subsystem(s). Default: current model, as returned by ds_get_current_model function
`tlSubsystems`	Cell array with names of TargetLink subsystems in specified model. Default: names of all TargetLink subsystems in model

Property	Description
`simMode`	Simulation mode to set TargetLink subsystems to.The following values are supported: ▪ 'TL_BLOCKS_HOST' - MIL simulation mode. ▪ 'TL_CODE_HOST' - SIL simulation mode. (default) ▪ 'TL_CODE_TARGET' - PIL simulation mode. ▪ 'TL_CODE_SFCN' - Stand-alone mode. If you select the stand-alone mode, the specified TargetLink subsystems is replaced with production code S-functions. This cannot be reverted. Default:'TL_BLOCKS_HOST'
`sFcnNames`	Names of simulation S-functions to be used for SIL or PIL simulation of specified TargetLink subsystems. Number of elements in cell array of strings must match number of TargetLink subsystems. Default: empty cell array
`subFunctions`	Names of production code functions to be used for SIL or PIL simulation of specified TargetLink subsystems. Number of elements in cell array of strings must match number of the TargetLink subsystems. Default: empty cell array

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	If an error occurs, a TargetLink message struct with error messages. Otherwise, an empty matrix.

Related topics

Basics

Basics on Simulation Modes and Preconditions (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

References

tl_build_host.. 101
tl_build_target.. 103
tl_get_sim_mode... 131

tl_sim

tl_sim

Purpose

Starts the simulation for a TargetLink model with logging support for referenced models.

Description

This function launches a MIL/SIL/PIL simulation for TargetLink, depending on the simulation mode. The model to be simulated must be open. The command can be used with the same input parameters as the Simulink sim command.

If the model references other models, this function performs the following before simulation starts:

- Detects all models referenced from the current model
- Opens referenced models and rebuilds their S-functions, if at least one of the following conditions is fulfilled:
  - This is the first simulation with the current model
  - Referenced models have been modified since the last run of tl_sim
  - The 'Global logging option' setting of the current model was changed since the last run of tl_sim
  - The 'Logging in referenced models' setting of the current model was changed since the last run of tl_sim
  - The Simulink logging format (ModelDataLogs or Dataset) of the root model was changed since the last run of tl_sim
  - The topology of the referenced models has been changed since last run of tl_sim
  - The referenced model has no simulation target

To perform MIL logging for referenced models, you must use the tl_sim command. If you use the Simulink 'Start simulation' button or Simulink's sim command instead, only blocks in the root model are logged, but not the blocks of referenced models.

You can link this command to a Tool Selector block. Select the Start Simulation tool.

The tl_sim command is performed when you start a simulation from the TargetLink Main dialog or any TargetLink Plot dialog.

Syntax

```
tl_sim(model, parameters)
```

Input parameters

The following input parameters are available:

Parameter	Description
`model`	Name of model to be simulated
`parameters`	Additional parameters, refer to Simulink documentation on sim command for possible parameters

Example

```
% open TargetLink demo for referenced models
tl_demos model_referencing

% Make sure that MIL mode is active
tl_set_sim_mode('Model','mdlref_fuelsys','SimMode','TL_BLOCKS_HOST');

% Perform a MIL simulation
tl_sim('mdlref_fuelsys');

% Perform build for root model and all referenced models
tl_build_host('Model','mdlref_fuelsys','IncludeSubItems','on');

% Perform a SIL simulation
tl_sim('mdlref_fuelsys');
```

Related topics

Basics

Basics on Simulation Modes and Preconditions (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

References

tl_set_sim_mode.. 132

tl_tl2rti

tl_tl2rti

Purpose

Prepares simulation of production code in dSPACE prototyping environment.

Description

This functions generates stand-alone S-functions from TargetLink subsystems in the source model. It replaces specified subsystems in the destination model by subsystems that contain S-function blocks. This lets you use TargetLink code in prototyping scenarios, such as embedding TargetLink code in real-time applications running on dSPACE real-time hardware using dSPACE Real-Time Interface (RTI). In addition, this function can perform the following steps:

- Copies all the source/header/library files required to compile and link the generated code to the real-time application into the destination directory.
- Creates a user makefile (<destinationModel>_usr.mk) listing these files, as required by dSPACE RTI.

  Updates existing user makefile and backs up original (<destinationModel>_usr.mk.old).
- Generates a user trace file (<destinationModel>_usr.trc) for the created stand-alone S-function(s) for use with a control and measurement tool such as ControlDesk.

Syntax

```
bSuccess = tl_tl2rti(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`SrcModel`	Name of root model. Default: current model
`TlSubsystems`	Name of TargetLink subsystem. Default: all TargetLink subsystems in model
`DDCodeGenerationUnits`	List of DD CodeGenerationUnit objects or DD CodeGenerationUnitGroup objects that code is to be generated for, specified as list of DD paths or list of DD handles. The specified DD objects must exist in DD0.
`DestModel`	Name of destination model. If destination model does not exist in destination directory, TargetLink interactively asks if destination model is to be created as a copy of source model.
`DestDirectory`	Directory of destination model. Default: .\TLStandAloneSFcn
`DestBlocks`	Full Simulink paths of destination model's blocks which are to be replaced by stand-alone S-function subsystems. Default: blocks at root of destination model named as specified TargetLink subsystems
`IncludeSubItems`	Generates code for nested systems (subsystems configured for incremental code generation and referenced models). Default: off

Property	Description
`GenerateGlobalSymbols`	If 'on', creates, compiles, and links a C-module tlstandalone_<tlSubsystem>_globals.c. Contains definitions of symbols needed for simulation (such as interface variables) that are specified as extern global and therefore not defined in generated code. Set to 'off' if these definitions are made in a user module to be compiled and linked together with generated code to avoid double symbol definitions. Default: 'on'
`GenerateTRC`	Enables or disables user TRC file generation/update. If 'on' a user TRC file, <DestModel>usr.trc, is generated/updated. Default: 'on'
`TrcFileHierarchy`	Hierarchy in TRC file, possible values are: ▪ 'functions' - TRC file reflects hierarchy of generated step functions ▪ 'subsystems' - TRC file reflects hierarchy of Simulink model Default: 'subsystems'
`FixPointSupport`	Enables or disables support for fixed point data types: ▪ 'on' - TRC file contains description of all global variables in generated code ▪ 'off' - TRC file contains description of global floating point variables and fixed-point variables without scaling (LSB=1, Offset=0) Default: 'on'
`UpdateUsrMakeFile`	Enables or disables RTI user makefile update. ▪ 'on' - Updates existing RTI user makefile <DestModel>_usr.mk If file does not exist, creates a new one ▪ 'off'- Does not update/create RTI user make file Default: 'on'
`GeneratePPCApplication`	Enables or disables generation of RTI PPC application. ▪ 'on' - Generates real-time application for dSPACE board with PPC processor is generated using dSPACE RTI, provided that RTI is installed ▪ 'off'- Does not generate real-time application Default: 'off'
`SkipCodeGeneration`	If 'on', skips the production code generation step. The user have to ensure in this case that a required production code was generated and that its description (DD Subsystem object, DD Application object) exists in the currently open DataDictionary. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	▪ 1 - Success ▪ 0 - Failure

Example

```
% In the .\RCPModel subdirectory generate a RCP model pipt1_rcp as a copy of the source model pipt1.
% For all TargetLink subsystems in the pipt1 model generate a stand-alone S-function and in the
% destination model pipt1_rcp replace the TargetLink subsystem by the corresponding S-function subsystems.
% Copy all necessary files needed for generating an RTI application from the model pipt1_rcp.
tl_tl2rti('SrcModel','pipt1',...
'DestModel','pipt1_rcp',...
'DestDirectory','.\RCPModel');
```

Related topics

HowTos

How to Prepare the Simulation of Production Code in a dSPACE Prototyping Environment (📖 TargetLink Preparation and Simulation Guide)

tlProductionCodeSILCompiler

tlProductionCodeSILCompiler

Purpose

Specifies the compiler to be used for building SIL simulation application.

Syntax overview

The following syntaxes are available:

`tlProductionCodeSILCompiler('-setup')`

Allows interactive specification of one of the supported Microsoft Express Edition compiler to be used for the building a SIL simulation.
The SIL compiler can differ from the selected Mex compiler. This can be usefull if a SIL Debugging feature is required, but it is not possible with a free Mex compiler, such as GCC. In this case a free Microsoft Express Edition, that a SIL Debugging is supported for, can be installed and selected as a SIL compiler.

`tlProductionCodeSILCompiler('-default')`

Resets the specification of the compiler to be used for building a SIL simulation application to default. It means that TargetLink assumes the current Matlab Mex-compiler as a SIL compiler.

Related topics

HowTos

How to Set or Change MEX and SIL Compilers (📖 TargetLink Preparation and Simulation Guide)

tlStartCallbackWithTimer

tlStartCallbackWithTimer

Purpose starts a callback within a timer

Syntax

```
tlStartCallbackWithTimer(callback)
```

Input parameters The following input parameters are available:

Parameter	Description
`callback`	callback string to be started within timer

Related topics Basics

Basics on Simulation Modes and Preconditions (📖 TargetLink Preparation and Simulation Guide)

tlRebuildFixedPointLibrary

tlRebuildFixedPointLibrary

Purpose Rebuilds the TargetLink Fixed-Point Library.

Syntax

```
[bError, msgList] = tlRebuildFixedPointLibrary(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Board`	Name of evaluation board. If not specified, the Fixed-Point Library will be rebuilt for the simulation target set in the main dialog
`Compiler`	Compiler Name. Can be omitted if only one compiler is specified for the specified board or the rebuild is performed for the HostPC32/HostPC64.
`CodeOpt`	Code generation target setting. Default: 'Generic ANSI-C'
`Assembler`	Code generation target setting. Default: 'off'
`RebuildAll`	If 'on', rebuilds all files, also existing OBJ files. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`bError`	error flag
`msgList`	error message

Example

```
% rebuild Fixed-Point Library for the simulation target set in the main dialog
tlRebuildFixedPointLibrary

% rebuild Fixed-Point Library for specified target
tlRebuildFixedPointLibrary('Board','TBTC1767','Compiler','Task32','CodeOpt','Generic Tricore1767/Task')

% rebuild Fixed-Point Library for HostPC
tlRebuildFixedPointLibrary('Board','HostPC64')

% rebuild Fixed-Point Library for TOM target
tlRebuildFixedPointLibrary('Board','HostPC64','CodeOpt','Generic C16x/Task86')
tlRebuildFixedPointLibrary('Board','TBTC1767','Compiler','Task32','CodeOpt','TOM Tricore1767/Task32','Assembler','on')
tlRebuildFixedPointLibrary('Board','TBTC1767','Compiler','Task32','CodeOpt','TOM Tricore1767/Task32','Assembler','off')
```

Related topics

Basics

Basics on Specifying the Location of the Sources and Binaries of the TargetLink Fixed-Point Library (📖 TargetLink Customization and Optimization Guide)

Accessing and Filtering Logged Simulation Data

tl_access_logdata

Where to go from here

Information in this section

tl_access_logdata

Purpose

Accesses logged simulation data on the TargetLink Data Server.

Syntax overview

The following syntaxes are available:

`simlabel = tl_access_logdata('GetSimulationLabels')`

Gets labels of all simulations saved in RAM

`simlabel = tl_access_logdata('GetLastSimulationLabel')`

Gets label of last simulation

`[blocks, msgstruct] = tl_access_logdata('GetLoggedBlocks', simlabel)`

Gets paths of all TargetLink blocks with logged simulation data

`[siminfo, msgstruct] = tl_access_logdata('GetSimulationInfo', simlabel)`

Gets simulation info (model, time, start time, stop time, lock, tlsubsystems).

`msgstruct = tl_access_logdata('DeleteSimulation', simlabel)`

Deletes simulation

`msgstruct = tl_access_logdata('SetSimulationLabel', simlabel, newlabel)`

Replaces existing simulation label(s) with new label(s)

`msgstruct = tl_access_logdata('SetSimulationLock', simlabel, lock)`

Sets simulation lock

`msgstruct = tl_access_logdata('SaveSimulation', simlabel, filename)`

Saves simulation to file

`msgstruct = tl_access_logdata('LoadSimulation', filename)`

Loads simulation from file

`[signallogdata, msgstruct] = tl_access_logdata('GetLoggedSignal', propertyName, propertyValue, ...)`

Returns logged simulation data (struct, Timeseries or Simulink.TimeSeries object).
The tl_access_logdata('GetLoggedSignal', ...) command can apply several filters to the logged simulation data and returns the cut set.

`[signalinfo, msgstruct] = tl_access_logdata('GetLoggedSignalInfo', propertyName, propertyValue, ...)`

Returns signal-specific information for logged simulation data.
The tl_access_logdata('GetLoggedSignalInfo', ...) command can apply several filters to the logged simulation data and returns the cut set.

`msgstruct = tl_access_logdata('PlotSignal', propertyName, propertyValue, ...)`

Displays logged simulation data in TargetLink's Plot Overview Window

`msgstruct = tl_access_logdata('SetNumberOfBufferedSimulinkLogSamples', numberofsamples)`

Limits number of samples for Simulink log buffer used for logging in MIL simulation. This affects the performance and the memory consumption. A shorter number of samples decreases memory consumption but increases simulation duration. The number of TargetLink log samples is not limited by this command.

`numberofsamples = tl_access_logdata('GetNumberOfBufferedSimulinkLogSamples')`

Gets current number of samples for Simulink log buffer `[numberofsamples, msgstruct] = tl_access_logdata('CalculateNumberOfBufferedSimulinkLogSamples', model)`

Calculates and sets recommended number of samples for Simulink log buffer for a specific model. The following model attributes affect the calculation:

- Number of TargetLink blocks
- Number of TargetLink subsystems in MIL mode
- Log settings of TargetLink blocks
- Global logging option

Example

```
% open TargetLink pipt1 demo
tl_demos pipt1

% Perform build
tl_build_host('Model','pipt1');

% perform a MIL simulation
tl_set_sim_mode('Model','pipt1','SimMode','TL_BLOCKS_HOST');
tl_sim('pipt1');

% perform a SIL simulation
tl_set_sim_mode('Model','pipt1','SimMode','TL_CODE_HOST');
tl_sim('pipt1');

% Get simulation labels
simLabel =tl_access_logdata('GetSimulationLabels')

% Rename simulation labels of last 2 simulations
tl_access_logdata('SetSimulationLabel', simLabel{end},'sil');
tl_access_logdata('SetSimulationLabel', simLabel{end-1},'mil');

% Get simulation data from MIL simulation of block 'e'
ld1 =tl_access_logdata('GetLoggedSignal','SimLabel','mil','Block','pipt1/picontroller/Subsystem/picontroller/e')

% Get simulation data from SIL simulation of variable built for block 'e'
ld2 =tl_access_logdata('GetLoggedSignal','SimLabel','sil','Block','pipt1/picontroller/Subsystem/picontroller/e')

sigInfo =tl_access_logdata('GetLoggedSignalInfo','Block','pipt1/picontroller/Subsystem/picontroller/e')

% Compare simulation results - SIL/PIL simulation results are saved as real world values
% Display maximum difference relative to specified LSB
maxDiffInLsb =max(abs(ld1.signal.y - ld2.signal.y)/ sigInfo.lsb)

% Temporary change plot color for simulation labeled 'sil' to red
tl_access_logdata('PlotSignal','SimLabel','sil','Color','r')
```

Remarks

You can specify some colors by name (or abbreviation) instead of the RGB value:

- 'yellow' ('y')
- 'magenta' ('m')
- 'cyan' ('c')
- 'red' ('r')
- 'green' ('g')

- 'blue' ('b')
- 'white' ('w')
- 'silver' ('s')
- 'black' ('k')

Related topics

Basics

Basics on Logging (📖 TargetLink Preparation and Simulation Guide)

tl_access_logdata('GetSimulationLabels')

Purpose

Gets labels of all simulations saved in RAM

Description

This command is part of the tl_access_logdata function.

Syntax

```
simlabel = tl_access_logdata('GetSimulationLabels')
```

Output parameters

The following output parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)

tl_access_logdata('GetLastSimulationLabel')

Purpose

Gets label of last simulation

Description

This command is part of the tl_access_logdata function.

Syntax

```
simlabel = tl_access_logdata('GetLastSimulationLabel')
```

Output parameters

The following output parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)

tl_access_logdata('GetLoggedBlocks', simlabel)

Purpose

Gets paths of all TargetLink blocks with logged simulation data

Description

This command is part of the tl_access_logdata function.

Syntax

```
[blocks, msgstruct] = tl_access_logdata('GetLoggedBlocks', simlabel)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)

Output parameters

The following output parameters are available:

Parameter	Description
`blocks`	Paths of all TargetLink blocks with logged simulation data
`msgstruct`	Messages generated during API method execution

tl_access_logdata('GetSimulationInfo', simlabel)

Purpose

Gets simulation info (model, time, start time, stop time, lock, tlsubsystems).

Description

This command is part of the tl_access_logdata function.

Syntax

```
[siminfo, msgstruct] = tl_access_logdata('GetSimulationInfo', simlabel)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)

Output parameters

The following output parameters are available:

Parameter	Description
`siminfo`	Simulation info (model, date, starttime, stoptime, locked, tlsubsystems)
`msgstruct`	Messages generated during API method execution

tl_access_logdata('DeleteSimulation', simlabel)

Purpose

Deletes simulation

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('DeleteSimulation', simlabel)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('SetSimulationLabel', simlabel, newlabel)

Purpose

Replaces existing simulation label(s) with new label(s)

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('SetSimulationLabel', simlabel, newlabel)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)
`newlabel`	New simulation label(s)

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('SetSimulationLock', simlabel, lock)

Purpose

Sets simulation lock

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('SetSimulationLock', simlabel, lock)
```

Input parameters

The following input parameters are available:

Parameter	Description
`lock`	Simulation lock(s) for given simulation(s), Boolean. If 'on', simulation is locked against modification and deletion. Number of locks and simulation labels must match.
`simlabel`	Simulation label(s)

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('SaveSimulation', simlabel, filename)

Purpose

Saves simulation to file

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('SaveSimulation', simlabel, filename)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simlabel`	Simulation label(s)
`filename`	Name of MAT file containing simulation data

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('LoadSimulation', filename)

Purpose

Loads simulation from file

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('LoadSimulation', filename)
```

Input parameters

The following input parameters are available:

Parameter	Description
`filename`	Name of MAT file containing simulation data

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('GetLoggedSignal', propertyName, propertyValue, ...)

Purpose

Returns logged simulation data (struct, Timeseries or Simulink.TimeSeries object).

The tl_access_logdata('GetLoggedSignal', ...) command can apply several filters to the logged simulation data and returns the cut set.

Description

This command is part of the tl_access_logdata function.

Syntax

```
[signallogdata, msgstruct] = tl_access_logdata('GetLoggedSignal', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'simlabel'`	Simulation label(s). Default: (last simulation)
`'block'`	Block path(s). Default: (all blocks)
`'signalname'`	Signal name(s). Default: (all signals)
`'starttime'`	Time of first simulated value to return. Default: 0.0
`'stoptime'`	Time of last simulated step to return. Default: last simulated time
`'format'`	Data format, the following formats are supported: ▪ struct ▪ timeseries ▪ simulink.timeseries Default: struct

Output parameters

The following output parameters are available:

Parameter	Description
`signallogdata`	Logged simulation data (struct with fields for data and time, Timeseries object or Simulink.TimeSeries object - according to the specified format)
`msgstruct`	Messages generated during API method execution

tl_access_logdata('GetLoggedSignalInfo', propertyName, propertyValue, ...)

Purpose

Returns signal-specific information for logged simulation data.

The tl_access_logdata('GetLoggedSignalInfo', ...) command can apply several filters to the logged simulation data and returns the cut set.

Description

This command is part of the tl_access_logdata function.

Syntax

```
[signalinfo, msgstruct] = tl_access_logdata('GetLoggedSignalInfo', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'simlabel'`	Simulation label(s). Default: (last simulation)
`'block'`	Block path(s). Default: (all blocks)
`'signalname'`	Signal name(s). Default: (all signals)

Output parameters

The following output parameters are available:

Parameter	Description
`signalinfo`	Signal-specific information for logged simulation data as a struct: ▪ simulationlabel - Label to identify simulation ▪ block - Block path ▪ signalname - Signal name (used for bus signals and TargetLink blocks with several output ports) ▪ simulationmode - ['MIL','SIL','PIL','unknown'] ▪ modelname - Name of the model containing logged block signal (can be a root model or a referenced model) ▪ signaltype - Data type of the Simulink signal ▪ tltype - TargetLink base data type (to be used in generated code) ▪ lsb - LSB (to be used in generated code) ▪ offset - Offset (to be used in generated code) ▪ min - Minimum value assured by user ▪ max - Maximum value assured by user ▪ simulatedmin - Struct containing the minimum value that appeared during simulation and the corresponding time ▪ simulatedmax - Struct containing the maximum value that appeared during simulation and the corresponding time
`msgstruct`	Messages generated during API method execution

tl_access_logdata('PlotSignal', propertyName, propertyValue, ...)

Purpose

Displays logged simulation data in TargetLink's Plot Overview Window

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('PlotSignal', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'simlabel'`	Simulation label(s). Default: (last simulation)
`'block'`	Block path(s). Default: (all blocks)
`'signalname'`	Signal name(s). Default: (all signals)
`'plotchannel'`	Plot channels. Default: all plot channels (-1)
`'color'`	Plot color, to be specified as: ▪ [String of Simulink color name (e.g., red) or its shortform (e.g., 'r') ▪ RGB array ([0..1 0..1 0..1]) ▪ uint8([0..255 0..255 0..255]) The number of specified colors must match the number of simulation labels. Default: current plot color
`'savecolor'`	Permanently assigns a color to given simulation. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('SetNumberOfBufferedSimulinkLogSamples', numberofsamples)

Purpose

Limits number of samples for Simulink log buffer used for logging in MIL simulation. This affects the performance and the memory consumption. A shorter number of samples decreases memory consumption but increases simulation duration. The number of TargetLink log samples is not limited by this command.

Description

This command is part of the tl_access_logdata function.

Syntax

```
msgstruct = tl_access_logdata('SetNumberOfBufferedSimulinkLogSamples', numberofsamples)
```

Input parameters

The following input parameters are available:

Parameter	Description
`numberofsamples`	Number of simulation steps after Simulink log data is transferred to TargetLink. Default: 'inf'

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution

tl_access_logdata('GetNumberOfBufferedSimulinkLogSamples')

Purpose

Gets current number of samples for Simulink log buffer

Description

This command is part of the tl_access_logdata function.

Syntax

```
numberofsamples = tl_access_logdata('GetNumberOfBufferedSimulinkLogSamples')
```

Output parameters

The following output parameters are available:

Parameter	Description
`numberofsamples`	Number of simulation steps after Simulink log data is transferred to TargetLink. Default: 'inf'

tl_access_logdata('CalculateNumberOfBufferedSimulinkLogSamples', model)

Purpose

Calculates and sets recommended number of samples for Simulink log buffer for a specific model. The following model attributes affect the calculation:

- Number of TargetLink blocks
- Number of TargetLink subsystems in MIL mode
- Log settings of TargetLink blocks
- Global logging option

Description

This command is part of the tl_access_logdata function.

Syntax

```
[numberofsamples, msgstruct] = tl_access_logdata('CalculateNumberOfBufferedSimulinkLogSamples',
model)
```

Input parameters

The following input parameters are available:

Parameter	Description
`model`	Simulink identifier for model

Output parameters

The following output parameters are available:

Parameter	Description
`msgstruct`	Messages generated during API method execution
`numberofsamples`	Number of simulation steps after Simulink log data is transferred to TargetLink. Default: 'inf'

Modifying Calibration Parameters During Simulation

Where to go from here

Information in this section

tlSimInterface

Where to go from here

Information in this section

tlSimInterface

Purpose M interface for the TargetLink simulation engine.

Example

```
% Connect to simulation platform HostPC.
% Use tl_get_host_simconfig function to get the board name 'HostPC32' or 'HostPC64' automatically
hostSimInfo = tl_get_host_simconfig;
[hPlatform, msgStruct]=tlSimInterface('ConnectToSimPlatform','BoardName', hostSimInfo.board);
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display('ShowDialog','off');
return;
end

% Reset production code simulation application.
% Before the start of the simulation, the application must be reset, i.e.,
% global variables without initialization value are set to 0
% and other variables are assigned their initial values
msgStruct =tlSimInterface('Reset', hPlatform);
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display('ShowDialog','off');
return;
end

% Get the addresses of selected parameters and write a new value to them before starting the simulation
blockPath ='online_parameter_modification/LissajousFigure/Subsystem/LissajousFigure/Frequency';
blockVariable ='gain';
[varInfoFrequency, msgStruct]=tlSimInterface('GetBlockVarAddr', hPlatform,...
'TLBlock', blockPath ,...
'TLBlockVariable', blockVariable);
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display('ShowDialog','off');
return;
end
msgStruct =tlSimInterface('Write', hPlatform,...
'VarInfos', varInfoFrequency,...
'Data',1);
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display('ShowDialog','off');
return;
end

% Disable automatic reset of simulation application by the simulation S-function.
% Otherwise the modification of the parameter value will be lost.
msgStruct =tlSimInterface('DisableResetBySimulationSFcn', hPlatform);
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display('ShowDialog','off');
return;
end

% Disconnect from simulation platform
msgStruct  =tlSimInterface('DisConnectFromSimPlatform', hPlatform);
if~isempty(msgStruct)
ds_error_register(msgStruct);
ds_error_display('ShowDialog','off');
return;
end
```

```
The online_parameter_modification demo model provides further examples.
```

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)
Examples of Implementing Online Parameter Modification (📖 TargetLink Preparation and Simulation Guide)
Using the TargetLink M-Script Interface (API) (📖 TargetLink Interoperation and Exchange Guide)

tlSimInterface('CallFcn', hPlatform, propertyName, propertyValue, ...)

Purpose

Calls a function specified by its info structure (as returned by GetFcnAddr command). Only functions with the prototype void <fcnName>(void) can be called. Otherwise the application will crash.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('CallFcn', hPlatform, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'FcnInfo'`	Info structure that describes a function to be called and returned by GetFcnAddr command.

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('ConnectToSimPlatform', propertyName, propertyValue, ...)

Purpose

Connects to the specified simulation platform. Further actions relate to the production code simulation application running on the simulation platform.

Description

This command is part of the tlSimInterface function.

Syntax

```
[hPlatform, msgStruct] = tlSimInterface('ConnectToSimPlatform', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'BoardName'`	Name of simulation platform: ▪ <EVB> - Name of evaluation board ▪ 'HostPC32' - SIL simulation, TargetLink 32-bit ▪ 'HostPC64' - SIL simulation, TargetLink 64-bit

Output parameters

The following output parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (📖 TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('DisableResetBySimulationSFcn', hPlatform)

Purpose

Disables a reset of the production code application by a simulation S-function at simulation start (SIL/PIL). TargetLink automatically enables reset after simulation (SIL/PIL) finishes.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('DisableResetBySimulationSFcn', hPlatform)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (📖 TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

References

tlSimInterface('DisableRestartBySimulationSFcn', hPlatform)

Purpose

Disables a call of TargetLink subsystem-specific main restart functions (if any) at start of the simulation of the production code simulation application (SIL/PIL). TargetLink automatically enables a call of restart functions after simulation (SIL/PIL) finishes.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('DisableRestartBySimulationSFcn', hPlatform)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('DisconnectFromSimPlatform', hPlatform)

Purpose

Disconnects from the specified simulation platform.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('DisconnectFromSimPlatform', hPlatform)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (📖 TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('EnableResetBySimulationSFcn', hPlatform)

Purpose

Enables a reset of the production code application by a simulation S-function previously disabled by the DisableResetBySimulationSFcn command.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('EnableResetBySimulationSFcn', hPlatform)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

References

tlSimInterface('DisableResetBySimulationSFcn', hPlatform).. 162
tlSimInterface('Read', hPlatform, propertyName, propertyValue, ...)....................................... 176

tlSimInterface('EnableRestartBySimulationSFcn', hPlatform)

Purpose

Enables a call of TargetLink subsystem-specific main restart functions at start of the simulation of the production code simulation application (SIL/PIL) previously disabled by the DisableRestartBySimulationSFcn command.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('EnableRestartBySimulationSFcn', hPlatform)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('GetBlockVarAddr', hPlatform, propertyName, propertyValue, ...)

Purpose

Obtains the address information of variables defined in the production code simulation application and specified by the block it is generated from.

Description

This command is part of the tlSimInterface function.

Syntax

```
[varInfo, msgStruct] = tlSimInterface('GetBlockVarAddr', hPlatform, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'TLBlock'`	Simulink identifier of TargetLink block
`'TLBlockVariable'`	Name of block variable, such as 'gain', 'output', etc., that the code variable was generated for
`'TLBlockVariableIndex'`	Index of block variable if block contains multiple block variables with identical name, such as multiple outputs. Optional.
`'ArrayOfStructIndexSequence'`	Sequence of indices to the fields of an array of struct variable. The field index of the structure of substructure has to be specified for each structure or substructure on the path to the leaf struct component. If the structure or substructure is a vector, one index has to be specified. If the structure or substructure is 2-D matrix, two indices have to be specified. If a structure or substructure is scalar, -1 has to be used as the index. Example: [1 0 -1 2] for the access to ArrayOfStructVar[1][0].c.d[2].e. Optional.

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message
`varInfo`	Variable info structure with the following fields: ▪ Name - Variable name ▪ Type - Data type ▪ Width - Dimension; [] denotes a scalar, [n] denotes an n-element vector, [n m] denotes an n x m matrix ▪ Deposit - Deposit in memory, possible values: 'row' and 'column'. Default: 'row' ▪ LSB - Least significant bit ▪ Offset - Offset ▪ Address - Address ▪ Module - Module the variable is defined in

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Writing Parameter Values via Hook Scripts (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('GetDDVarAddr', hPlatform, propertyName, propertyValue, ...)

Purpose

Obtains the address information of variables defined in the production code simulation application and specified by the DD identifier (handle or path) of their corresponding Variable objects in the Subsystem area.

Description

This command is part of the tlSimInterface function.

Syntax

```
[varInfoList, msgStruct] = tlSimInterface('GetDDVarAddr', hPlatform, propertyName,
propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'DDVariables'`	DD identifiers (paths/handles) of DD variable objects
`'ArrayOfStructIndexSequences'`	Cell array of vectors of sequences of indices to the fields of an array of struct variables. The cell array lenght is equal to the length of the specified DD Variables. If a DD Variable does not represent a field of an array of struct variable the corresponding vector in the cell array remains empty. Optional.

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`varInfoList`	Vector of variable info structures with the following fields: ▪ Name - Variable name ▪ Type - Data type ▪ Width - Dimension; [] denotes a scalar, [n] denotes an n-element vector, [n m] denotes an n x m matrix ▪ Deposit - Deposit in memory, possible values: 'row' and 'column'. Default: 'row' ▪ LSB - Least significant bit ▪ Offset - Offset ▪ Min - Variable lower limit ▪ Max - Variable upper limit ▪ Address - Address ▪ Module - Module the variable is defined in

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (📖 TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('GetFcnAddr', hPlatform, propertyName, propertyValue, ...)

Purpose

Obtains the address information of functions defined in the production code simulation application and specified by their names and the name of the module they are defined in.

Description

This command is part of the tlSimInterface function.

Syntax

```
[fcnInfoList, msgStruct] = tlSimInterface('GetFcnAddr', hPlatform, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Functions'`	Names of functions whose address is to be obtained
`'Module'`	Names of modules the variables/functions are defined in

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`fcnInfoList`	Vector of function info structures with the following fields: ▪ Name - Function name ▪ Kind - Function's address kind, possible values are: ▪ Default ▪ Far ▪ Near ▪ 32Bit_OP_CODE ▪ 16Bit_OP_CODE ▪ Address - Address ▪ Module - Module the function is defined in
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('GetMapVarAddr', hPlatform, propertyName, propertyValue, ...)

Purpose

Obtains the address information of variables defined in the production code simulation application and specified by their names and the name of the module they are defined in.

Description

This command is part of the tlSimInterface function.

Syntax

```
[varInfoList, msgStruct] = tlSimInterface('GetMapVarAddr', hPlatform, propertyName,
propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'MapVariables'`	Names of variables whose address is to be obtained
`'Module'`	Names of modules the variables/functions are defined in

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`varInfoList`	Vector of variable info structures with the following fields: ▪ Name - Variable name ▪ Type - Data type ▪ Width - Dimension; [] denotes a scalar, [n] denotes an n-element vector, [n m] denotes an n x m matrix ▪ Deposit - Deposit in memory, possible values: 'row' and 'column'. Default: 'row' ▪ LSB - Least significant bit ▪ Offset - Offset ▪ Min - Variable lower limit ▪ Max - Variable upper limit ▪ Address - Address ▪ Module - Module the variable is defined in
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('IsApplDownloaded', propertyName, propertyValue, ...)

Purpose

Checks whether production code simulation application was loaded to specified simulation platform

Description

This command is part of the tlSimInterface function.

Syntax

```
[bOk, applFileName, msgStruct] = tlSimInterface('IsApplDownloaded', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'BoardName'`	Name of simulation platform: ▪ <EVB> - Name of evaluation board ▪ 'HostPC32' - SIL simulation, TargetLink 32-bit ▪ 'HostPC64' - SIL simulation, TargetLink 64-bit

Output parameters

The following output parameters are available:

Parameter	Description
`bOk`	▪ 1 - Application loaded to simulation platform ▪ 0 - Application not loaded
`applFileName`	Name and path of production code simulation application loaded to simulation platform
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (📖 TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('Read', hPlatform, propertyName, propertyValue, ...)

Purpose

Reads the values of variables specified by their info structures (returned by one of the address evaluation commands).

Description

This command is part of the tlSimInterface function.

Syntax

```
[data, msgStruct] = tlSimInterface('Read', hPlatform, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'VarInfos'`	Vector of info structures that describe variables to be accessed (see varDescList for detailed structure description)

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`data`	Reads data. Double/cell array of doubles depending on number of read variables.
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

tlSimInterface('Reset', hPlatform)

Purpose

Resets the production code simulation application.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('Reset', hPlatform)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (📖 TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (📖 TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (📖 TargetLink Preparation and Simulation Guide)

References

tlSimInterface('Write', hPlatform, propertyName, propertyValue, ...)

Purpose

Writes values to variables specified by their info structures (returned by one of the address evaluation commands). If the limits (Min/Max components of the info structure) are set, this function issues a warning if the value to be written exceeds the given limits.

Description

This command is part of the tlSimInterface function.

Syntax

```
msgStruct = tlSimInterface('Write', hPlatform, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'VarInfos'`	Vector of info structures that describe variables to be accessed (see varDescList for detailed structure description)
`'Data'`	Data to be written. Double/cell array of doubles depending on number of variables to be written.

Input parameters

The following input parameters are available:

Parameter	Description
`hPlatform`	Simulation platform handle returned by ConnectToSimPlatform command

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Structure containing message info with the following fields: ▪ type - Message type ('fatal', 'error', 'warning', 'note') ▪ number - Message number ▪ title - Message title ▪ msg - Message text ▪ objectName - Simulink/Stateflow/DD object related to message ▪ objectHandle - Handle of Simulink/Stateflow/DD object ▪ module - Name of module (M file) where message occurred ▪ fcn - Name of subfunction in module (M file) ▪ line - Line in the module (M file) where message occurred ▪ clock - Date and time the message occurred ▪ confirmed - 1, if the user has confirmed the message; 0, otherwise ▪ objectKind - Kind of object related to the message

Related topics

Basics

Basics on Modifying Parameter Values for Simulation (TargetLink Preparation and Simulation Guide)

Examples

Example of Switching Data Variants During Simulation (TargetLink Preparation and Simulation Guide)
Example of Writing Calibration Parameters During Simulation (TargetLink Preparation and Simulation Guide)
Example of Writing Parameter Values via Hook Scripts (TargetLink Preparation and Simulation Guide)

tlSimParameterUpdate

Where to go from here

Information in this section

tlSimParameterUpdate

Purpose

Modifies the online parameter automatically

Syntax overview

The following syntaxes are available:

```
msgStruct = tlSimParameterUpdate('UpdateClasses', model, TLSubsystem, inputList, hPlatformHandle,
propertyName, propertyValue, ...)
```

Updates all values of variables of the specified classes

```
msgStruct = tlSimParameterUpdate('UpdateVariables', model, TLSubsystem, inputList, hPlatformHandle,
propertyName, propertyValue, ...)
```

Updates all values of variables specified by DD variable paths

tlSimParameterUpdate('UpdateClasses', model, TLSubsystem, inputList, hPlatformHandle, propertyName, propertyValue, ...)

Purpose

Updates all values of variables of the specified classes

Description

This command is part of the tlSimParameterUpdate function.

Syntax

```
msgStruct = tlSimParameterUpdate('UpdateClasses', model, TLSubsystem, inputList, hPlatformHandle,
propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'ShowUnchanged'`	If true, unchanged values are also listed in the update report
`'Tolerance'`	Values are not updated if differences are within the relative tolerance (in %)
`'ReportName'`	Prefix for report file name

Input parameters

The following input parameters are available:

Parameter	Description
`inputList`	Cell array of variable classes
`model`	Name of current model
`TLSubsystem`	Name of current TargetLink subsystem
`hPlatformHandle`	Handle of simulation platform

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Message struct

tlSimParameterUpdate('UpdateVariables', model, TLSubsystem, inputList, hPlatformHandle, propertyName, propertyValue, ...)

Purpose

Updates all values of variables specified by DD variable paths

Description

This command is part of the tlSimParameterUpdate function.

Syntax

```
msgStruct = tlSimParameterUpdate('UpdateVariables', model, TLSubsystem, inputList, hPlatformHandle,
propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'ShowUnchanged'`	If true, unchanged values are also listed in the update report
`'Tolerance'`	Values are not updated if differences are within the relative tolerance (in %)
`'ReportName'`	Prefix for report file name

Input parameters

The following input parameters are available:

Parameter	Description
`inputList`	Cell array of variable paths
`TLSubsystem`	Name of current TargetLink subsystem
`model`	Name of current model
`hPlatformHandle`	Handle of simulation platform

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Message struct

Exchanging Data

Where to go from here

Information in this section

tl_export_files

tl_export_files

Purpose

Exports files generated by TargetLink to a separate directory.

Description

This function exports files generated by TargetLink to a separate directory:

- Source files that are needed to build a stand-alone application with the production code generated by TargetLink
- Standard TargetLink header files and libraries
- Generated documentation
- A2L files
- DD file

Code generation units (CGUs) that files have to be exported for can be specified by the names of their corresponding DD Subsystem objects:

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

Files of the following CGUs can be excluded from export, if separate code generation was performed for them:

- Referenced models
- Subsystems configured for incremental code generation

Syntax

```
tl_export_files(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Simulink model identifier. Default: current model
`Subsystems`	Name(s) of DD Subsystem object(s). Default: Names of all DD Subsystem objects
`SrcDir`	Source directory. Default: Data Dictionary ProjectFolders specification
`DestDir`	Destination directory. Default: .\export
`Target`	Code generation target processor. Default: Microprocessor from target setting of recent code generation
`Compiler`	Code generation target compiler, e.g., 'MRI45' Default: Compiler from target setting of recent code generation
`CopySystemFiles`	Copies standard TargetLink files to directories <DestDir>\lib and <DestDir>\include. Default: 'on'
`CopyDocumentation`	Copies documentation files to directory <DestDir>\doc Default: 'on'
`DocumentationDir`	Directory which contains documentation files. Default: TLProj\doc
`CopyTools`	Copies a set of tools to directory <DestDir>_tools Default: 'on'
`CopyStandaloneSFcnFiles`	Copies files generated by Standalone Model Manager to destination directory. Default: 'off'
`IgnoreSubItems`	Ignores sub items of specified systems. If 'IgnoreSubItems' is 'on', production code files generated for sub items are not exported. Default: 'off'
`SwcDescFileNames`	Name(s) of AUTOSAR software description files containing description of software components that TargetLink subsystem is to be generated for. If AUTOSAR SWC description is contained in multiple packages and multiple files, these files must be specified as a cell array or with an asterisk.
`Verbose`	Opens TargetLink File Export Utility dialog to let you select a target/compiler combination and the file types to be copied. Default: 'on'

Property	Description
`Application`	Name of DD Application object. Default: Application object of the recent code generation
`BoardName`	Board name, corresponds to the BoardPackages directory name

Example

```
% Open TargetLink fuelsys demo
tl_demos fuelsys;

% Perform a build
tl_build_host('Model','fuelsys');

% Without parameter 'Verbose = off', the following command opens the File Export UI.
% The UI shows only Target/Compiler combinations that match the target/compiler configuration
% of the last code generation.
tl_export_files;

% The following command copies generated files and system files for the specified
% Target/Compiler combination to folder c:\ecu_sources.
% The file export UI is not opened.
tl_export_files(...
'Model','fuelsys',...
'Subsystems',{'fuelratecontroller'},...
'Target','C16x',...
'Compiler','Task86',...
'DestDir','c:\ecu_sources',...
'Verbose','off');
```

Remarks

Files are usually exported after the application is finally validated via simulations and its run-time behavior, memory size and execution time are acceptable.

This command also checks for dependencies to standard header files and libraries, according to the selected target processor and compiler type. The resulting set of files is independent from the TargetLink installation and can be used stand-alone on any computer.

The files in the destination folder can be integrated with any company-specific software environment that often exists for a given ECU.

Related topics

Basics

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

HowTos

How to Export Generated Files from the TargetLink Environment (📖 TargetLink Interoperation and Exchange Guide)

References

Export Hook Scripts (📖 TargetLink File Reference)
File Export Utility (📖 TargetLink Tool and Utility Reference)

tl_export_container

tl_export_container

Purpose

Exports a container for exchanging AUTOSAR data with SystemDesk

Description

This function lets you export a container from the Data Dictionary. The container consists of production code files, AUTOSAR files, and A2L files.

Syntax

```
tl_export_container(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Subsystems`	Name of one or more subsystems TargetLink exports containers for. If you call the API command without parameters, TargetLink exports containers for all subsystems of currently open DD project file.
`Batchmode`	If set to 'off', Export Container dialog is displayed. Default: 'on'

Property	Description
`BuildObject`	Data Dictionary paths of the build object created during the build process that the object files are to be exported from. Default: ''
`WorkflowDefinitionFile`	File path to a workflow definition file that should be used for the container export. If no workflow definition file is specified, the standard workflow definition file is used for the export.

Example

```
% export container for the Controller subsystem
tl_export_container('Subsystems','Controller');
if ds_error_check
disp('Container export failed.');
end

% export files for the all subsystems
tl_export_container;
if ds_error_check
disp('Container export failed.');
end
```

Related topics

Basics

Interoperating with SystemDesk via SWC Containers (TargetLink Interoperation and Exchange Guide)

tl_pack_model

tl_pack_model

Purpose

Bundles all files required to work with TargetLink model in a non-TargetLink environment

Syntax

```
bSuccess = tl_pack_model(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`model`	Model to be packed
`newfolder`	Specifies/creates folder to store extracted files in (destination folder)
`newdir`	Parent folder of destination folder
`removetlframe`	Indicates whether simulation frames are stripped from TargetLink subsystems. Default: 'on'
`copysfcn`	Indicates whether to include all S-function files, mandatory for blocks such as Custom Code blocks type I. Default: 'on'
`createzipfile`	Indicates whether destination folder is zipped Default: 'on'

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	True on success, false on error

Example

```
% The following command extracts all the files from the pipt1
% TargetLink model into pureSL_files.zip. The file resides in
% the same folder as the pipt1 model.
tl_pack_model('model','pipt1');
```

Related topics

References

Pack Model Dialog (TargetLink Tool and Utility Reference)
tl_clear_system.. 50

Generating and Updating Models From the DD

Where to go from here

Information in this section

tl_generate_swc_model

tl_generate_swc_model

Purpose

Generates/updates TargetLink subsystem from description of software components.

Description

This function creates/updates one or more TargetLink subsystems according to the software components defined in the DD. If you specify AUTOSAR files as input, all the software component descriptions contained in these files are imported to the DD. Existing models are updated by default.

Syntax

```
tl_generate_swc_model(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of model to which TargetLink subsystem is added/copied. If specified model does not exist, creates new model. Default: current model
`DestDirectory`	Path of directory where specified model resides/is to be created.Default: model directory if <Model> specifies a current model, working directory otherwise

Property	Description
`DestSubsystem`	Depends on value of UseOneTlSubsystemForAllSwcs parameter: ▪ 'On' - Simulink path of subsystem in specified model to be replaced by new TargetLink subsystem. If specified subsystem does not exist a new TargetLink subsystem is created on root level of the specified model. ▪ 'Off' - Simulink path of subsystem in model where new TargetLink subsystems are created. If specified subsystem does not exist, the new TargetLink subsystems are created on root level of specified model.
`TLSubsystemID`	ID(s) of created TargetLink subsystem(s). Default: unique ID in the model
`SwcDescFileNames`	List of AUTOSAR files containing description of the software components that subsystems are to be generated/updated for. The contents of the files are imported into Data Dictionary.
`DDFileName`	Name of DD file to import AUTOSAR files to or where software components are defined. The specified DD file must exist. This property is ignored if the UseCurrentDD property is set.
`UseCurrentDD`	Uses current DD file for TargetLink subsystem generation/update. Default: 'off'
`SoftwareComponents`	Names of the software components selected for subsystem generation/update. Default: all the software components specified in DD/AUTOSAR file(s)
`EnableUpdate`	Lets you specify whether TargetLink should update an existing subsystem: ▪ On - Updates existing subsystems. If no subsystem exists, creates a new one. ▪ Off - Creates new subsystems Default: 'on'
`ShowReport`	Lets you specify whether to show update report in MATLAB Web Browser: ▪ 'on' - Shows report ▪ 'off' - Does not show report Update report is named SwcMdlUpdat_<model>.html and placed in current working directory. Default: 'off'
`ForceOperationCallSubsystemUsage`	Lets you specify how synchronous operation calls are implemented. The following values are possible: ▪ 'on' - Always implements synchronous operation calls via a operation call subsystem. ▪ 'off' - Uses TargetLink InPorts/Bus Inports or TargetLink OutPorts/Bus Outports blocks to model synchronous operation calls, if possible. The usage of the TargetLink port blocks is not possible for: ▪ bidirectional operations ▪ operations with application error ▪ operations with transformer error In this cases TargetLink implements the operation by means of operation call subsystem. Default: 'off'

Property	Description
`TerminateInOutPorts`	Lets you specify if TargetLink InPort/Bus Inport or TargetLink OutPort/Bus Outport blocks specifying AUTOSAR communication are connected to the Simulink Ground or Simulink Terminator blocks ('on') or not ('off'). Default: 'on'
`AddOperationSubsystemTriggerPort`	Lets you specify if Trigger ports whose trigger type is set to 'function-call' are added to operation subsystems. ▪ On - Trigger ports are added. ▪ Off - Trigger ports are not added. For operation subsystems without arguments, Trigger Port blocks are always added. Default: 'off'
`AddDataSendPointTriggerPort`	Lets you specify if TargetLink OutPort/Bus OutPort blocks representing data send points are to be embedded into a function call subsystem or not. Default: 'off'
`AddRunnableTriggerPort`	Lets you specify whether FcnCall-Trigger Port blocks are placed within created runnable subsystems. Possible values are: ▪ 'Always' - Trigger Port block is added to the runnable subsystem ▪ 'Never' - Trigger Port block is not added to the runnable subsystem ▪ 'SWCDependent' - Trigger Port block is added to the runnable subsystem if the parented SWC contains multiple runnables, otherwise not. In case of a SWC subsystem update the existing runnable subsystems are also considered Default: 'SWCDependent'
`AddComSpecBlocks`	Lets you specify if TargetLink ReceiverComSpec and SenderComSpec blocks are to be added or not. Default: 'off'
`AddBlocksForDataElementUpdated`	Lets you specify whether TargetLink Data Store Memory and Data Store Read blocks to model the update flag for sender-receiver communication are to be added or not. Default 'on'.
`UseOneTlSubsystemForAllSwcs`	Lets you specify where SWCs are modeled: ▪ 'on' - Models software components in virtual subsystems of one TargetLink subsystem ▪ 'off' - Models each software component in separate TargetLink subsystems Default: 'on'
`GenerateStimuliSubsystems`	Lets you specify if TargetLink creates and connects subsystems with stimuli signals to the following ports: ▪ TargetLink subsystem inports ▪ Runnable outports: i.e., TargetLink OutPort blocks configured for AUTOSAR communication ▪ TargetLink InPort and OutPort blocks of subsystems that implement operation calls Default: 'on'

Property	Description
`MergeRunnableStimuli`	Lets you specify whether to merge stimuli signals in one bus signal before connecting the signals to Runnable outports. However, TargetLink does not merge stimuli signals that are to be connected to TargetLink OutPort blocks configured for operation arguments or operation return values. Default: 'off'
`GenerateStimuliLib`	Lets you specify if TargetLink generates a library for stimuli subsystems for TargetLink subsystems. You can enable stimuli subsystem generation by using the GenerateStimuliSubsystems property. By default TargetLink uses <model>_stimuli_lib as the library name. You can use the StimuliLibName property to specify an adapted library name. TargetLink replaces existing library blocks if available. Default: 'off'
`StimuliLibName`	Name of library that you can generate using GenerateStimuliLib parameter. Default: <model>_stimuli_lib
`UseDataStoreBlocksForNvDataAccesses`	Lets you specify which blocks to use to model NVDataAccess points: ▪ 'On' - DataStoreRead or DataStoreWrite blocks are used. ▪ 'Off' - TL_[Bus]Inport or TL_[Bus]Outport blocks are used. Default: 'off'
`UseDataStoreBlocksForInterRunnableVariables`	Lets you specify which blocks to use to model interrunnable variables: ▪ 'On' - DataStoreRead or DataStoreWrite blocks are used. ▪ 'Off' - TL_[Bus]Inport or TL_[Bus]Outport blocks are used. Default: 'off'
`UseDataStoreBlocksForSenderReceiverAccesses`	Lets you specify which blocks to use to model SenderReceiverAccess points: ▪ 'On' - DataStoreRead or DataStoreWrite blocks are used. ▪ 'Off' - TL_[Bus]Inport or TL_[Bus]Outport blocks are used. Default: 'off'
`ImplicitNvDataAccessKind`	Lets you specify the kind of implicit write accesses to NvData: ▪ IWriteRef - Implicit (IWriteRef) access is modelled. ▪ IWrite - Implicit (IWrite) access is modelled. Default: 'IWriteRef'
`ImplicitSenderReceiverAccessKind`	Lets you specify the kind of implicit write accesses to sender/receiver data elements: ▪ IWriteRef - Implicit (IWriteRef) access is modelled. ▪ IWrite - Implicit (IWrite) access is modelled. Default: 'IWrite'
`DataFileName`	Name of the MAT file to which the Simulink.Bus objects created in the MATLAB Base Workspace are saved, if applicable. If such a file does not exist, it is created. Otherwise, the Simulink.Bus objects are added to the existing file. Default: <model>_Data.mat
`UseBusCreatorBlocksForStimuliGeneration`	Generates several constant blocks and bus creator blocks instead of a bus capable constant block for stimuli. Default: 'off'

Example

```
% Generate the Controller TargetLink subsystem in the MyModel model from the
% following AUTOSAR software description file: controller_swc.arxml.
tl_generate_swc_model('Model','MyModel','DestSubsystem','MyModel/Controller',...
'SoftwareComponents','controller',...
'SwcDescFileName','controller_swc.arxml');
```

Remarks

You can specify property default values of your own in the Data Dictionary at /Pool/Autosar/Config/FrameModelGeneration.

Only selected properties have predefined default values in the Data Dictionary. Add further properties as required.

Related topics

Basics

Generating/Updating a Frame Model from Classic AUTOSAR Data (TargetLink Classic AUTOSAR Modeling Guide)

tlSyncSystemSignature

tlSyncSystemSignature

Purpose

Synchronizes a specified Data Dictionary Signature or Block object with a specified Simulink system.

Description

Depending on selected synchronization mode, the function is able to perform different tasks.

If SyncMode is set to 'DD2Model' and the ProcessExistingSystem property is set to 'Update', the function updates the signature of existing Simulink systems according to the specified DD Signature object. During the update, new TargetLink Bus Inport/InPort and Bus Outport/OutPort blocks are added to the system, and selected properties of the existing blocks, e.g., the name or DD references, are modified. Superfluous blocks are not deleted, but only reported as 'to be deleted'.

You can specify to update the system directly or indirectly, by means of a parent system. In the latter case, TargetLink performs an update for all the systems that belong to the parent system's hierarchy and are associated with the specified DD

Signature object. A Simulink system is associated with a DD Signature object if it contains a TargetLink Function block that references this DD Signature object. The DD Signature object can also be specified directly or indirectly. In case of indirect specification, you can specify a Block object with BlockType = TL_Function that references the required DD Signature object.

If the system to be updated does not exist, the function creates a new one.

The results of the 'DD2Model' synchronization are included in the HTML report.

If SyncMode is set to 'ConsistencyCheck', the function checks if the signature of the existing Simulink systems agrees with the specified DD Signature object. However, it does not perform any system modifications. The system and DD Signature object can be specified in the same manner as described for the 'DD2Model' synchronization mode.

The results of the consistency check are included in the HTML report.

If SyncMode is set to 'Model2DD', the function creates an initial system signature specification below the specified DD Signature object of a specific Simulink system. This means it adds a relevant DD SignaturePort object for each TargetLink Bus Inport/InPort and Bus Outport/OutPort block residing at the root level of the system. The system must be associated with the DD Signature object that the system signature specification is to be created for.

Syntax

```
[msgStruct, consistencyCheckResult] = tlSyncSystemSignature(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`DDSignature`	Data Dictionary identifier (path or handle) of DD Signature object
`DDFunctionBlock`	Data Dictionary identifier (path or handle) of the Block object with BlockType = TL_Function.
`SyncMode`	Synchronization mode. The following values are possible: ▪ 'DD2Mode' - Updates or generates a Simulink system according to the specified DD Signature or DD Function Block object. ▪ 'Model2DD' - Creates a signature description in the specified DD Signature object according to the signature specified in the Simulink system. ▪ 'ConsistencyCheck' - Checks the consistency of the signatures between a specified DD Signature object and Simulink system. Default: 'DD2Model'

Property	Description
`ParentSystem`	Simulink path of the parent system where the system(s) is to be updated/checked reside resp. where the new system is to be created. The parent system must exist, with one exception: The system to be created should be a TargetLink subsystem and a parent system specifies a model name. In this case the parent model will be open, if it exists, or created.
`SystemName`	Name of the system to be created. Default: Name derived from the NameTemplate property of the Signature object, if applicable. Otherwise: Subsystem
`SystemPath`	Simulink system path. Denotes the following, depending on the selected synchronization: ▪ The system to update. ▪ The system whose signature consistency to check. ▪ The system whose signature description to create below the DD Signature object.
`SystemType`	Type of the system to be created. The following values are possible: ▪ 'Subsystem' - Simulink subsystem. ▪ 'TLSubsystem' - TargetLink subsystem. ▪ 'Model' - Referenced model and Model Reference block in the parent system. Default: 'Subsystem'
`ProcessExistingSystem`	This option is taken into account only for SyncMode = 'DD2Model'. It specifies the behavior in case there is already system like the specified one. The following values are possible: ▪ 'Abort' - Synchronization is aborted. ▪ 'Overwrite' - The existing system is overwritten. ▪ 'Autorename' - A new system with an unambiguous name is created. ▪ 'Update' - The system is updated. Default: 'Update'
`GenerateStimuliSubsystems`	This option is taken into account only for SystemType = 'TLSubsystem'. It specifies if subsystems with stimulus signals are to be connected to the following ports: ▪ TargetLink subsystem inports ▪ TargetLink subsystem outports Default: 'on'
`ReportName`	Name of the HTML report to be generated, without extension. Default: SyncReport_<DDSignatureObjectName>, if the DD Signature object is known, SyncReport - otherwise.
`ShowReport`	Specifies if the generated HTML report is to be open or not. The following values are possible: ▪ 'on' - The HTML report is open. ▪ 'off' - The HTML report is not open. Default: 'on'

Property	Description
`EnableTLBlockTypeModification`	Specifies if the type of the existing TargetLink block can be modified during system signature update. For example: ▪ TL_InPort/TL_OutPort to TL_BusInport/TL_BusOutport ▪ TL_BusInport/TL_BusOutport to TL_InPort/TL_OutPort ▪ TL_BusInport/TL_BusOutport to TL_BusInport/TL_BusOutport with a different number of leaf bus elements If 'off', the block type does not change, and only information about this inconsistency is shown in the HTML report. Modifying the block type is possible only if the SynchronizationCriterion property of the corresponding DD SignaturePort object is set to 'ByPortName'. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Struct with messages created during call to this function. If it is called without an output argument, the messages are printed in the MATLAB Command Window and shown in the TargetLink Message Browser.
`consistencyCheckResult`	Result of the consistency check between the signatures of the DD Signature object and corresponding Simulink system. Returned as a vector of structures with following fields: ▪ `systemPath` - Path of the Simulink system. 'Unknown' if the system is unknown. ▪ `checkResult` - Result of the consistency check for the Simulink system. For the 'checkResult' field, the following values are possible: ▪ `Unknown` - Result unknown. The check could not be completed. ▪ `Consistent` - The signatures of the DD Signature object and the Simulink system are identical. ▪ `Inconsistent` - The signatures of the DD Signature object and the Simulink system differ.

Example

```
% Update a system MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem/MyFcnSubsystem
% associated with the following DD Signature object: /Pool/ModelDesign/Signatures/MySignature.
tlSyncSystemSignature('SyncMode','DD2Model',...
'DDSignature','/Pool/ModelDesign/Signatures/MySignature',...
'SystemPath','MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem/MyFcnSubsystem');


% Create a new model MyModel with TLSubsystem MyTLSubsystem from the following DD Signature object:
% /Pool/ModelDesign/Signatures/MySignature.
tlSyncSystemSignature('SyncMode','DD2Model',...
'DDSignature','/Pool/ModelDesign/Signatures/MySignature',...
'ParentSystem','MyModel',...
'SystemName','MyTLSubsystem',...
'SystemType','TLSubsystem');


% Create a new subsystem MySystem within the parent system
% MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem
% from the Block object with BlockType=TL_Function with the DD handle 256.
% The Block object is linked with the DD Signature object named MySystem
tlSyncSystemSignature('DDFunctionBlock',256,...
'ParentSystem','MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem');


% Check system signature consistency of all subsystems in MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem system
% that are associated with the following DD Signature object: /Pool/ModelDesign/Signatures/MySignature DD Signature
object.
tlSyncSystemSignature('SyncMode','ConistencyCheck',...
'DDSignature','/Pool/ModelDesign/Signatures/MySignature',...
'ParentSystem','MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem')


% Create a signature description in the /Pool/ModelDesign/Signatures/MySignature object
% from the following subsystem: MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem/MyFcnSubsystem
tlSyncSystemSignature('SyncMode','Model2DD',...
'DDSignature','/Pool/ModelDesign/Signatures/MySignature',...
'SystemPath','MyModel/MyTLSubsystem/Subsystem/MyTLSubsystem/MyFcnSubsystem');
```

tlTransformerError

Where to go from here

Information in this section

tlTransformerError

Purpose

provides functionality helpful for simulation of AUTOSAR transformer errors

Syntax overview

The following syntaxes are available:

```
tlTransformerError('CreateSimulinkBusObject', propertyName, propertyValue, ...)
```

Creates a Simulink.Bus object defining a bus needed for transformer error simulation.
The following rule determines where this object is created:

- No model -> MATLAB Base Workspace
- Model without Simulink Data Dictionary -> MATLAB Base Workspace
- Model with Simulink Data Dictionary -> model's Simulink Data Dictionary

If a file is specified, the bus definition is exported to it.

```
tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...)
```

Creates Simulink.Signal objects with the specified names as required for transformer error simulation. A suitable Simulink.Bus object is also created.
The following rule determines where these objects are created:

- No model -> MATLAB Base Workspace
- Model without Simulink Data Dictionary -> MATLAB Base Workspace
- Model with Simulink Data Dictionary -> model's Simulink Data Dictionary

If a file is specified, the definitions of the Simulink objects are exported to it.

`tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)`

Adds stimuli to the specified model for each specified Simulink.Signal object as needed for transformer error simulation. The stimuli of single Simulink.Signal objects are composed of Constant, Bus Creator and Data Store Write blocks. The Simulink.Signal objects are not created. To create them, call the CreateSimulinkSignalObjects command.
If no model is specified, the stimulus blocks are added to the newly created model and can be copied to the final destination model, for example.

`tlTransformerError('PrepareModelForSimulation', propertyName, propertyValue, ...)`

Prepares the transformer error simulation for the specified AUTOSAR software components. This command performs the following steps:

1. Obtaining the names of the Simulink.Signal objects associated with the transformer errors to be simulated from the Data Dictionary.
2. Creating the required Simulink.Bus and Simulink.Signal objects.
3. Adding suitable stimulus blocks to the specified model that are needed to stimulate different values of the transformer errors.

The names of the Simulink.Signal objects are taken from the TransformerErrorSignalLabel property set at the DD ComSpec objects below the DD ReceiverPort or DD SenderPort object whose ErrorHandling property is set to 'TransformerErrorHandling'.

Related topics

Basics

Basics on Data Transformation (📖 TargetLink Classic AUTOSAR Modeling Guide)
Basics on Simulating Classic-AUTOSAR-Compliant SWCs (📖 TargetLink Classic AUTOSAR Modeling Guide)

HowTos

How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication (📖 TargetLink Classic AUTOSAR Modeling Guide)

References

tlTransformerError('CreateSimulinkBusObject', propertyName, propertyValue, ...)................... 199
tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...)............. 201
tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)........................ 202
tlTransformerError('PrepareModelForSimulation', propertyName, propertyValue, ...)................ 203

tlTransformerError('CreateSimulinkBusObject', propertyName, propertyValue, ...)

Purpose

Creates a Simulink.Bus object defining a bus needed for transformer error simulation.

The following rule determines where this object is created:

- No model -> MATLAB Base Workspace
- Model without Simulink Data Dictionary -> MATLAB Base Workspace
- Model with Simulink Data Dictionary -> model's Simulink Data Dictionary

If a file is specified, the bus definition is exported to it.

Description

This command is part of the tlTransformerError function.

Syntax

```
tlTransformerError('CreateSimulinkBusObject', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'BusObjectName'`	Name of the Simulink.Bus object. Default: 'tlTransformerErrorBus'
`'FileName'`	Name of the file to which to export the definitions of the Simulink.Bus or Simulink.Signal objects. Default: -/-
`'ModelName'`	Model name. Default: current model

Related topics

Basics

Basics on Data Transformation (📖 TargetLink Classic AUTOSAR Modeling Guide)
Basics on Simulating Classic-AUTOSAR-Compliant SWCs (📖 TargetLink Classic AUTOSAR Modeling Guide)

HowTos

How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication (📖 TargetLink Classic AUTOSAR Modeling Guide)

References

tlTransformerError.. 198
tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...).............. 201
tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)......................... 202
tlTransformerError('PrepareModelForSimulation', propertyName, propertyValue, ...)................ 203

tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...)

Purpose

Creates Simulink.Signal objects with the specified names as required for transformer error simulation. A suitable Simulink.Bus object is also created.

The following rule determines where these objects are created:

- No model -> MATLAB Base Workspace
- Model without Simulink Data Dictionary -> MATLAB Base Workspace
- Model with Simulink Data Dictionary -> model's Simulink Data Dictionary

If a file is specified, the definitions of the Simulink objects are exported to it.

Description

This command is part of the tlTransformerError function.

Syntax

```
tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'SignalObjectNames'`	Name of the Simulink.Signal objects. Default: -/-
`'FileName'`	Name of the file to which to export the definitions of the Simulink.Bus or Simulink.Signal objects. Default: -/-
`'BusObjectName'`	Name of the Simulink.Bus object. Default: 'tlTransformerErrorBus'
`'ModelName'`	Model name. Default: current model
`'InitValueAsStruct'`	If 'on', the created Simulink.Signal objects are initialized with a structure with the components 'errorCode' and 'transformerClass'. If 'off', the init value is a plain value. A structured init value is required if the 'Underspecified initialization detection' model parameter of the model where the Simulink.Signal objects are referenced is set to 'Simplify'. If this parameter is set to 'Classic', a plain init value is needed. Default: 'on', if model is not specified. Otherwise, model-dependent.

Related topics

Basics

Basics on Data Transformation (📖 TargetLink Classic AUTOSAR Modeling Guide)
Basics on Simulating Classic-AUTOSAR-Compliant SWCs (📖 TargetLink Classic AUTOSAR Modeling Guide)

HowTos

How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication (📖 TargetLink Classic AUTOSAR Modeling Guide)

References

tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)

Purpose

Adds stimuli to the specified model for each specified Simulink.Signal object as needed for transformer error simulation. The stimuli of single Simulink.Signal objects are composed of Constant, Bus Creator and Data Store Write blocks. The Simulink.Signal objects are not created. To create them, call the CreateSimulinkSignalObjects command.

If no model is specified, the stimulus blocks are added to the newly created model and can be copied to the final destination model, for example.

Description

This command is part of the tlTransformerError function.

Syntax

```
tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'ModelName'`	Model name. Default: current model

Property	Description
`'SignalObjectNames'`	Name of the Simulink.Signal objects. Default: -/-
`'BusObjectName'`	Name of the Simulink.Bus object. Default: 'tlTransformerErrorBus'

Related topics

Basics

Basics on Data Transformation (📖 TargetLink Classic AUTOSAR Modeling Guide)
Basics on Simulating Classic-AUTOSAR-Compliant SWCs (📖 TargetLink Classic AUTOSAR Modeling Guide)

HowTos

How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication (📖 TargetLink Classic AUTOSAR Modeling Guide)

References

tlTransformerError('PrepareModelForSimulation', propertyName, propertyValue, ...)

Purpose

Prepares the transformer error simulation for the specified AUTOSAR software components. This command performs the following steps:

1. Obtaining the names of the Simulink.Signal objects associated with the transformer errors to be simulated from the Data Dictionary.
2. Creating the required Simulink.Bus and Simulink.Signal objects.
3. Adding suitable stimulus blocks to the specified model that are needed to stimulate different values of the transformer errors.

The names of the Simulink.Signal objects are taken from the TransformerErrorSignalLabel property set at the DD ComSpec objects below the DD ReceiverPort or DD SenderPort object whose ErrorHandling property is set to 'TransformerErrorHandling'.

Description

This command is part of the tlTransformerError function.

Syntax

```
tlTransformerError('PrepareModelForSimulation', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'SoftwareComponents'`	Names of the SoftwareComponent DD object. In case the affected SoftwareComponent DD object does not reside directly below /Pool/Autosar/SoftwareComponents objects, the DD path relative to the /Pool/Autosar/SoftwareComponents must be specified, for example MySoftwareComponentGroup/MySoftwareComponent
`'ModelName'`	Model name. Default: current model
`'FileName'`	Name of the file to which to export the definitions of the Simulink.Bus or Simulink.Signal objects. Default: -/-
`'BusObjectName'`	Name of the Simulink.Bus object. Default: 'tlTransformerErrorBus'

Related topics

Basics

Basics on Data Transformation (📖 TargetLink Classic AUTOSAR Modeling Guide)
Basics on Simulating Classic-AUTOSAR-Compliant SWCs (📖 TargetLink Classic AUTOSAR Modeling Guide)

HowTos

How to Model and Simulate Transformer Error Logic in Sender-Receiver Communication (📖 TargetLink Classic AUTOSAR Modeling Guide)

References

tlTransformerError.. 198
tlTransformerError('CreateSimulinkBusObject', propertyName, propertyValue, ...).................. 199
tlTransformerError('CreateSimulinkSignalObjects', propertyName, propertyValue, ...).............. 201
tlTransformerError('CreateStimulusBlocks', propertyName, propertyValue, ...)........................ 202

Generating Custom Code TLC Files

tl_generate_customcode_tlc

tl_generate_customcode_tlc

Purpose

Builds TLC scripts for TargetLink Custom Code blocks.

Syntax

```
bSuccess = tl_generate_customcode_tlc(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`block`	List of Custom Code block(s) whose script file(s) are to be generated

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	True on success, false on error

Example

```
% build a custom code TLC script for the CC Block block in the controller subsystem
tl_generate_customcode_tlc('Block','mymodel/controller/CC block');
```

Related topics

Basics

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

References

Custom Code Block (📖 TargetLink Model Element Reference)

Generating Functional Mock-Up Units

tl_generate_fmu

tl_generate_fmu

Purpose

Generates a FMU for the specified TargetLink subsystem.

Description

This function creates a FMU container for each of the specified TargetLink subsystems. The FMU container is packed into a zip file <TLSubsystemName>.fmu. Because the code generated for the specified TargetLink subsystem is a part of the FMU container, this function requires that the code has already been generated. If desired, the user can specify Data Dictionary CodeGenerationUnit objects whose code is to be part of the FMU container. By default TargetLink obtains the Data Dictionary CodeGenerationUnit object automatically.

Syntax

```
[bError, msgData] = tl_generate_fmu(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of model containing TargetLink subsystems that FMU containers are to be generated for.
`TLSubsystems`	Names of the TargetLink subsystems in the specified model that FMU containers are to be generated for. Default: All TL subsystems in the model
`DDCodeGenerationUnits`	Names of the DD CodeGenerationUnits objects whose code is to be added to the FMU containers generated for specified TargetLink subsystems. If set to 'auto', TargetLink automatically determines whether code generation for DD CodeGenerationUnit objects is required by the FMU; if code has not been generated, TargetLink takes stub code and informs you.

Property	Description
`UUID`	List of UUIDs for FMU containers generated for specified TargetLink subsystems. The number and order of elements in the list must match the number and order of the specified TargetLink subsystems. After a successful FMU container generation, TargetLink stores the UUIDs you specified in the TargetLink subsystem the FMU container was generated from. Make sure to save the model containing the TargetLink subsystems you generated an FMU container for. Default: UUIDs associated with TargetLink subsystem, if any. Otherwise, UUIDs generated by TargetLink.
`FmuContainerDir`	Directory to generate the FMU container in. Default: '.\TLFMU'
`GenerateGlobalSymbols`	If 'on', generates a C module named tl_globalsdefs_generated.c and adds it to the FMU container. The module contains definitions of interface variables specified as extern global and therefore not defined in the generated code. To avoid double symbol definitions of interface variables made by the Addfile block, this option must be set to 'off'. Default: 'off'
`IncludeSystemFiles`	If 'on', TargetLink copies the Fixed-Point Library sources to the FMU container, if required. Default: 'on'
`IncludeSourceFiles`	If 'on', TargetLink includes source and header files in the container. Default: 'on'
`IncludeBinaryFiles`	If 'on', TargetLink includes Windows DLLs (32-bit and 64-bit) in an FMU container that are built from the C source and header files in the container. If desired, the Linux shared library (64-Bit) is also built. In this case the location of the GCC Linux 64-Bit compiler must be specified in the X86_64_LINUX_GCC_ROOT environment variable. Default: 'on'
`SimConfigPackageDir`	If specified, platform specific header files contained in the FMU container as well as binaries, if applicable, are generated for the corresponding simulation platform. Otherwise (default) the generated platform specific header files are compatible: ▪ to all 32 and 64 bit platforms (Windows and Linux) if system files (fixed point library) are not included ▪ to all 32 and 64 bit platforms (Windows and Linux) with little endian byte order if system files (fixed point library) are included As a valid property value a board package directory must be specfied, e.g. <TL_ROOT>\Matlab\ApplicationBuild\BoardPackages\HostPC64\GCC

Output parameters

The following output parameters are available:

Parameter	Description
`bError`	false on success, else true
`msgData`	message struct (empty on success)

Example

```
% Generate FMU containers for all TargetLink subsystems contained in current model.
% The generated FMU are located in the directory .\TLFMU.
tl_generate_fmu;

% Generate FMU container for TargetLink subsystem 'throttle_control' in the model 'ECUmdl'.
% The FMU is to be generated into destination directory e:\MyFMUs.
tl_generate_fmu(...
'Model','ECUmdl',...
'TlSubsystems','throttle_control',...
'FmuContainerDir','e:\MyFMUs');
```

Generating Simulink Implementation Containers

tlGenerateSic

tlGenerateSic

Purpose

Generates a dSPACE Simulink implementation container using the TargetLink Code Generator.

Description

This function creates a Simulink implementation container for each of the specified TargetLink subsystems. The Simulink implementation container is packed into a ZIP file <TLSubsystemName>.sic. Because the code generated for the specified TargetLink subsystem is part of the container, this function requires that the code has already been generated. If desired, you can specify DD CodeGenerationUnit objects whose code is to be part of the container. By default, TargetLink gets the DD CodeGenerationUnit object automatically.

Syntax

```
tlGenerateSic(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	**Description**
`Model`	Name of the model containing TargetLink subsystems for which to generate Simulink implementation containers . Default: Current model
`TLSubsystems`	Names of the TargetLink subsystems in the specified model for which Simulink implementation containers are to be generated. Default: All TargetLink subsystems in the model.
`DDCodeGenerationUnits`	Names of the DD CodeGenerationUnits objects whose code is to be added to the Simulink implementation containers generated for specified TargetLink subsystems. If set to 'auto', TargetLink automatically determines whether code generation for DD CodeGenerationUnit objects is required by the Simulink implementation container. If no code has been generated, TargetLink uses stub code and informs you of this. Default: 'auto'

Property	Description
`ContainerDir`	Directory in which to generate the Simulation implementation container. Default: '.\TLSIC'
`IncludeSystemFiles`	The possible values are: ▪ `on` - TargetLink copies the Fixed-Point Library sources to the Simulation implementation container. (default) ▪ `off` - TargetLink does not copy the Fixed-Point Library sources to the Simulation implementation container.
`VariableDescriptionFileFormat`	Specifies the format of the variable description file to be included in the Simulink implementation container. Possible values are: ▪ `TRC` - A TRC file is included; for SICs used in ConfigurationDesk (default) ▪ `A2L` - An A2L file is included; for SICs used in VEOS Player and ConfigurationDesk.
`ShowModelHierarchyInTrcFile`	Applies only to VariableFileDescriptionFormat = 'TRC'. ▪ `on` - The Simulink system hierarchy of the TargetLink subsystem for which the Simulink implementation container was generated is shown in the TRC file. ▪ `off` - The generated function hierarchy is shown in the TRC file. (default)
`IncludeExternalVariablesInTRCFile`	Applies only to VariableFileDescriptionFormat = 'TRC'. ▪ `on` - TargetLink includes the description of global external variable into the TRC file (default) ▪ `off` - TargetLink does not include the description of global external variable into the TRC file. This option can be usefull to avoid ConfigurationDesk's conflict that is displayed if in two or more SICs that are assigned to the same application process identical global variable are described in the TRC file.
`IncludeStubCodeFileVariablesInTRCFile`	Applies only to VariableFileDescriptionFormat = 'TRC'. ▪ `on` - TargetLink includes the description of global variables that are definied in a stub code file into the TRC file (default) ▪ `off` - TargetLink does not include the description of global variables defined in a stub code file into the TRC file. This option can be usefull to avoid ConfigurationDesk's conflict that occurs if in two or more SICs that are assigned to the same application process identical global variable are described in the TRC file.

Property	Description
`GenerateGlobalSymbols`	Applies only to interface variables of the TargetLink subsystem's root functions that are not explicitly specified as Simulink implementation container interfaces, but as extern global variables. This option must be set to 'off' if the definition of these global variables is already provided by the Addfile block. Default: 'on'

Generating V-ECU Implementations

tl_generate_vecu_implementation

tl_generate_vecu_implementation

Purpose

Generates a V-ECU implementation for the specified TargetLink subsystem.

Description

This function creates a V-ECU package container for each of the specified TargetLink subsystems. The V-ECU package container is packed into a zip file <TLSubsystemName>.vecu. Because the code generated for the specified TargetLink subsystem is a part of the V-ECU implementation container, this function requires that the code has already been generated. If desired, the user can specify Data Dictionary CodeGenerationUnit objects whose code is to be part of the V-ECU implementation container. By default TargetLink obtains the Data Dictionary CodeGenerationUnit object automatically.

Syntax

```
tl_generate_vecu_implementation(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of model containing TargetLink subsystems that V-ECU implementation containers are to be generated for. Default: Current model
`TLSubsystems`	Names of TargetLink subsystems residing in specified model for which V-ECU implementation containers are to be generated. Default: All TargetLink subsystems in the model.
`DDCodeGenerationUnits`	Names of the DD CodeGenerationUnits objects whose code is to be added to the V-ECU implementation containers generated for specified TargetLink subsystems. If set to 'auto', TargetLink automatically determines whether code generation for DD CodeGenerationUnit objects is required by V-ECU implementation; if code has not been generated, TargetLink takes stub code and informs you.

Property	Description
`UUID`	List of UUID for V-ECU implementation containers generated for specified TargetLink subsystems. Number and order of elements in list must match the number and order of specified TargetLink subsystems. After a successful V-ECU implementation container generation, TargetLink stores the UUID you specified in the TargetLink subsystem the V-ECU implementation container was generated from. Make sure to save the model containing the TargetLink subsystem(s) you generated a V-ECU implementation container for. Default: UUID associated with TargetLink subsystem, if any. Otherwise, UUID generated by TargetLink.
`VEcuContainerDir`	Directory where V-ECU implementation is to be generated. Default: .\TLVECU
`GenerateGlobalSymbols`	If 'on', generates C-module named tl_globalsdefs_generated.c and adds it to the V-ECU implementation container. Module contains definitions of interface variables specified as extern global and therefore not defined in the generated code. To avoid double symbol definitions of interface variables made by the Addfile block, this option must be set to 'off'. Default: 'on'

Example

```
% Generate V-ECU implementation containers for all TargetLink subsystems
% contained in current model. The generated V-ECU implementations are located
% in the directory .\TLVECU
tl_generate_vecu_implementation;

% Generate V-ECU implementation container for TargetLink subsystem 'throttle_control'
% in the model 'ECUmdl'. The V-ECU implementation is to be
% generated into destination directory e:\MyVECUs.
tl_generate_vecu_implementation(...
'Model','ECUmdl',...
'TlSubsystems','throttle_control',...
'VEcuContainerDir','e:\MyVECUs');
```

Related topics

Basics

Basics on Interoperating with Other dSPACE Tools for Virtual Validation
(📖 TargetLink Interoperation and Exchange Guide)

Handling Errors and Messages

Where to go from here

Information in this section

ds_error_check

ds_error_check

Purpose

Returns number of messages of specified severity.

Syntax

```
numMsgs = ds_error_check(severity)
```

Input parameters

The following input parameters are available:

Parameter	Description
`severity`	▪ 'Fatals' - Returns number of fatals ▪ 'Errors' - Returns number of fatals and errors ▪ 'Warnings' - Returns number of fatals, errors, and warnings ▪ 'Advices' - Returns total number of fatals, errors, warnings and advices ▪ 'Notes' - Returns number of fatals, errors, warnings and notes ▪ 'OnlyFatals' - Returns number of fatals (same as 'Fatals') ▪ 'OnlyErrors' - Returns number of errors ▪ 'OnlyWarnings' - Returns number of warnings ▪ 'OnlyAdvices' - Returns number of advices ▪ 'OnlyNotes' - Returns number of notes Default: 'Errors'

Output parameters

The following output parameters are available:

Parameter	Description
`numMsgs`	Number of messages with specified severity

Example

```
% Call utility M file, which might set some error conditions.
tl_build_target('Model','my_model');
if ds_error_check
disp('There was an error while executing TL_BUILD_TARGET');
end
```

Remarks

If the result is zero, all previous commands since the last ds_error_clear were executed successfully.

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_clear

ds_error_clear

Purpose

Clears messages in TargetLink's message system.

Description

This function clears messages in TargetLink's message system. By default, all messages are cleared. DD messages are left untouched. To clear both TargetLink and DD messages, use ds_error_none.

Syntax

```
ds_error_clear(n)
```

Input parameters

The following input parameters are available:

Parameter	Description
n	Indices of TargetLink messages that should be cleared (default: all messages)

Example

```
This example shows you how to clear error message #3.
ds_error_clear(3);
```

Remarks

If you want to clear only a specific message, the index of the corresponding error message (see ds_error_msg) can be passed to the function as a parameter.

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_display

ds_error_display

Purpose

Displays messages.

Description

This function displays messages, or shows them in the Message Browser.

If neither 'ShowDialog' nor 'PrintMessage' is 'on', the messages' status remains unconfirmed.

Syntax

```
msgIdx = ds_error_display(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`ShowDialog`	Opens Message Browser if there are unconfirmed messages. In batch mode, this is the same as 'PrintMessage'. Default: 'on'
`PrintMessage`	Prints messages in MATLAB Command Window. Default: 'on'
`ClearMessage`	Clears displayed messages from TargetLink's message system. Default: 'on'

Output parameters

The following output parameters are available:

Parameter	Description
`msgIdx`	Indices of confirmed or displayed messages

Remarks

The return value msgIdx is a vector of message indices that were displayed. If there are no unconfirmed messages, the function returns immediately and msgIdx is an empty matrix.

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get

Where to go from here

Information in this section

ds_error_get

Purpose

Returns parameters of TargetLink's message system.

Syntax overview

The following syntaxes are available:

```
batchMode = ds_error_get('BatchMode')
```

Returns TargetLink's batch mode

`msgStruct = ds_error_get('Message', msgNum)`

Returns msgNum-th message from TargetLink's message system

`batchModePrintMessage = ds_error_get('BatchModePrintMessage')`

Returns TargetLink's batch mode print status

`groupedIndices = ds_error_get('GroupedIndices')`

Returns indices of registered messages grouped by type

`messageNumbers = ds_error_get('DefaultExcludedMessages')`

Returns the message numbers which are initially excluded for display in the Message Browser

`msgData = ds_error_get('CurrentState')`

Returns all data of TargetLink's message system

`msgStruct = ds_error_get('AllMessages')`

Returns all messages in a message struct

`msgStruct = ds_error_get('EmptyMessageStruct')`

Returns an empty message struct (constructor function)

`ds_error_get('MessageStruct', propertyName, propertyValue, ...)`

Creates message struct and sets specified fields (constructor function)

Example

```
% print all messages
fori=1:ds_error_check
   msg =ds_error_get('Message',i);
fprintf('%s(%d), msg = %s\n',msg.mfile,msg.line,msg.msg);
end


% store the current error state
prevMsgData =ds_error_get('CurrentState');

% invoke user-defined tool which clears the message list and issues new error messages
my_tl_tool;

% display the new messages together with the previous messages
ds_error_merge(prevMsgData);
ds_error_display


% create message struct and set some fields
msg =ds_error_get('MessageStruct',...
'type','error',...
'title','Error in API Function',...
'msg','M-script aborted due to an error',...
'ObjectName','/pipt1/picontroller');

% register the message
ds_error_register(msg);
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('BatchMode')

Purpose

Returns TargetLink's batch mode

Description

This command is part of the ds_error_get function.

Syntax

```
batchMode = ds_error_get('BatchMode')
```

Output parameters

The following output parameters are available:

Parameter	Description
batchMode	If 'on', TargetLink is in batch

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('Message', msgNum)

Purpose

Returns msgNum-th message from TargetLink's message system

Description

This command is part of the ds_error_get function.

Syntax

```
msgStruct = ds_error_get('Message', msgNum)
```

Input parameters

The following input parameters are available:

Parameter	Description
`msgNum`	Index of message in TagetLink's message list

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Message struct, empty matrix if no messages(s) was/were returned

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('BatchModePrintMessage')

Purpose

Returns TargetLink's batch mode print status

Description

This command is part of the ds_error_get function.

Syntax

```
batchModePrintMessage = ds_error_get('BatchModePrintMessage')
```

Output parameters

The following output parameters are available:

Parameter	Description
`batchModePrintMessage`	If 'on', messages are printed to MATLAB Command Window in batch mode

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('GroupedIndices')

Purpose

Returns indices of registered messages grouped by type

Description

This command is part of the ds_error_get function.

Syntax

```
groupedIndices = ds_error_get('GroupedIndices')
```

Output parameters

The following output parameters are available:

Parameter	Description
`groupedIndices`	Struct with indices of messages grouped by type: ▪ .fatal - Indices of fatal error messages ▪ .error - Indices of error messages ▪ .warning - Indices of warning messages ▪ .advice - Indices of advices ▪ .note - Indices of notes

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('DefaultExcludedMessages')

Purpose

Returns the message numbers which are initially excluded for display in the Message Browser

Description

This command is part of the ds_error_get function.

Syntax

```
messageNumbers = ds_error_get('DefaultExcludedMessages')
```

Output parameters

The following output parameters are available:

Parameter	Description
`messageNumbers`	Message numbers

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('CurrentState')

Purpose

Returns all data of TargetLink's message system

Description

This command is part of the ds_error_get function.

Syntax

```
msgData = ds_error_get('CurrentState')
```

Output parameters

The following output parameters are available:

Parameter	Description
`msgData`	Struct with data of TargetLink's message system

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('AllMessages')

Purpose

Returns all messages in a message struct

Description

This command is part of the ds_error_get function.

Syntax

```
msgStruct = ds_error_get('AllMessages')
```

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Message struct, empty matrix if no messages(s) was/were returned

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('EmptyMessageStruct')

Purpose

Returns an empty message struct (constructor function)

Description

This command is part of the ds_error_get function.

Syntax

```
msgStruct = ds_error_get('EmptyMessageStruct')
```

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Message struct, empty matrix if no messages(s) was/were returned

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_get('MessageStruct', propertyName, propertyValue, ...)

Purpose

Creates message struct and sets specified fields (constructor function)

Description

This command is part of the ds_error_get function.

Syntax

```
ds_error_get('MessageStruct', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'type'`	Message type: 'fatal', 'error', 'warning', 'advice', or 'note'. Default: 'error'
`'number'`	Message number. Default: '0'
`'title'`	Message title. Default: ''
`'msg'`	Message. Default: ''
`'objectName'`	Name of Simulink block, Stateflow object, DD object, file or MATLAB variable related to message. Default: ''
`'objectHandle'`	Handle of Simulink block, Stateflow object or DD object related to message. Default: []
`'module'`	Name of the module (M file) which produced the message. Default: ''
`'fcn'`	Name of subfunction in module (M file). Default: ''
`'line'`	Code line number in the module (M file) which produced the message. Default: '-1'
`'clock'`	Date and time the message was produced with the date returned by MATLAB's now function. Default: 'now'
`'confirmed'`	If '1', user-confirmed message. Default: '0'

Property	Description
`'objectKind'`	Type of object related to message, as specified with objectName or objectHandle: ▪ 'slblock' - Simulink block ▪ 'sfobject' - Stateflow object ▪ 'ddobject' - Data Dictionary object ▪ 'file' - File name ▪ 'mxarray' - MATLAB variable Default: 'slblock'

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_log

ds_error_log

Purpose

Writes messages to the logfile.

Description

Writes messages of TargetLink's message system to the logfile.

Syntax

```
ds_error_log(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`MessageList`	Struct with messages to be written to logfile. Default: all error messages in TargetLink's message system

Property	Description
`LogFile`	Name of logfile. Default: ds_messages.log
`Append`	If 'on', appends messages to logfile. If 'off', creates/overwrites logfile. Default: 'off'
`DiagnosticInfo`	Writes additional information such as computer, operating system, TargetLink version, etc. to the logfile. Default: 'on'
`ShowErrors`	Writes fatals and errors to logfile. Default: 'on'
`ShowWarnings`	Writes warnings to logfile. Default: 'on'
`ShowAdvices`	Writes advices to logfile. Default: 'on'
`ShowNotes`	Writes notes to logfile. Default: 'on'
`Excludes`	Number of messages to omit from logfile. Default: []
`Sort`	▪ 'Occurrence' - Sorts messages by occurrence ▪ 'Number' - Sorts messages by number Default: ''

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_merge

ds_error_merge

Purpose

Merges messages.

Description

This function merges specified messages with previously stored messages. The updated message list is returned and can later be used as input for successive calls of ds_error_merge. The initial msgData struct can be obtained with ds_error_get('CurrentState').

Syntax

```
msgData = ds_error_merge(msgData)
```

Input parameters

The following input parameters are available:

Parameter	Description
msgData	Struct with message data to be merged

Output parameters

The following output parameters are available:

Parameter	Description
msgData	Returned message data

Example

```
% The following code shows how the msgData message list is created
% and afterwards updated twice with new messages. Finally, the
% Message Browser opens.
msgData =ds_error_get('CurrentState');
... % Some API commands called ds_error_none() and output new messages.
msgData =ds_error_merge(msgData);
... % Some API commands called ds_error_none() and output new messages.
msgData =ds_error_merge(msgData);
ds_error_display;
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_msg

ds_error_msg

Purpose

Displays a message, and/or registers it in TargetLink's message system.

Description

This function displays a message and/or registers it in TargetLink's message system. The message is displayed in a modal dialog that must be closed by the user.

Syntax

```
msgIdx = ds_error_msg(msg, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Title`	Message title displayed as title of dialog box. No title is displayed if Title = 'off'. Default: <MessageType> IN <caller> (all in capital letters)
`ShowDialog`	Displays message in the model dialog box and asks for user confirmation. If batch mode is activated (with ds_error_set('BatchMode','on')), then 'ShowDialog' behaves like 'PrintMessage'. Default: 'on'
`PrintMessage`	Prints message in MATLAB Command Window. Default: 'on'
`PrintStack`	Prints stack of calling M files in MATLAB Command window. Default: 'off'
`RegisterMessage`	Registers message in TargetLink's message system. Default: 'on'
`MessageType`	Specifies message severity: ▪ 'fatal' ▪ 'error' ▪ 'warning' ▪ 'advice' ▪ 'note' Default: 'error'
`MessageNumber`	Message number, identifies chapter in online help for TargetLink's messages Default: 0

Property	Description
`Module`	Name of module where error occurred. Default: ''
`ObjectName`	Name of object associated with message. If full path of Simulink block or Stateflow object, 'Open' and 'Show' buttons are enabled in message dialog. Default: ''
`ObjectHandle`	Handle of object associated with message. If handle of Simulink block or Stateflow object, 'Open' and 'Show' buttons are enabled in message dialog. Default: -1
`ObjectKind`	Specifies the type of the associated object: ▪ 'slblock' - Simulink block or system ▪ 'sfobject' - Stateflow object ▪ 'ddobject' - DD object ▪ 'mxarray' - MATLAB variable ▪ 'file' - File Default: 'slblock'

Input parameters

The following input parameters are available:

Parameter	Description
`msg`	Message string or cell array or strings

Output parameters

The following output parameters are available:

Parameter	Description
`msgIdx`	Index in TargetLink's message system message list. If the message was not registered, msgIdx is an empty matrix.

Example

```
ds_error_msg('my error message')
ds_error_msg({'msg_line1''msg_line2''msg_lineN'});% multiple lines
ds_error_msg('my error message','ShowDialog','off');% do not display dialog
ds_error_msg('my warning message',...
'Title','Configuration Problem',...
'PrintStack','on',...
'MessageType','warning');
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_none

ds_error_none

Purpose

Clears messages.

Description

This function clears messages from TargetLink's message system, including Data Dictionary messages. This function can be invoked before a tool starts, to clear messages from previous code generation runs.

Syntax

```
numMsgs = ds_error_none(ClearDDMessageList)
```

Input parameters

The following input parameters are available:

Parameter	Description
`ClearDDMessageList`	If '0', DD messages remain in TargetLink's message system. Default: '1'

Output parameters

The following output parameters are available:

Parameter	Description
`numMsgs`	Number of messages in TargetLink's message system that were cleared

Example

```
% clear messages in TargetLink's message system, and all DD messages
ds_error_none;

% clear messages in TargetLink's message system, but leave DD messages
ds_error_none(0);
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_register

ds_error_register

Purpose

Registers message in TargetLink's message system.

Syntax overview

The following syntaxes are available:

```
msgIdx = ds_error_register(msg, propertyName, propertyValue, ...)
```

Register a message

```
msgIdx = ds_error_register(msgStruct)
```

Register a message structure

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Title`	Message title. If Title = 'off', no title is displayed. Default: <MessageType> IN <caller> (all in capital letters).

Property	Description
`PrintStack`	Prints a stack of the calling M files in the MATLAB Command Window. Default: 'off'
`MessageType`	Specifies message severity: ▪ 'fatal' ▪ 'error' ▪ 'warning' ▪ 'advice' ▪ 'note' Default: 'error'
`MessageNumber`	Message number, identifies chapter in online help for TargetLink's messages. Default: 0
`Module`	Name of module where the error occurred. Default: ''
`ObjectName`	Name of the object associated with the message. If it is the full path of a Simulink block or Stateflow object, the Open and Show buttons are enabled in the message dialog. Default: ''
`ObjectHandle`	Handle of object associated with the message. If it is the handle of a Simulink block or Stateflow object, the Open and Show buttons are enabled in the message dialog. Default: -1
`ObjectKind`	Specifies the type of the associated object: ▪ 'slblock' - Simulink block or system ▪ 'sfobject' - Stateflow object ▪ 'ddobject' - DD object ▪ 'mxarray' - MATLAB variable ▪ 'file' - File Default: 'slblock'

Input parameters

The following input parameters are available:

Parameter	Description
`msg`	Message as character vector or cell array of character vectors. A cell array of character vectors produces a multiline message.

Parameter	Description
`msgStruct`	Message struct that specifies messages to be registered: ▪ .type - Message type ▪ .number - Message number ▪ .title - Message title ▪ .msg - Message text ▪ .objectName - Simulink object related to message ▪ .objectHandle - Handle of Simulink object ▪ .module - Name of module (M file) where message occurred ▪ .fcn - Name of subfunction in module (M file) ▪ .line - Line in module (M file) where message occurred ▪ .clock - Date and time when message occurred ▪ .confirmed - If 1, user-confirmed message ▪ .objectKind - Type of associated object

Output parameters

The following output parameters are available:

Parameter	Description
`msgIdx`	Index in TargetLink's message system message list. Empty matrix if message not registered.

Example

```
ds_error_register('my error message')
ds_error_register({'msg_line1''msg_line2''msg_lineN'});% multiple lines
ds_error_register('my warning message',...
'Title','Configuration Problem',...
'PrintStack','on',...
'MessageType','warning');

% invoke tool which produces a message struct, and register it
[~, msgStruct]=tl_check_system('system','myModel/Subsystem');
ds_error_register(msgStruct);
ds_error_display;
```

Remarks

If both the 'ObjectName' and the 'Objecthandle' properties are specified, they must reference the same object.

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_error_set

ds_error_set

Purpose

Sets parameters of TargetLink's message system.

Syntax

```
ds_error_set(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`BatchMode`	Switches batch mode. If 'on', no user interaction is required. Can be used to invoke TargetLink tools in scripts. Default: 'off'
`BatchModePrintMessage`	Prints messages in the MATLAB Command Window in batch mode. Default: 'on'
`DefaultExcludedMessages`	Excludes messages with specified numbers from Message Browser. Default: []
`RestoreState`	Replaces current messages with messages in specified struct. Input argument must be a valid message struct as returned by ds_error_get('CurrentState'). Default: []

Example

```
% set TargetLink into batch mode
ds_error_set('BatchMode','on');
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg

Where to go from here

Information in this section

ds_msgdlg

Purpose

Manages the TargetLink Message Browser.

Syntax overview

The following syntaxes are available:

`dlgFig = ds_msgdlg('show')`

Shows Message Browser

`dlgFig = ds_msgdlg('update', propertyName, propertyValue, ...)`

Creates/updates Message Browser with new list of messages, clears previous display

`dlgFig = ds_msgdlg('clear')`

Clears messages in Message Browser

`dlgFig = ds_msgdlg('close')`

Hides Message Browser

`ds_msgdlg('delete')`

Deletes Message Browser

`dlgFig = ds_msgdlg('find')`

Finds Message Browser

`dlgFig = ds_msgdlg('callback', propertyName, propertyValue, ...)`

Invokes control callback

Example

```
[tmp,msgStruct]=tl_prepare_system('system','myModel/controller');
ds_error_register(msgStruct);
ds_msgdlg('update');
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg('show')

Purpose

Shows Message Browser

Description

This command is part of the ds_msgdlg function.

Syntax

```
dlgFig = ds_msgdlg('show')
```

Output parameters

The following output parameters are available:

Parameter	Description
dlgFig	Handle of Message Browser, empty matrix if no Message Browser exists

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg('update', propertyName, propertyValue, ...)

Purpose

Creates/updates Message Browser with new list of messages, clears previous display

Description

This command is part of the ds_msgdlg function.

Syntax

```
dlgFig = ds_msgdlg('update', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Title'`	Message Browser title
`'ShowErrors'`	Displays error messages. Default: 'on'
`'ShowWarnings'`	Displays warning messages. Default: 'on'
`'ShowNotes'`	Displays note messages. Default: 'off'
`'StatusMessage'`	Prints status messages in MATLAB Command Window, if all messages are excluded from Message Browser. Default: 'on'
`'ShowAdvices'`	Displays advices. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`dlgFig`	Handle of Message Browser, empty matrix if no Message Browser exists

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg('clear')

Purpose

Clears messages in Message Browser

Description

This command is part of the ds_msgdlg function.

Syntax

```
dlgFig = ds_msgdlg('clear')
```

Output parameters

The following output parameters are available:

Parameter	Description
`dlgFig`	Handle of Message Browser, empty matrix if no Message Browser exists

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg('close')

Purpose

Hides Message Browser

Description

This command is part of the ds_msgdlg function.

Syntax

```
dlgFig = ds_msgdlg('close')
```

Output parameters

The following output parameters are available:

Parameter	Description
`dlgFig`	Handle of Message Browser, empty matrix if no Message Browser exists

Related topics

Basics

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

References

Message Handling (TargetLink Tool and Utility Reference)

ds_msgdlg('delete')

Purpose

Deletes Message Browser

Description

This command is part of the ds_msgdlg function.

Syntax

```
ds_msgdlg('delete')
```

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg('find')

Purpose

Finds Message Browser

Description

This command is part of the ds_msgdlg function.

Syntax

```
dlgFig = ds_msgdlg('find')
```

Output parameters

The following output parameters are available:

Parameter	Description
`dlgFig`	Handle of Message Browser, empty matrix if no Message Browser exists

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

ds_msgdlg('callback', propertyName, propertyValue, ...)

Purpose

Invokes control callback

Description

This command is part of the ds_msgdlg function.

Syntax

```
dlgFig = ds_msgdlg('callback', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Control'`	Name of control whose callback to invoke

Output parameters

The following output parameters are available:

Parameter	Description
`dlgFig`	Handle of Message Browser, empty matrix if no Message Browser exists

Related topics

Basics

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

References

Message Handling (📖 TargetLink Tool and Utility Reference)

Maintaining and Documenting

Where to go from here

Information in this section

tl_clean

tl_clean

Purpose

Deletes all files generated by TargetLink.

Description

This function deletes TargetLink-generated files from the working directory. The following subdirectories are also deleted:

- .\TLSim
- .\TlProj
- .\doc
- .\CodeViewFiles

Be careful when you use this utility. All generated files including C, H, OBJ, A2L, etc. will be lost.

Syntax

```
tl_clean()
```

Related topics

Basics

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

tldoc

Where to go from here

Information in this section

tldoc

Purpose

Generates HTML or PDF documentation of the production code generated by TargetLink, and/or the simulation results.

Syntax overview

The following syntaxes are available:

```
[docFid, linkFid] = tldoc('Create', propertyName, propertyValue, ...)
```

Creates new documentation that subsequent tldoc commands can add information to. The encoding of the HTML documentation files is UTF-8.

```
tldoc('Overview', docFid, propertyName, propertyValue, ...)
```

Adds general information to documentation

```
tldoc('TargetLink Code Generation Units', docFid, propertyName, propertyValue, ...)
```

Adds information about the specified TargetLink code generation units to documentation. Code generation units can be TargetLink subsystems, referenced models, or DD CodeGenerationUnit objects in Data Dictionary.

```
tldoc('Simulation Results', docFid, propertyName, propertyValue, ...)
```

Adds information about specified simulation results to documentation

```
tldoc('Close', docFid, propertyName, propertyValue, ...)
```

Closes generated documents

```
tldoc('Convert', propertyName, propertyValue, ...)
```

Converts generated documentation from HTML to PDF format.

Example

```
% Create new documentation files with the base name 'picontroller' in the pipt1_Documentation directory.
docBaseName ='.\pipt1_Documentation\picontroller';
[docFid, linksFid]=tldoc('Create'...
,'DocFileName', docBaseName ...
);

% Add general information to the documentation of the Controller subsystem.
tldoc(docFid,'Overview'...
,'DocFileName',docBaseName ...
,'ModelName','pipt1');


% Add information about the generated functions.
tldoc(docFid,'TargetLink Code Generation Units'...
,'DocFileName',  docBaseName ...
,'ModelName','pipt1'...
,'TlSubsystems','picontroller');


% Close the generated documentation and open it again.
tldoc(docFid,'Close'...
,'DocFileName', docBaseName ...
,'Show','on'...
);


For more examples, refer to tldoc_html_customized.sam and tldoc_pdf_customized.sam.
```

Remarks

The tldoc function has a number of commands for performing various tasks on the HTML document. Each command has its own set of property name/property value pairs for specifying the necessary information. To ensure that information generated by subsequent calls of tldoc commands is written to the same file, a MATLAB file identifier called fid is used. It is the file identifier of the main file which was returned by a previous call of the tldoc('Create', ...) command. The fid parameter is used for all tldoc commands, except for tldoc('Create', ...) and tldoc('Convert', ...), which do not add information to an existing file.

Related topics

Basics

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

HowTos

How to Generate the Documentation (📖 TargetLink Interoperation and Exchange Guide)

References

Document Generation Utility (📖 TargetLink Tool and Utility Reference)

tldoc('Create', propertyName, propertyValue, ...)

Purpose

Creates new documentation that subsequent tldoc commands can add information to. The encoding of the HTML documentation files is UTF-8.

Description

This command is part of the tldoc function.

Syntax

```
[docFid, linkFid] = tldoc('Create', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'DocFileName'`	Base name of document files. It can contain path information such as \TLProj\doc\DocPrj_1 Default: name of current model, if open. Otherwise, name of current DD file.
`'Language'`	Sets language of generated documentation. English ('uk') and German ('de') are supported out of the box. You can add additional language strings to the tldoc_layout.m file. Default: 'uk'
`'TableOfContentsLevel'`	Level of the table of contents in the generated PDF file. Possible values: 1..4 Default: 3

Property	Description
`'Title'`	Title to be shown in the headline of the generated PDF documentation and at the cover page. Default: TargetLink Automatic Documentation: <DocFileName>
`'CoverPage'`	If 'on' cover page is created for the generated PDF documentation. Default: off
`'Subtitle'`	Subtitle to be generated at the covert page. If empty, subtitle is not listed at the cover page. Default: -/-
`'Project'`	Name of the project to be generated at the covert page. If empty, project name is not listed at the cover page. Default: -/-
`'ProjectNumber'`	Project number to be generated at the covert page. If empty, project number is not listed at the cover page. Default: -/-
`'AdditionalCoverPageInfo'`	Cell array with additional information to be generated at the cover page specified as list of captionName/captionValue pairs, for example: {'Project Leader', 'ProjectLeaderName',... 'Status', 'ProjectStatue'} Default: -/-
`'Author'`	Name of the author to be generated at the covert page. If empty, author is not listed at the cover page. Default: -/-
`'AutoDocFilter'`	Filter for Autodoc Customization blocks to be taken into account during documentation generation. Only blocks whose custom tag is the same as one of the strings specified in the filter are evaluated. An empty string '' means Autodoc Customization blocks without a custom tag. If no string is specified in the filter, all Autodoc Customization blocks are considered. Default: {}

Output parameters

The following output parameters are available:

Parameter	Description
`docFid`	File identifier of main HTML documentation file
`linkFid`	File identifier of HTML navigation frame

Related topics

References

tldoc.. 246

tldoc('Overview', docFid, propertyName, propertyValue, ...)

Purpose

Adds general information to documentation

Description

This command is part of the tldoc function.

Syntax

```
tldoc('Overview', docFid, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'DocFileName'`	Base name of document files. It can contain path information such as \TLProj\doc\DocPrj_1 Default: name of current model, if open. Otherwise, name of current DD file.
`'Language'`	Sets language of generated documentation. English ('uk') and German ('de') are supported out of the box. You can add additional language strings to the tldoc_layout.m file. Default: 'uk'
`'ModelName'`	Name of root model the document generator collects requested information from. Default: current model
`'AddToLinks'`	If 'on', adds entry for this section to <DocFileName>_links.html file. Default: 'on'
`'FontSize'`	Sets font size of generated documentation. Possible values: 1- 5. Default: 3
`'ViewMode'`	Optimizes output of generated HTML documentation either for an HTML browser or for printing purposes: ▪ 'Screen' - Images or tables widths/heights calculated by the browser ▪ 'Print' - Image or table widths/heights as specified by user Default: Print
`'ImageMaxWidth'`	Sets maximum width of image inserted into generated documentation in millimeters (unit: mm) or as a pixel value (unit: px). The value must be specified with the unit, e.g., '170mm' or '600px'. No value denotes that there is no limitation to the image width. Evaluated only if ViewMode property is set to 'Print'. Default: []

Property	Description
`'ImageMaxHeight'`	Sets maximum height of image inserted into generated documentation in millimeters (unit: mm) or as a pixel value (unit: px). The value must be specified with the unit, e.g., '170mm' or '600px'. No value denotes that there is no limitation to the image height. Evaluated only if ViewMode property is set to 'Print'. Default: []
`'IncludeSimulationData'`	Adds SIL/PIL simulation-related information, such as evaluation board used. Default: 'on'
`'DDProjectFileName'`	Name of DD file that the Document Generator collects requested information from. Default: currently open Data Dictionary
`'Department'`	Department of author
`'AdditionalInfo'`	Adds additional information about documentation basis: e.g., list of TargetLink subsystems and DD CodeGenerationUnits, simulation data and model comment. Default: 'on'
`'ShowImage'`	If 'on', adds an image of Simulink model (Overview) or function systems (TargetLink Code Generation Units). Default: 'on'
`'ImageNameTemplate'`	Specifies how image files are named. You can use (combinations of) the following TargetLink name macros. Command: Overview ▪ '$M'- Model name ▪ '$B' - Block name Default: $M Command: TargetLink Code Generation Units ▪ '$M'- Model name ▪ '$N' - TargetLink root system name ▪ '$F' - Function name in capital letters ▪ '$f' - Function name as in the code ▪ '$I' - TargetLink subsystem ID ▪ '$B' - Block name Default: $N_$B
`'ImageFormatType'`	Format of the created image files. The following formats are supported: SVG ('svg') and PNG ('png'). Default: 'svg'

Input parameters

The following input parameters are available:

Parameter	Description
`docFid`	File identifier of main HTML documentation file as returned by Create command

Related topics

References

tldoc 246

tldoc('TargetLink Code Generation Units', docFid, propertyName, propertyValue, ...)

Purpose

Adds information about the specified TargetLink code generation units to documentation. Code generation units can be TargetLink subsystems, referenced models, or DD CodeGenerationUnit objects in Data Dictionary.

Description

This command is part of the tldoc function.

Syntax

```
tldoc('TargetLink Code Generation Units', docFid, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'ModelName'`	Name of root model the document generator collects requested information from. Default: current model
`'DDProjectFileName'`	Name of DD file that the Document Generator collects requested information from. Default: currently open Data Dictionary
`'Application'`	Name of application the document generator collects information from

Property	Description
`'AtomicSubsystems'`	If 'on', adds information about code parts generated for atomic subsystems which are directly inlined in the parent function. NOTE: Variables used in the inlined code and corresponding to one atomic subsystem are also described in the parented function part. No function prototype is generated. A reference to the parent function is generated. Default: 'off'
`'AutodocFunctionSelection'`	If 'on', only functions generated for subsystems and Stateflow charts with an AutoDoc Customization block are documented. Default: 'off'
`'Functions'`	Functions generated by TargetLink to be included in generated documentation. Equivalent to using AutodocFunctionSelection property. Default: 'all'
`'GeneralInfo'`	If 'on', adds general information about selected code generation units. Default: 'on'
`'FunctionInterface'`	If 'on', adds information about prototypes and input/output signals of all functions contained in selected code generation units. Default: 'on'
`'OmitCaptions'`	If 'on', captions are not included in generated documentation. Default: 'off'
`'SeparateFiles'`	If 'on', each documented function is stored in separate file. Default: 'off'
`'FileNameTemplate'`	Specifies how additional files are named if SeparateFile property is enabled. You can use one or a combination of the following TargetLink name macros: ▪ '$M' - Model name ▪ '$N' - TargetLink root system name ▪ '$I' - TargetLink subsystem ID ▪ '$F' - Function name in capital letters ▪ '$f' - Function name as in the code
`'RTOSInfo'`	If 'on', prints Multirate/RTOS-specific data such as tasks, messages, and events. Default: 'on'
`'BlockData'`	If 'on', adds information about displayable and calibratable variables in code. Default: 'on'
`'PrintRequirementInfo'`	If 'on', prints requirement info. Default: 'on'
`'RequirementInfoWithScreenShot'`	If 'on', prints screenshot of every system containing Simulink block or Stateflow object referencing a Requirement Info object. Default: 'on'
`'TypeInfo'`	If 'on', adds information about data type names, variable classes, user data types, etc. Default: 'on'

Property	Description
`'UseUserTypeName'`	If 'on', uses user type names in variable tables. Default: 'on'
`'Statistics'`	If 'on', adds statistical data about selected TargetLink subsystems, e.g., summary of applied block types and scaling information. Default: 'on
`'Stateflow'`	If 'on', prints Stateflow-specific data. Default: 'on'
`'ExchangeableWidthMacros'`	If 'on', prints an overview of width macros used in production code. For details on width macros, refer to Basics on Variable Vector Widths. Default: 'on'
`'AUTOSARInfo'`	If 'on', prints information on AUTOSAR software components, if any. Default: 'on'
`'AUTOSARBlockData'`	If 'on', prints information on AUTOSAR calibratable parameters and on per instance memories used in documented runnables. Default: 'on'
`'AddToLinks'`	If 'on', adds entry for this section to <DocFileName>_links.html file. Default: 'on'
`'FontSize'`	Sets font size of generated documentation. Possible values: 1- 5. Default: 3
`'ViewMode'`	Optimizes output of generated HTML documentation either for an HTML browser or for printing purposes: ▪ 'Screen' - Images or tables widths/heights calculated by the browser ▪ 'Print' - Image or table widths/heights as specified by user Default: Print
`'ImageMaxWidth'`	Sets maximum width of image inserted into generated documentation in millimeters (unit: mm) or as a pixel value (unit: px). The value must be specified with the unit, e.g., '170mm' or '600px'. No value denotes that there is no limitation to the image width. Evaluated only if ViewMode property is set to 'Print'. Default: []
`'ImageMaxHeight'`	Sets maximum height of image inserted into generated documentation in millimeters (unit: mm) or as a pixel value (unit: px). The value must be specified with the unit, e.g., '170mm' or '600px'. No value denotes that there is no limitation to the image height. Evaluated only if ViewMode property is set to 'Print'. Default: []
`'TableLayout'`	Set of table setting specifications applied to generated documentation. You can define different sets in the tldoc_layout.m file. Default: default

Property	Description
`'ImageNameTemplate'`	Specifies how image files are named. You can use (combinations of) the following TargetLink name macros. Command: Overview ▪ '$M'- Model name ▪ '$B' - Block name Default: $M Command: TargetLink Code Generation Units ▪ '$M'- Model name ▪ '$N' - TargetLink root system name ▪ '$F' - Function name in capital letters ▪ '$f' - Function name as in the code ▪ '$I' - TargetLink subsystem ID ▪ '$B' - Block name Default: $N_$B
`'ShowImage'`	If 'on', adds an image of Simulink model (Overview) or function systems (TargetLink Code Generation Units). Default: 'on'
`'Language'`	Sets language of generated documentation. English ('uk') and German ('de') are supported out of the box. You can add additional language strings to the tldoc_layout.m file. Default: 'uk'
`'DocFileName'`	Base name of document files. It can contain path information such as \TLProj\doc\DocPrj_1 Default: name of current model, if open. Otherwise, name of current DD file.
`'FunctionsHierarchy'`	If 'on ' functions hierarchy is listed. Deafult: 'on'
`'ImageFormatType'`	Format of the created image files. The following formats are supported: SVG ('svg') and PNG ('png'). Default: 'svg'
`'CodeGenerationUnits'`	List of model-based and DD-based code generation units to be documented. The model-based code generation units have to be specified by their names, the DD-based code generation units have to be specified by their DD paths. Only code generation units that production code was generated for can be documented. If set to 'all', all code generation units in the specified TargetLink Data Dictionary and model are considered. Default: 'all'
`'GenerationMode'`	Specifies how the documentation is to be generated. The following values are possible: ▪ 'standard' - Documentation is generated only for the specified code generation units. ▪ 'incremental' - Documentation is generated for the specified code generation units. If there are nested incremental code generation units, an existing documentation is embedded into the generated one for them. If such documentation does not exist yet, it is created.

Property	Description
`'IncrementalDocFilesLocation'`	Location of the documentation of incremental code generation units (CGU) to be embedded in currently generated documentation. It is specified as a cell array of cguName/cguDocLocation pairs. cguDocLocation can be either the directory where the <cguName>_main.html file is located, or the full name of the *_main.html file. If the cell array is empty, the following default location is assumed: <currentDocDirectory>\<cguName>\<cguName>_main.html Default: {}

Input parameters

The following input parameters are available:

Parameter	Description
`docFid`	File identifier of main HTML documentation file as returned by Create command

Related topics

References

tldoc.. 246

tldoc('Simulation Results', docFid, propertyName, propertyValue, ...)

Purpose

Adds information about specified simulation results to documentation

Description

This command is part of the tldoc function.

Syntax

```
tldoc('Simulation Results', docFid, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'DocFileName'`	Base name of document files. It can contain path information such as \TLProj\doc\DocPrj_1 Default: name of current model, if open. Otherwise, name of current DD file.

Property	Description
`'ModelName'`	Name of root model the document generator collects requested information from. Default: current model
`'AddToLinks'`	If 'on', adds entry for this section to <DocFileName>_links.html file. Default: 'on'
`'Simulations'`	Names of simulations to be documented. Default 'all'
`'PlotStyle'`	Specifies how logged signals are plotted: ▪ 'all in one' - One plot window for each simulation containing single plots for each logged signal, simulation-specific ▪ 'separate' - One plot window for each logged signal, containing signal plots across simulations ▪ 'off' - No plots included
`'SimulationInfo'`	If 'on', documents brief information about simulation. Default: 'on
`'ShowSignalLimits'`	If 'on', shows scaling limits of variables in plots. Default: 'off'
`'FontSize'`	Sets font size of generated documentation. Possible values: 1- 5. Default: 3
`'ViewMode'`	Optimizes output of generated HTML documentation either for an HTML browser or for printing purposes: ▪ 'Screen' - Images or tables widths/heights calculated by the browser ▪ 'Print' - Image or table widths/heights as specified by user Default: Print
`'ImageMaxWidth'`	Sets maximum width of image inserted into generated documentation in millimeters (unit: mm) or as a pixel value (unit: px). The value must be specified with the unit, e.g., '170mm' or '600px'. No value denotes that there is no limitation to the image width. Evaluated only if ViewMode property is set to 'Print'. Default: []
`'ImageMaxHeight'`	Sets maximum height of image inserted into generated documentation in millimeters (unit: mm) or as a pixel value (unit: px). The value must be specified with the unit, e.g., '170mm' or '600px'. No value denotes that there is no limitation to the image height. Evaluated only if ViewMode property is set to 'Print'. Default: []
`'ImageFormatType'`	Format of the created image files. The following formats are supported: SVG ('svg') and PNG ('png'). Default: 'svg'

Input parameters

The following input parameters are available:

Parameter	Description
`docFid`	File identifier of main HTML documentation file as returned by Create command

Related topics

References

tldoc.. 246

tldoc('Close', docFid, propertyName, propertyValue, ...)

Purpose

Closes generated documents

Description

This command is part of the tldoc function.

Syntax

```
tldoc('Close', docFid, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'DocFileName'`	Base name of document files. It can contain path information such as \TLProj\doc\DocPrj_1 Default: name of current model, if open. Otherwise, name of current DD file.
`'Show'`	If 'on', displays generated HTML documentation in web browser. Default: 'on'
`'SeparateListForUserChapterLinks'`	If 'on', the links to the user chapters are listed at the end of the navigation page. Otherwise the order of the hyperlinks and the order of the documentation chapters agree. Applies only for HTML documentation. Default: 'off'

Input parameters

The following input parameters are available:

Parameter	Description
`docFid`	File identifier of main HTML documentation file as returned by Create command

Related topics

References

tldoc('Convert', propertyName, propertyValue, ...)

Purpose

Converts generated documentation from HTML to PDF format.

Description

This command is part of the tldoc function.

Syntax

```
tldoc('Convert', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'FileList'`	Specifies XML file that contains the file name list as generated by documentation generation process. Exclusive with 'FileName'.
`'FileName'`	Specifies name of the HTML file to be converted. Useful if only one file is to be converted. Exclusive with 'FileList'.
`'CreatePDFFile'`	If 'on', converts specified file(s) to PDF. Default: 'on'
`'Show'`	If 'on', opens converted files. Default: 'on'

Property	Description
`'SystemFontName'`	Name of the system installed font that is to be used in the generated PDF documentation, for example 'MS Gothic','Meiryo'. Exclusive with 'TTFFile'. The list of fonts installed at your system can be found under <WindowsRoot>/Fonts. If no system font is specified the default font used in the PDF documentation depends on the Matlab encoding: For 'Shift_JIS' MS Mincho (Windows 7) resp. Yu Gothic UI (Windows 10), otherwise - sans-serif. Note: The system font is not embedded into the generated PDF documentation. Default:-/-
`'TTFFile'`	Full path of the TTF-file defining a custom font to be used in the generated PDF documentation. Exclusive with 'SystemFontName'. Note: The custom font is embedded into the generated PDF documentation. Default:-/-

Related topics

References

tldoc.. 246

tlRequirementInfo

tlRequirementInfo

Purpose

Manages requirement information at TargetLink model elements.

Syntax overview

The following syntaxes are available:

```
[errorflag, msg] = tlRequirementInfo('Add', hBlock, requirement)
```
adds new requirements to requirements data string

```
[errorflag, msg] = tlRequirementInfo('Remove', hBlock, idx)
```
removes requirement entries from requirements data string

```
[reqData, errorflag, msg] = tlRequirementInfo('Get', hBlock)
```
gets struct of requirement information

```
[errorflag, msg] = tlRequirementInfo('Set', hBlock, reqData)
```
sets struct of requirement information

Input parameters

The following input parameters are available:

Parameter	Description
hBlock	Simulink identifier of Simulink block or Stateflow object.
requirement	Cell representing a requirement entry, i.e. {<reference to Data Dictionary RequirementInfo object>, <annotation>}
idx	Index of requirement object in struct. If no index is specified, all requirement entries are removed.
reqData	Requirement data struct consisting of a reference field and an annotation field.

Output parameters

The following output parameters are available:

Parameter	Description
reqData	Requirement data struct consisting of a reference field and an annotation field.
errorflag	Non-zero number in case of failure, zero on success.
msg	Error message, empty on success.

Related topics

Basics

Basics on Using DD-Based Requirement Information (📖 TargetLink Interoperation and Exchange Guide)

Performing Checks

Where to go from here

Information in this section

tl_check_module_ownership

tl_check_module_ownership

Purpose

Checks the specification of the module ownership for the given system.

Description

This function checks the module ownership for a given TargetLink subsystem, subsystem configured for incremental code generation, or referenced model (CGUs).

If a DD ModuleOwnerShip object exists for the given CGU, or if it is created by this function, its DD handle is returned. Otherwise an empty matrix is returned. In the following cases an error is displayed:

- E4407 - Module ownership is specified only at the Function block
- E4412 - Different module ownerships for one system are specified at the Function block and in the ModuleOwnership object in the Data Dictionary
- E4402 - There is more than one ModuleOwnership object per system in the Data Dictionary

NOTE: This function takes only DD ModuleOwnerShip objects into account that are not excluded from the code generation (ExcludeFromCodeGeneration property is set to 'off').

Syntax

```
hDdModuleOwnerShip = tl_check_module_ownership(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Name of the root model containing the model components that are to be checked. Default: current model
`Systems`	List of the systems to be checked, specified as a cell array of the systems' Simulink paths, or as an array of their Simulink handles. Default: All models CGUs. Whether referenced models are checked by default depends on the SearchInRefModels property.
`SearchInRefModels`	Specifies whether referenced models are to be checked by default (see Systems property). Default: 'off'
`CreateModuleOwnerShip`	Specifies whether missing ModuleOwnerShip objects are to be created in the Data Dictionary. Default: 'off'
`CreateModules`	Specifies whether missing Module objects are to be created in the Data Dictionary. Default: 'off'
`ModuleGroup`	Path in the Data Dictionary where the Module objects are to be created. Default: Pool/Modules

Output parameters

The following output parameters are available:

Parameter	Description
`hDdModuleOwnerShip`	Handle of the Data Dictionary ModuleOwnerShip object. Empty matrix if no ModuleOwnerShip object exists.

Example

```
% Check the module ownership specification for incremental subsystem
% mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_SensorCorrection
tl_check_module_ownership('Systems',...
'mdlref_fuelsys/fuelratecontroller/Subsystem/fuelratecontroller/r_SensorCorrection');
```

tl_check_usertypes

tl_check_usertypes

Purpose

Checks user types in Simulink model.

Description

This function checks TargetLink blocks in a Simulink system if variables are scaled with user data types that are not defined in the Data Dictionary.

Syntax

```
tl_check_usertypes(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Simulink model name. Default: bdroot

Example

```
% check user types in the current Simulink model
tl_check_usertypes
% check user types in model "poscontrol"
tl_check_usertypes('model','poscontrol')
```

Switching TargetLink's Blockset Mode

tlOperationMode

tlOperationMode

Purpose

Gets or sets TargetLink's operation mode.

Syntax overview

The following syntaxes are available:

```
currentMode = tlOperationMode('Get')
```

Returns the current operation mode

```
tlOperationMode('Set', operationMode)
```

Sets the current operation mode to 'ModelingOnly' or 'FullFeatured'

```
isFullFeatured = tlOperationMode('IsFullFeatured')
```

Returns true if FullFeatured mode is active

Input parameters

The following input parameters are available:

Parameter	Description
`operationMode`	Mode to be set: 'ModelingOnly' or 'FullFeatured'

Output parameters

The following output parameters are available:

Parameter	Description
`currentMode`	Current operation mode of TargetLink
`isFullFeatured`	true if FullFeatured mode is active

Related topics

Basics

Overview of the TargetLink Operation Modes (📖 TargetLink Blockset Guide)

Examples

Example of Working in the Modeling Only operation mode (📖 TargetLink Blockset Guide)

Data Dictionary MATLAB API Tools

Where to go from here

Information in this section

ddv

ddv

Purpose

Retrieves the value of a DD Variable object.

Description

This function returns the value and width of a DD Variable object (= property "Width" and "Value"). If there are data or code variants, the value and width associated with the currently active variant is returned. The function also considers variable vector width configurations.

Use this function to associate a block variable with the value of a DD Variable object.

If the value cannot be retrieved (e.g., because the DD Variable object does not exist), the function aborts with an error if invoked with less than four output arguments. Invalid ddv expressions used in a model thus result in Simulink initialization errors.

Syntax

```
[value, width, hDDVariable, msg] = ddv(DDVar)
```

Input parameters

The following input parameters are available:

Parameter	Description
`DDVar`	Relative DD path beginning at /Pool/Variables, or full DD path of DD Variable object

Output parameters

The following output parameters are available:

Parameter	Description
`value`	Value of variable as specified by the Variable object's Value property
`width`	Width of variable a specified by the Variable object's Width property, or the associated ExchangeableWidth object
`hDDVariable`	DD handle of Variable object (empty matrix if object does not exist)
`msg`	Error message (empty on success)

Example

```
% get the value of Variable /Pool/Variables/nCyl
value =ddv('nCyl');

% get the value of Variable /Pool/Variables/CalVars/Kp3
value =ddv('CalVars/Kp3');

% get the value of Variable /Subsystems/ECU/injection/Variables/TorqueMap
[value,width,hVar,msg]=ddv('/Subsystems/ECU/injection/Variables/TorqueMap');
if~isempty(msg)
disp(['Error retrieving value: ' msg]);
end

% use TL API to associate table matrix of lookup block with a DD Variable object /Pool/Variables/TableMatrices/table_5
tl_set(hLookupBlock,'table.value','ddv(''TableMatrices/table_5'')');
```

dsdd_check_msg

dsdd_check_msg

Purpose

Checks for errors and messages after the Data Dictionary has been accessed.

Description

This function checks for errors and messages after the Data Dictionary has been accessed. If there is any message, it is registered in TargetLink's message system and the corresponding message number is returned.

If the only input argument is the string 'register', all DD messages from the DD message list are registered.

If the second input argument is the string 'register', then the message is only registered, and no message box is displayed.

Registered messages can be displayed by opening the TargetLink Message Browser, or by calling the API function ds_error_display.

Syntax

```
errorCode = dsdd_check_msg(errorCode, msgText)
```

Input parameters

The following input parameters are available:

Parameter	Description
`errorCode`	Error code as returned from the DD MATLAB API
`msgText`	Text to be displayed instead of the generic message text (optional). This text may contain the following tokens which will be replaced by the corresponding objects: ▪ $OBJECT$ - Name of the object which caused the error ▪ $MSG$ - Message string ▪ \n - Newline character Alternatively, the string 'register' specifies that the message should be registered in TargetLink's message list only.

Output parameters

The following output parameters are available:

Parameter	Description
`errorCode`	0 on success, the DD error code on failure

Example

```
% invoke DD MATLAB API command
[h,err]=dsdd('GetAttribute','/Pool/Variables/Gain','hDDObject');

% display message box and return on error
ifdsdd_check_msg(err),return;end

% invoke DD MATLAB API command
[h,err]=dsdd('GetAttribute','/Pool/Variables/Gain','hDDObject');

% display message box with custom message and return on error
ifdsdd_check_msg(err,'My own message\n$MSG$\n$OBJECT$'),return;end

% invoke DD MATLAB API command
[h,err]=dsdd('GetAttribute','/Pool/Variables/Gain','hDDObject');

% register message and return on error (do not open a message box)
ifdsdd_check_msg(err,'register'),return;end

% display message box with registered message
ds_error_display;
```

dsdd_compare_optionset

dsdd_compare_optionset

Purpose

Compares the code generator options of a model with an option set.

Description

This function shows the differences in the code generator options of a model compared to an option set from the TargetLink Data Dictionary.

Syntax

```
[differences, msgStruct] = dsdd_compare_optionset(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Handle or name of model. If this argument is missing, the function uses the current model
`OptionSetName`	Name of option set

Output parameters

The following output parameters are available:

Parameter	Description
`differences`	Structure containing differences that have been found
`msgStruct`	Structure containing error and warning messages

Related topics

Basics

Basics on Configuring the Code Generator for Production Code Generation (📖 TargetLink Customization and Optimization Guide)

References

Code Generator Options (📖 TargetLink Model Element Reference)
dsdd_export_optionset........ 274
dsdd_get_creator_options........ 277
dsdd_import_optionset........ 279
Edit Code Generator Options (📖 TargetLink Data Dictionary Manager Reference)
Export Options to Model <Model> (📖 TargetLink Data Dictionary Manager Reference)
Import Options from Model <Model> (📖 TargetLink Data Dictionary Manager Reference)

dsdd_export_a2l_file

dsdd_export_a2l_file

Purpose

Exports an A2L file from the Data Dictionary.

Description

This function exports an A2L file from the specified DD Subsystem objects and for the platform specified by board/compiler or TargetInfo/TargetConfig pairs. Via additional options it is possible to control the contents of the generated A2L file.

Syntax

```
bSuccess = dsdd_export_a2l_file(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Application`	Name of DD Application object where platform-specific DD Build object is created. Default: Existing Application object if there is no more than one DD Application object
`Build`	Name of DD Build object to be created or used. Default: 'Build'
`Board`	Name of the evaluation board. Must be set by pairs with the 'Compiler' property. Default: -/-
`Compiler`	Name of the compiler. Must be set by pairs with the 'Board' property. Default: -/-
`TargetInfoDir`	Directory where the target info file, TargetInfo.xml file, resides. It is taken into account only if the property pair 'Board' and 'Compiler' is not set. Default: -/-
`TargetConfigDir`	Directory where the target config file, TargetConfig.xml, resides. It is taken into account only if the property pair 'Board' and 'Compiler' is not set. Default: -/-
`MapFileName`	Name of the linker map file to be parsed to get address information. Default: -/-
`Subsystems`	Names of DD Subystem objects to generate an A2L file for. Default: -/-
`File`	Name of generated A2L file. Default: untitled.a2l
`StyleSheet`	Name of style sheet used for XSL transformation. Default: If <A2LStyleSheetsTLPreferencesDirectory> set: <A2LStyleSheetsTLPreferencesDirectory>\a2l_export_control.xsl Otherwise <TLInstallation>\Dsdd\A2L\StyleSheets\a2l_export_control.xsl

Property	Description
`ASAP1BInterfaces`	List of ASAP1B interfaces. Default: -/-
`Phase`	Phase of the A2L export to be executed: ▪ 'ConvertToA2l' ▪ 'WriteA2Lfile' ▪ 'CreateAll' Default: 'CreateAll'
`UseLookupStructures`	If 'off', creates CHARACTERISTIC for Curves and Maps with shared axes (COM_AXIS). If 'on', creates CHARACTERISTIC for Curves and Maps with standard axes and one common RECORD_LAYOUT for axis and table values. Default: 'off'
`OverwriteCalProperties`	If 'on', overwrites the following DD Variable object's properties generated by previous A2L file generator run: ▪ Accuracy ▪ Resolution ▪ MaxDiff ▪ ByteOrder Default: 'on'
`MergeA2LModules`	If 'off', generates a separate MODULE element for each specified DD Subsystem object. Default: 'on'
`ProjectFrame`	If 'on', generates PROJECT element as frame for MODULE elements in A2L file. Default: 'on'
`NamePrefix`	Name prefix to be used in names of COMPU_METHOD and RECORD_LAYOUT elements. Default: -/-
`AllowCFormatSpecifiers`	If 'on', for the display format specified at the DD Scaling object and mapped to the Format property of the corresponding COMPU_METHOD, the C format syntax (e.g., %5.2f) is also supported. If 'off', the display format must be given in the ASAM MCD-2MC valid syntax of %[length].[layout]. Default: 'on'
`UseUnderlyingEnumDataTypeInfo`	If 'on', uses the information about the underlying integer data type for the C enums data type that was specified via the Typedef objects below the <Build>/EnumDataTypes DD object. The underlying integer data type of C enums depends on the platform and on the compiler settings. Therefore, the underlying integer data types that TargetLink gets for SIL/PIL simulation might differ from the underlying integer data types that are valid for the final ECU application. Default: 'off'
`ExternalVariables`	If 'on', includes description of calibratable and measurebale external variables into the A2L file. Default: 'off'

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	▪ 0 - Error ▪ 1 - Success

Example

```
bSuccess =dsdd_export_a2l_file(...
'Application','pipt1',...
'Build','Build',...
'Subsystems','picontroller',...
'File','pipt1.a2l',...
'ASAP1BInterfaces','CCP',...
'MergeA2lModules','on',...
'Board','Promo167',...
'Compiler','Task60',...
'MapFileName','pipt1.map');
```

dsdd_export_optionset

dsdd_export_optionset

Purpose

Transfers the option set properties to a model.

Description

This function sets the code generator options of a model according to the options specified in an option set stored in the TargetLink Data Dictionary.

Syntax

```
msgStruct = dsdd_export_optionset(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Handle or name of model. If this argument is missing, the function uses the current model
`OptionSetName`	Name of option set

Output parameters

The following output parameters are available:

Parameter	Description
`msgStruct`	Message structure containing error messages

Related topics

Basics

Basics on Configuring the Code Generator for Production Code Generation (📖 TargetLink Customization and Optimization Guide)

References

Code Generator Options (📖 TargetLink Model Element Reference)
dsdd_compare_optionset.. 270
dsdd_get_creator_options.. 277
dsdd_import_optionset.. 279
Edit Code Generator Options (📖 TargetLink Data Dictionary Manager Reference)
Export Options to Model <Model> (📖 TargetLink Data Dictionary Manager Reference)
Import Options from Model <Model> (📖 TargetLink Data Dictionary Manager Reference)

dsdd_free

dsdd_free

Purpose

Clears TargetLink's Data Dictionary repositories.

Description

This function closes all Data Dictionary (DD) workspaces, the DD Manager, and the TargetLink Property Manager, and removes TargetLink DD MEX functions from memory. All DD data is deleted. TargetLink's Data Dictionary is thus set into its initial state.

Syntax

```
dsdd_free()
```

dsdd_get_block_path

dsdd_get_block_path

Purpose

Returns the Simulink or Stateflow path associated with a DD object.

Description

This function returns the Simulink or Stateflow path of the Simulink block or Stateflow object associated with a DD object. This enables to find objects, such as the TargetLink block for which a variable (described by a DD Variable object in the Subsystems area) has been generated.

Syntax

```
blockPath = dsdd_get_block_path(DDObject)
```

Input parameters

The following input parameters are available:

Parameter	Description
`DDObject`	DD handle or path of DD object

Output parameters

The following output parameters are available:

Parameter	Description
`blockPath`	Simulink or Stateflow path of associated Simulink block, or Stateflow object

Example

```
Which block is represented by a variable in production code generated for the ar_poscontrol demo model?

dsdd_get_block_path('/Subsystems/TL_Controller/controller/Variables/controller_runnable/S12_e1')

ans =

ar_poscontrol/TL_Controller/Subsystem/TL_Controller/Controller_Runnable/e1
```

dsdd_get_creator_options

dsdd_get_creator_options

Purpose

Returns options used to create the specified Data Dictionary subsystem object.

Syntax overview

The following syntaxes are available:

```
[options, values] = dsdd_get_creator_options(subsystem)
```

Returns the list of all options and their values

```
[value_1, ..., value_n] = dsdd_get_creator_options(subsystem, option_1, ..., option_n)
```

Return values of 1:n options

Input parameters

The following input parameters are available:

Parameter	Description
`subsystem`	DD object identifier of subsystem object
`option_1`	First option
`...`	...
`option_n`	n-th option

Output parameters

The following output parameters are available:

Parameter	Description
`value_1`	Value of first option
`...`	...
`value_n`	Value of n-th option.
`options`	List of option names
`values`	Cell array of values

Example

```
[allOptions, allValues]=dsdd_get_creator_options('controller');
[ansiCode, optLevel]=dsdd_get_creator_options('controller',...
'ANSI-C compatible code','Optimization level');
```

Related topics

Basics

Basics on Configuring the Code Generator for Production Code Generation (📖 TargetLink Customization and Optimization Guide)

References

dsdd_get_width

dsdd_get_width

Purpose

Returns the value of the Width property associated with a specified variable value.

Description

The Width and Value properties of DD Variable objects must match:

- If the Width property is empty, the Variable object specifies a scalar. In this case, the Value (if not empty) must be a scalar.
- If the Width property is a scalar, the Value property (if not empty) must be a vector with as many elements.
- If the Width property is a vector, the Value property (if not empty) must be a matrix. Each element of the Width specifies the number of elements of one dimension.

The dsdd_get_width tool returns the value for the Width property for a specified Value property.

Syntax

```
width = dsdd_get_width(value)
```

Input parameters

The following input parameters are available:

Parameter	Description
`value`	Value for the DD Variable's Value property

Output parameters

The following output parameters are available:

Parameter	Description
`width`	Resulting value for the Width property of the DD Variable object

Example

```
% hDDVariable is the DD handle of a DD Variable object.
value =[123];
width =dsdd_get_width(value);
dsdd('SetValue', hDDVariable, value);
dsdd('SetWidth', hDDVariable, width);
```

dsdd_import_optionset

dsdd_import_optionset

Purpose

Creates a code generator option set in the TargetLink Data Dictionary

Description

This function reads the code generator options from a specified model and saves the options as an option set in the TargetLink Data Dictionary.

Syntax

```
[hOptionset, msgStruct] = dsdd_import_optionset(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`Model`	Handle or name of model. If this argument is missing, the function uses the current model
`OptionSetName`	Name of option set
`ForceOverwrite`	Forces overwriting of an existing option set

Output parameters

The following output parameters are available:

Parameter	Description
`hOptionset`	Reference of created option set
`msgStruct`	Message structure containing error messages

Related topics

Basics

Basics on Configuring the Code Generator for Production Code Generation (📖 TargetLink Customization and Optimization Guide)

References

Code Generator Options (📖 TargetLink Model Element Reference)
dsdd_compare_optionset.. 270
dsdd_export_optionset.. 274
dsdd_get_creator_options.. 277
Edit Code Generator Options (📖 TargetLink Data Dictionary Manager Reference)
Export Options to Model <Model> (📖 TargetLink Data Dictionary Manager Reference)
Import Options from Model <Model> (📖 TargetLink Data Dictionary Manager Reference)

dsdd_manage_application

Where to go from here

Information in this section

dsdd_manage_application

Purpose

Manages DD Application objects.

Syntax overview

The following syntaxes are available:

```
application = dsdd_manage_application('GetApplication', propertyName, propertyValue, ...)
```
Gets application name for given model/subsystem

```
dsdd_manage_application('SetApplication', propertyName, propertyValue, ...)
```
Sets application name for given model/subsystem

```
hDDApplication = dsdd_manage_application('UpdateConfig', propertyName, propertyValue, ...)
```
Updates application configuration (nested subsystems)

```
subsystemHierarchy = dsdd_manage_application('GetSubsystems', propertyName, propertyValue, ...)
```
Returns nested systems hierarchy of specified system

```
info = dsdd_manage_application('CheckSubsystems', propertyName, propertyValue, ...)
```
Checks if subsystems belonging to an application are compatible to target/compiler/basetypes/library function calls

Example

```
% Get the application name for the current model
application =dsdd_manage_application('GetApplication')

% Update the subsystem configuration for application 'pipt1'
dsdd_manage_application('UpdateConfig','Application','pipt1','Subsystem','pipt1/picontroller/Subsystem/picontroller')

% Get subsystems hierarchy
subsystemHierarchy =dsdd_manage_application('GetSubsystems','Application','poscontrol',...
'SubsystemName','controller')

% Check the subsystems belonging to the current application
info =dsdd_manage_application('CheckSubsystems','Subsystems', subsystemHierarchy,...
'Target','C16x',...
'Compiler','TASK60');
```

dsdd_manage_application('CheckSubsystems', propertyName, propertyValue, ...)

Purpose

Checks if subsystems belonging to an application are compatible to target/compiler/basetypes/library function calls

Description

This command is part of the dsdd_manage_application function.

Syntax

```
info = dsdd_manage_application('CheckSubsystems', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Subsystems'`	Vector of structures that describe subsystem's hierarchy as returned by GetSubsystems function
`'Target'`	Target processor abbreviation
`'Compiler'`	Compiler abbreviation

Output parameters

The following output parameters are available:

Parameter	Description
`info`	Struct with information on checked DD Subsystem objects

dsdd_manage_application('GetApplication', propertyName, propertyValue, ...)

Purpose

Gets application name for given model/subsystem

Description

This command is part of the dsdd_manage_application function.

Syntax

```
application = dsdd_manage_application('GetApplication', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Subsystem'`	Path of TargetLink subsystem (configured for incremental code generation)/referenced model block (Command: SetApplication/GetApplication)/referenced model (Command: UpdateConfig)

Output parameters

The following output parameters are available:

Parameter	Description
`application`	Name of application

dsdd_manage_application('GetSubsystems', propertyName, propertyValue, ...)

Purpose

Returns nested systems hierarchy of specified system

Description

This command is part of the dsdd_manage_application function.

Syntax

```
subsystemHierarchy = dsdd_manage_application('GetSubsystems', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Application'`	Name of application: e.g., 'poscontrol'
`'SubsystemName'`	Name of DD Subsystem object

Output parameters

The following output parameters are available:

Parameter	Description
`subsystemHierarchy`	Vector with handles of DD Subsystem objects

dsdd_manage_application('SetApplication', propertyName, propertyValue, ...)

Purpose

Sets application name for given model/subsystem

Description

This command is part of the dsdd_manage_application function.

Syntax

```
dsdd_manage_application('SetApplication', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Subsystem'`	Path of TargetLink subsystem (configured for incremental code generation)/referenced model block (Command: SetApplication/GetApplication)/referenced model (Command: UpdateConfig)
`'Application'`	Name of application: e.g., 'poscontrol'

dsdd_manage_application('UpdateConfig', propertyName, propertyValue, ...)

Purpose

Updates application configuration (nested subsystems)

Description

This command is part of the dsdd_manage_application function.

Syntax

```
hDDApplication = dsdd_manage_application('UpdateConfig', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Application'`	Name of application: e.g., 'poscontrol'
`'Subsystem'`	Path of TargetLink subsystem (configured for incremental code generation)/referenced model block (Command: SetApplication/GetApplication)/referenced model (Command: UpdateConfig)

Output parameters

The following output parameters are available:

Parameter	Description
`hDDApplication`	Handle of DD Application object

dsdd_manage_build

Where to go from here

Information in this section

dsdd_manage_build

Purpose

Creates a DD Build object.

Syntax overview

The following syntaxes are available:

`[hBuild, targetInfo] = dsdd_manage_build('Create', propertyName, propertyValue, ...)`

Creates a DD Build object with the specified name in the specified DD Application object tree as follows:

- Sets the platform-specific properties of the embedded `<Application>/<Build>/TargetInfo` object on the basis of the `TargetInfo.xml` file.
- Imports the descriptions of the base types from the `TargetConfig.xml` file into the `<Application>/<Build>/TargetInfo/BaseTypes` object.

DD Build objects created with this function can be used for A2L file generation. The location of the `TargetInfo.xml` and `TargetConfig.xml` files can be specified directly or by means of the compiler and evaluation board names.

`dsdd_manage_build('ImportSymbolTable', propertyName, propertyValue, ...)`

Imports symbol table from MAP file to specified DD Build object

Example

```
% Create a Build object MyBuild under Application object MyApplication for the platform TBTC1766/Task32
hDDBuild =dsdd_manage_build('Create',...
'Name','myBuild',...
'Application','MyApplication',...
'Board','TBTC1766',...
'Compiler','Task32');

% Import the symbol table
dsdd_manage_build('ImportSymbolTable',...
'Build',hDDBuild ,...
'MapFileName','ECUApplication.map')
```

dsdd_manage_build('Create', propertyName, propertyValue, ...)

Purpose

Creates a DD Build object with the specified name in the specified DD Application object tree as follows:

- Sets the platform-specific properties of the embedded `<Application>/<Build>/TargetInfo` object on the basis of the `TargetInfo.xml` file.
- Imports the descriptions of the base types from the `TargetConfig.xml` file into the `<Application>/<Build>/TargetInfo/BaseTypes` object.

DD Build objects created with this function can be used for A2L file generation. The location of the `TargetInfo.xml` and `TargetConfig.xml` files can be specified directly or by means of the compiler and evaluation board names.

Description

This command is part of the dsdd_manage_build function.

Syntax

```
[hBuild, targetInfo] = dsdd_manage_build('Create', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Name'`	Name of the DD Build object to create.
`'Application'`	Name of the parent DD Application object where to create the DD Build object. Default: Existing DD Application object, or 'Application' if there is more than one DD Application object.
`'Board'`	Name of the evaluation board. If the TargetInfo/TargetConfig locations are not specified, searches for the TargetInfo.xml file in <TL_InstRoot>\Matlab\Tl\ApplicationBuilder\BoardPackages\<Board>\<Compiler> and for the TargetConifg.xml file in <TL_InstRoot>\Matlab\Tl\TargetConfiguration\<ProcessorFamily>\<CompilerFamily>.
`'Compiler'`	Compiler abbreviation. If the TargetInfo/TargetConfig locations are not specified, searches for the TargetInfo.xml file in <TL_InstRoot>\Matlab\Tl\ApplicationBuilder\BoardPackages\<Board>\<Compiler> and for the TargetConifg.xml file in <TL_InstRoot>\Matlab\Tl\TargetConfiguration\<ProcessorFamily>\<CompilerFamily>.
`'TargetInfoDir'`	Path to the TargetInfo.xml file.
`'TargetConfigDir'`	Path to the TargetConfig.xml file.

Output parameters

The following output parameters are available:

Parameter	Description
`hBuild`	DD handle of created DD Build object
`targetInfo`	targetInfo structure contained in specified TargetInfo.xml file (optional)

dsdd_manage_build('ImportSymbolTable', propertyName, propertyValue, ...)

Purpose Imports symbol table from MAP file to specified DD Build object

Description This command is part of the dsdd_manage_build function.

Syntax

```
dsdd_manage_build('ImportSymbolTable', propertyName, propertyValue, ...)
```

Property value pairs Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Build'`	DD handle or path of the DD Build object.
`'MapFileName'`	Name of the linker map file to parse.
`'ParserConfigFile'`	Name of symbol table import's configuration XML file. Is located in <TL_InstRoot>\dsdd\SymbolTableParser\Parser and describes the parser to be used for symbol table extraction. Default: As obtained by this function.

dsdd_manage_project

Where to go from here

Information in this section

dsdd_manage_project

Purpose

Manages DD project. DD project files are always loaded into DD0.

Syntax overview

The following syntaxes are available:

`dsdd_manage_project('Open', projectFile)`

Opens the DD project file. The file is looked for on the MATLAB search path. If it does not exist, it is created from a DD template file which is selected by the user.

`dsdd_manage_project('GetProjectFile', simulinkSystem)`

Gets the name of the project file associated with the current model or from the global preferences

`dsdd_manage_project('SetProjectFile', projectFile, simulinkSystem)`

Sets the name of the project file associated with the current model

`dsdd_manage_project('Close', propertyName, propertyValue, ...)`

Closes DD project

`dsdd_manage_project('Save')`

Saves DD0 to DD project file

`dsdd_manage_project('SaveAs', projectFile, propertyName, propertyValue, ...)`

Saves DD0 to specified file. If no file is specified, the command opens a Save File dialog.
The new file becomes the current DD project file.

`dsdd_manage_project('MdlPostLoadFcn', simulinkSystem)`

Opens DD project file associated with specified model. This command is called in TargetLink models' PostLoadFcn callback.

`dsdd_manage_project('MdlPreSaveFcn', simulinkSystem)`

Saves DD project associated with model as specified with TargetLink's ProjectFileAutosave option. This command is called up in TargetLink models' PreSaveFcn callback.

`bSuccess = dsdd_manage_project('Check', projectFile)`

Validates the /Pool and /Config area of DD0, or the specified DD project file.

`dsdd_manage_project('ClearAll')`

Clears DD. Corresponds with DD MATLAB API ClearAll command.

`dsdd_manage_project('SaveCopyAs', projectFile)`

Saves DD0 to specified file in snapshot mode, which means that DD0 and all included subtrees are saved to one file.
The snapshot DD file does not become the current DD project file.

Example

```
% get the name of the project file associated with the current model, or from the global preferences
projectFile =dsdd_manage_project('GetProjectFile');

% open new DD project file - if the file does not exist, it is created from a
% template selected by the user
dsdd_manage_project('Open','myProject.dd');

% open file selection dialog and save DD contents to selected file
dsdd_manage_project('SaveAs');

% save DD contents to file ".\myproject.dd"; ask user if file should be overwritten
dsdd_manage_project('SaveAs','myproject');

% save DD contents to file ".\myproject.dd"; do not ask user if file should be overwritten
dsdd_manage_project('SaveAs','myproject','overwrite','on');

% save DD contents to file ".\myproject.dd"; overwrite file if it exists; cleanup DD
dsdd_manage_project('Close','myproject','Overwrite','on');

% load DD project file associated with model "mymodel" (must be open)
dsdd_manage_project('MdlPostLoadFcn','mymodel');

% save DD project file associated with model "mymodel" (must be open)
dsdd_manage_project('MdlPreSaveFcn','mymodel');
```

dsdd_manage_project('Check', projectFile)

Purpose

Validates the /Pool and /Config area of DD0, or the specified DD project file.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
bSuccess = dsdd_manage_project('Check', projectFile)
```

Input parameters

The following input parameters are available:

Parameter	Description
`projectFile`	Name of DD project file. The file is looked for on the current MATLAB search path.

Output parameters

The following output parameters are available:

Parameter	Description
`bSuccess`	1 on success, 0 on failure. Used only with the Check command.

dsdd_manage_project('ClearAll')

Purpose

Clears DD. Corresponds with DD MATLAB API ClearAll command.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('ClearAll')
```

dsdd_manage_project('Close', propertyName, propertyValue, ...)

Purpose

Closes DD project

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('Close', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Overwrite'`	If 'on', existing files are overwritten without notifying the user. Only used with the SaveAs and the SaveCopyAs commands.

dsdd_manage_project('GetProjectFile', simulinkSystem)

Purpose

Gets the name of the project file associated with the current model or from the global preferences

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('GetProjectFile', simulinkSystem)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simulinkSystem`	Simulink system

dsdd_manage_project('MdlPostLoadFcn', simulinkSystem)

Purpose

Opens DD project file associated with specified model. This command is called in TargetLink models' PostLoadFcn callback.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('MdlPostLoadFcn', simulinkSystem)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simulinkSystem`	Simulink system

dsdd_manage_project('MdlPreSaveFcn', simulinkSystem)

Purpose

Saves DD project associated with model as specified with TargetLink's ProjectFileAutosave option. This command is called up in TargetLink models' PreSaveFcn callback.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('MdlPreSaveFcn', simulinkSystem)
```

Input parameters

The following input parameters are available:

Parameter	Description
`simulinkSystem`	Simulink system

dsdd_manage_project('Open', projectFile)

Purpose

Opens the DD project file. The file is looked for on the MATLAB search path. If it does not exist, it is created from a DD template file which is selected by the user.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('Open', projectFile)
```

Input parameters

The following input parameters are available:

Parameter	Description
`projectFile`	Name of DD project file. The file is looked for on the current MATLAB search path.

dsdd_manage_project('Save')

Purpose

Saves DD0 to DD project file

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('Save')
```

dsdd_manage_project('SaveAs', projectFile, propertyName, propertyValue, ...)

Purpose

Saves DD0 to specified file. If no file is specified, the command opens a Save File dialog.

The new file becomes the current DD project file.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('SaveAs', projectFile, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Overwrite'`	If 'on', existing files are overwritten without notifying the user. Only used with the SaveAs and the SaveCopyAs commands.

Input parameters

The following input parameters are available:

Parameter	Description
`projectFile`	Name of DD project file. The file is looked for on the current MATLAB search path.

dsdd_manage_project('SaveCopyAs', projectFile)

Purpose

Saves DD0 to specified file in snapshot mode, which means that DD0 and all included subtrees are saved to one file.

The snapshot DD file does not become the current DD project file.

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('SaveCopyAs', projectFile)
```

Input parameters

The following input parameters are available:

Parameter	Description
`projectFile`	Name of DD project file. The file is looked for on the current MATLAB search path.

dsdd_manage_project('SetProjectFile', projectFile, simulinkSystem)

Purpose

Sets the name of the project file associated with the current model

Description

This command is part of the dsdd_manage_project function.

Syntax

```
dsdd_manage_project('SetProjectFile', projectFile, simulinkSystem)
```

Input parameters

The following input parameters are available:

Parameter	Description
`projectFile`	Name of DD project file. The file is looked for on the current MATLAB search path.
`simulinkSystem`	Simulink system

dsdd_validate

dsdd_validate

Purpose

Frontend to DD validation.

Description

This function starts validation for specified DD objects. If called without input arguments, all objects in DD0 are validated.

The validation result is printed to the MATLAB Command Window. Additionally, if any invalid objects have been found, the TargetLink Message Browser opens and displays the associated messages. This lets you review and modify the objects in the DD Manager.

VALIDATION LEVELS:

- 0 - In this level, nothing is validated. Writing level 0 to an object means tagging it as "not validated".

- 1 - The object is checked for invalid properties, which are properties that must not be associated with the object or have invalid values. Additionally, the program checks whether properties can have variants. If specified, invalid properties are deleted or set to correct values. If variants are not allowed, all variants but variant 0 are removed. If there is no variant 0, the first variant is kept and its ID set to 0. The validation level is set to 1 if no errors have been detected, or if all errors could be fixed.
- 2 - In addition to what is checked in level 1, the object's position in the DD is checked. An error is thrown if the object kind of the object's parent does not match the Data Model. However, the object will not be moved or deleted if this error comes up. Thus, the fix attribute is ignored for level 2.
- 3 - In addition to what is checked in levels 1 and 2, dependencies between properties are checked. For example, the Min and Max properties of a Variable object specify the lower and upper bounds for the variable. Level 3 also checks if Min < Max. As in level 2, the fix attribute is ignored for level 3 errors.
- 4 - The default validation level. In addition to what is checked in levels 1, 2 and 3, all non-empty reference properties are checked if they point at valid objects.

Syntax

```
info = dsdd_validate(DDObjects, propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`level`	Validation level Default: 4
`logfile`	File to log validation messages Default: 'dsdd_validate.log'
`verbose`	Prints more infos to MATLAB Command Window Default: 'on'

Input parameters

The following input parameters are available:

Parameter	Description
`DDObjects`	DD object path or handle of object which should be validated. All objects in the subtree defined by the specified object(s) are processed. More that one object can be specified by a cell array with DD object identifiers (DD handles, or paths).

Output parameters

The following output parameters are available:

Parameter	Description
`info`	Contains statistical data about the validated objects: ▪ .numOfValidatedObjects - Number of validated objects ▪ .numOfNotValidatedObjects - Number of objects which could not be validated because of missing read access ▪ .numOfInvalidObjects - Number of objects which do not comply with validation level ▪ .numOfFixedObjects - Number of objects which have successfully been fixed

Example

```
% level 3 validation of /Pool
dsdd_validate('/Pool');

% level 4 validation of /Config and /Pool
dsdd_validate({'/Config','/Pool'},'Level',4,'LogFile','validate.log');
```

dsddman

Where to go from here

Information in this section

dsddman

Purpose

Command-line interface to Data Dictionary Manager.

Syntax overview

The following syntaxes are available:

```
dsddman()
```

Opens the Data Dictionary Manager UI

```
dsddman('Open', file)
```

Opens a DD file and displays its content `isOpen = dsddman('IsGuiOpen')`

Checks whether the DD Manager UI is open

`dsddman('Refresh')`

Refreshes Data Dictionary Manager (except Model Browser and Difference Browser)

`dsddman('Select', objectIdentifier)`

Selects specified object and displays it

`selectedObjects = dsddman('GetSelected')`

Gets the selected object in the Data Dictionary Navigator

`dsddman('Edit', objectIdentifier)`

Opens dialog to edit specified object

`msgID = dsddman('AddMessage', propertyName, propertyValue, ...)`

Adds message to the Message Browser

`dsddman('DemandCustomOutputView', propertyName, propertyValue, ...)`

Creates a custom output view

`dsddman('AddCustomMessage', propertyName, propertyValue, ...)`

Adds a message to custom output view

`dsddman('ClearView', propertyName, propertyValue, ...)`

Clears the custom output view

`dsddman('CloseView', propertyName, propertyValue, ...)`

Closes the custom output view

`dsddman('Compare', propertyName, propertyValue, ...)`

Compares two DD project files or DD workspaces.

`errorCode = dsddman('ReloadMenuExtensionSpecification')`

Reloads the files that contain the specifications of the menu extensions into the Data Dictionary Manager

Example

```
% Create a difference pane with two DD objects.
dsddman('Compare','RefObject','//DD0/Pool','CorObject','//DD4/Pool')

% Create a difference pane with two DD files (the settings argument is optional).
dsddman('Compare','RefFile','pipt1_new.dd','CorFile','pipt1_old.dd','Settings','CompOptions.xml')
```

dsddman()

Purpose

Opens the Data Dictionary Manager UI

Description

This command is part of the dsddman function.

Syntax

```
dsddman()
```

dsddman('AddCustomMessage', propertyName, propertyValue, ...)

Purpose

Adds a message to custom output view

Description

This command is part of the dsddman function.

Syntax

```
dsddman('AddCustomMessage', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Name'`	Name of custom output view
`'Title'`	Message title
`'Message'`	Message text (must not be empty)
`'Reference'`	Object associated with message
`'RefKind'`	Reference's kind of object: ▪ slblock - Simulink block ▪ sfobject - Stateflow object ▪ ddobject - DD object ▪ file - File ▪ mxarray - MATLAB array
`'Tag'`	Tag (may be used to introduce message categories)

dsddman('AddMessage', propertyName, propertyValue, ...)

Purpose

Adds message to the Message Browser

Description

This command is part of the dsddman function.

Syntax

```
msgID = dsddman('AddMessage', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Title'`	Message title
`'Message'`	Message text (must not be empty)
`'Reference'`	Object associated with message
`'RefKind'`	Reference's kind of object: ▪ slblock - Simulink block ▪ sfobject - Stateflow object ▪ ddobject - DD object ▪ file - File ▪ mxarray - MATLAB array
`'Type'`	Severity of message: ▪ Info ▪ Warning ▪ Error
`'Id'`	ID of message that should be the parent of this message
`'Number'`	Error or info number of the message, e.g., 5040 for the following DD message: The specified object does not exist.

Output parameters

The following output parameters are available:

Parameter	Description
`msgID`	Message identifier

dsddman('ClearView', propertyName, propertyValue, ...)

Purpose

Clears the custom output view

Description

This command is part of the dsddman function.

Syntax

```
dsddman('ClearView', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Name'`	Name of custom output view

dsddman('CloseView', propertyName, propertyValue, ...)

Purpose

Closes the custom output view

Description

This command is part of the dsddman function.

Syntax

```
dsddman('CloseView', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Name'`	Name of custom output view

dsddman('Compare', propertyName, propertyValue, ...)

Purpose

Compares two DD project files or DD workspaces.

Description

This command is part of the dsddman function.

Syntax

```
dsddman('Compare', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'RefFile'`	Reference file.
`'RefObject'`	Referenced DD object.
`'CorFile'`	Corresponding file.
`'CorObject'`	Corresponding DD object.
`'Settings'`	Comparison settings file (created by Compare Tool Options dialog).

dsddman('DemandCustomOutputView', propertyName, propertyValue, ...)

Purpose

Creates a custom output view

Description

This command is part of the dsddman function.

Syntax

```
dsddman('DemandCustomOutputView', propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`'Name'`	Name of custom output view

dsddman('Edit', objectIdentifier)

Purpose

Opens dialog to edit specified object

Description

This command is part of the dsddman function.

Syntax

```
dsddman('Edit', objectIdentifier)
```

Input parameters

The following input parameters are available:

Parameter	Description
`objectIdentifier`	Specifies object to be processed

dsddman('GetSelected')

Purpose

Gets the selected object in the Data Dictionary Navigator

Description

This command is part of the dsddman function.

Syntax

```
selectedObjects = dsddman('GetSelected')
```

Output parameters

The following output parameters are available:

Parameter	Description
`selectedObjects`	Selected objects in the Data Dictionary Navigator

dsddman('IsGuiOpen')

Purpose

Checks whether the DD Manager UI is open

Description

This command is part of the dsddman function.

Syntax

```
isOpen = dsddman('IsGuiOpen')
```

Output parameters

The following output parameters are available:

Parameter	Description
`isOpen`	True if the Data Dictionary Manager is open

dsddman('Open', file)

Purpose

Opens a DD file and displays its content

Description

This command is part of the dsddman function.

Syntax

```
dsddman('Open', file)
```

Input parameters

The following input parameters are available:

Parameter	Description
`file`	DD file

dsddman('Refresh')

Purpose

Refreshes Data Dictionary Manager (except Model Browser and Difference Browser)

Description

This command is part of the dsddman function.

Syntax

```
dsddman('Refresh')
```

dsddman('Select', objectIdentifier)

Purpose

Selects specified object and displays it

Description

This command is part of the dsddman function.

Syntax

```
dsddman('Select', objectIdentifier)
```

Input parameters

The following input parameters are available:

Parameter	Description
`objectIdentifier`	Specifies object to be processed

dsddman('ReloadMenuExtensionSpecification')

Purpose

Reloads the files that contain the specifications of the menu extensions into the Data Dictionary Manager

Description

This command is part of the dsddman function.

Syntax

```
errorCode = dsddman('ReloadMenuExtensionSpecification')
```

Output parameters The following output parameters are available:

Parameter	Description
`errorCode`	error code (0 on success)

tlExecuteDDCustomCommand

tlExecuteDDCustomCommand

Purpose Executes a DD CustomCommand or CustomCommandGroup object.

Syntax

```
errorCode = tlExecuteDDCustomCommand(ddCmdObj, propertyName, propertyValue, ...)
```

Property value pairs Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`PrintToCommandWindow`	Prints messages on the execution progress in the MATLAB Command Window. Default: 'on'
`PrintToDDManager`	Prints messages on the execution progress in the Message Browser of the Data Dictionary Manager. Default: 'off'

Input parameters The following input parameters are available:

Parameter	Description
`ddCmdObj`	The handle of the DD CustomCommand or CustomCommandGroup object.

Output parameters

The following output parameters are available:

Parameter	Description
`errorCode`	0: No errors. 1: An error occurred.

Related topics

HowTos

How to Specify Custom Command Calls (📖 TargetLink Data Dictionary Basic Concepts Guide)

Handling Buses

tlSimulinkBusObject

tlSimulinkBusObject

Purpose

Processes Simulink.Bus objects in TargetLink.

Syntax overview

The following syntaxes are available:

```
busObjectName = tlSimulinkBusObject('CreateBusObject', hDDObj, propertyName, propertyValue, ...)
```

Creates a Simulink.Bus object from a DD object.

```
hDDObject = tlSimulinkBusObject('CreateDDVariable', busObject, propertyName, propertyValue, ...)
```

Creates DD Variable object from a Simulink.Bus object.

```
hDDObject = tlSimulinkBusObject('CreateDDTypedef', busObject, propertyName, propertyValue, ...)
```

Creates DD Typedef object from a Simulink.Bus object.

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`BusObjectName`	Name of the Simulink.Bus root object. default: name of hDDObj
`DDVariableName`	Name of the DD Variable object. default: name of the Simulink.Bus object
`DDTypedefName`	Name of the DD Typedef object. default: name of the Simulink.Bus object
`Recursive`	Specifies whether the Simulink.Bus will be traversed recursively. default: on
`Overwrite`	Specifies whether an existing Data Dictionary object will be overwritten. default: on

Input parameters

The following input parameters are available:

Parameter	Description
`hDDObj`	DD object identifier. Valid object kinds are those that have an embedded child object named 'components'.
`busObject`	Name of the Simulink.Bus object.

Output parameters

The following output parameters are available:

Parameter	Description
`busObjectName`	Name of the generated Simulink.Bus root object.
`hDDObject`	ID of the generated DD Variable object.

Related topics

Basics

Basics on Modeling Buses via Data Store Blocks (📖 TargetLink Preparation and Simulation Guide)

Finding Hook Scripts

tlFindHook

tlFindHook

Purpose

Searches for TargetLink hook scripts whose names match the specified pattern.

Description

This function searches in TargetLink configuration directories for TargetLink hook scripts whose names match the specified pattern. The configuration directories are:

- All directories returned by tl_get_config_path if tl_get_config_path.m exists in the current working directory or on the MATLAB's search path
- Current working directory and its 'config' subdirectory if tl_get_config_path.m does not exist in the current working directory and on the MATLAB's search path

Configuration directories that do not reside on the MATLAB's search path are ignored.

Syntax

```
[hookScriptList, hookScriptFullFileNameList] = tlFindHook(hookNamePattern)
```

Input parameters

The following input parameters are available:

Parameter	Description
`hookNamePattern`	Pattern for hook scripts names; may contain wildcards.

Output parameters

The following output parameters are available:

Parameter	Description
`hookScriptList`	Cell array with names of hook scripts whose names match the specified pattern.
`hookScriptFullFileNameList`	Cell array with the full file names of the hook scripts whose names match the specified pattern.

Example

```
% search for pre codegen hook scripts
hookFcnList =tlFindHook('*pre_codegen_hook*.m');
```

Remarks

To create tl_get_config_path.m, use tlCustomizationFiles('Create', 'tl_get_config_path', <path>).

Supporting MATLAB Code

Where to go from here

Information in this section

tlCreateMATLABFunctionDDObjects

tlCreateMATLABFunctionDDObjects

Purpose

Generates DD objects for specifying internal MATLAB code variables and MATLAB sub-functions.

Description

This function automatically creates all objects in the Data Dictionary that are required to specify internal MATLAB code variables and MATLAB sub-functions. It connects MATLAB Function blocks and Stateflow MATLAB functions in the model to their corresponding DD objects via the matlabfunction property. All actions that modified the Data Dictionary during one execution of the function are logged to a file.

Syntax

```
msgStruct = tlCreateMATLABFunctionDDObjects(propertyName, propertyValue, ...)
```

Property value pairs

Additional function parameters are expected as propertyName/propertyValue pairs, refer to the following table:

Property	Description
`XmlFiles`	List of XML files containing information to create the appropriate DD objects. These XML files are generated by the Code Generator if the OutputMATLABCodeInfo Code Generator option is active. The Code Generator uses the following pattern for the XML file names: MATLABCodeInfo_<subsystem ID>_<primary function name>.xml.

Property	Description
`Model`	Model name or handle. If no model is specified, the current model is used.
`AutoSave`	Specifies whether to automatically save modified models. This also applies to referenced models. The following values are possible: ▪ `'on'` - Automatically saves modified models. ▪ `'off'` - Modified models are not saved automatically. Default: 'off'
`AutoDelete`	Specifies whether to automatically delete DD objects that are no longer used for the execution of this function. The following values are possible: ▪ `'on'` - Automatically deletes obsolete DD objects. ▪ `'off'` - Generates a log file entry stating that obsolete objects exist. This option affects only DD objects in the following DD trees: ▪ /Pool/Variables/MATLAB_LocalVariables ▪ /Pool/Functions/MATLAB_Subfunctions Default: 'off'
`LogFile`	Name of the log file that contains actions performed by this function. Default: 'ddlookup.log'

Output parameters The following output parameters are available:

Parameter	Description
`msgStruct`	Structure with messages.

tl_get_mlfcnobjects

tl_get_mlfcnobjects

Purpose Compiles a list of MATLAB code model elements of a specified class.

Syntax

```
[hMLFcnObjList, modelElementTypes] = tl_get_mlfcnobjects(systemList, objClass)
```

Input parameters

The following input parameters are available:

Parameter	Description
`systemList`	Simulink systems to be searched for MATLAB code model elements.
`objClass`	Either a list of model element types or the string 'all' as a shortcut for all MATLAB code model element types known by TargetLink. Default: 'all'

Output parameters

The following output parameters are available:

Parameter	Description
`hMLFcnObjList`	Column vector with handles of the found MATLAB code model elements or an empty matrix if no objects were found.
`modelElementTypes`	Cell array containing the model element types.

Example

```
% get all MATLAB code model elements located in the block diagrams model1 and model2.
h =tl_get_mlfcnobjects({'model1','model2'});

% get inputs and outputs of MATLAB function blocks in myModel.
hMlFcnBlocks =tl_get_blocks('myModel','Stateflow')
[h, meTypes]=tl_get_mlfcnobjects(hMlFcnBlocks,{'MLFcnInput','MLFcnOutput'})
```

API Data Types

Description of the TargetLink API Data Types

Introduction

TargetLink API data types describe which values can be set for specified properties. Enum types describe which of a limited set of numeric values can be set to a property, and what each value means for TargetLink. Some properties can be set to strings which refer to objects in the **TargetLink Data Dictionary**. The DDRef types describe where in the Data Dictionary tree these objects are looked for. Special types like NameMacro and CIdentifier describe constraints which apply for possible values of properties so specified.

AccuWidthEnum

The table below describes the AccuWidthEnum API data type.

Value	String	Description
8	8 bit	This accu width is available for all targets. Note that the associated delay line and coefficient datatypes must comply.
16	16 bit	This accu width is available for all targets. Note that the associated delay line and coefficient datatypes must comply.
32	32 bit	This accu width is available for all targets. Note that the associated delay line and coefficient datatypes must comply.
48	48 bit	This accu width is available for SH2 targets only. Note that the associated delay line and coefficient datatypes must comply.
64	64 bit	This accu width is available for all targets. Note that the associated delay line and coefficient datatypes must comply.

AddFileModeEnum

The table below describes the AddFileModeEnum API data type.

Value	String	Description
1	Include to production code sourcefiles	The user-supplied code module will be included to the generated codefile(s).
2	Compile & link to production code application	The user-supplied code module will be compiled and linked to the production code application.

Value	String	Description
3	Include as system header file	The user-supplied code module will be included as a system header file to the generated codefile(s).

AlarmActivationKindEnum

The table below describes the AlarmActivationKindEnum API data type.

Value	String	Description
1	ABSOLUTE	SetAbsAlarm is used to start the cyclic alarms.
2	RELATIVE	SetRelAlarm is used to start the cyclic alarms.

AutosarModeEnum

The table below describes the AutosarModeEnum API data type.

Value	String	Description
1	Non-AUTOSAR	Enable standard blockspecific settings.
2	Classic	Enable blockspecific Classic AUTOSAR settings.
3	Adaptive	Enable blockspecific Adaptive AUTOSAR settings.

AutoscalingModeEnum

The table below describes the AutoscalingModeEnum API data type.

Value	String	Description
0	Calculate arbitrary scaling	The autoscaling tool will calculate scaling parameters with an arbitrary factor.
1	Calculate power-of-two scaling	The autoscaling tool will calculate scaling parameters with an factor equal to 2^n, where n is of type integer.
2	No autoscaling	The associated variable will be skipped by the autoscaling tool.

BitShiftDirectionEnum

The table below describes the BitShiftDirectionEnum API data type.

Value	String	Description
1	Left	Direction to shift the bits is left.
2	Right	Direction to shift the bits is right.
3	Bidirectional	Use positive integers for right shifts and negative integers for left shifts.

BitShiftNumberSourceEnum

The table below describes the BitShiftNumberSourceEnum API data type.

Value	String	Description
1	Dialog	The bits to shift are given by the property nbitshiftright.
2	Input port	The bits to shift are fed into the block via an additional inport. The property nbitshiftright is ignored.

Boolean

The table below describes the Boolean API data type.

Value	String	Description
1	on	Sets the associated Boolean property to "on".
0	off	Sets the associated Boolean property to "off".

BitsToExtractEnum

The table below describes the BitsToExtractEnum API data type.

Value	String	Description
1	Upper half	Extract the upper half of bits.
2	Lower half	Extract the lower half of bits.
3	Range starting with most significant bit	Extract a range of bits starting with most significant bit.
4	Range ending with least significant bit	Extract a range of bits ending with least significant bit.
5	Range of bits	Extract a range of bits.

BreakpointsDataSourceEnum

The table below describes the BreakpointsDataSourceEnum API data type.

Value	String	Description
1	Dialog	The breakpoint data is given by the property input.value.
2	Input port	The breakpoint data is passed in with an additional input port.

CodeCoverageEnum

The table below describes the CodeCoverageEnum API data type.

Value	String	Description
0	No code coverage	Disable code coverage tests.
1	Statement coverage (C0)	Enable statement coverage tests. All code blocks must be evaluated. Empty branches are not generated for conditional statements.
2	Decision coverage (C1)	Enable decision coverage tests. All branches of conditional statements must be evaluated, including empty branches.

CodeGenerationModeEnum

The table below describes the CodeGenerationModeEnum API data type.

Value	String	Description
1	Standard	Enable standard code generation.
2	Classic AUTOSAR	Enable code generation using blockspecific Classic AUTOSAR settings.
4	Adaptive AUTOSAR	Enable code generation using blockspecific Adaptive AUTOSAR settings.
3	RTOS	Enable multirate code generation if more than one sample time is specified.

CommentPlacementEnum The table below describes the CommentPlacementEnum API data type.

Value	String	Description
1	At block variable declaration	Place comment at variable declaration.
2	At block variable evaluation	Place comment at variable evaluation.

CommunicationKindEnum The table below describes the CommunicationKindEnum API data type.

Value	String	Description
1	Unspecified	The communication kind is unspecified.
2	Sender-Receiver	Sender-Receiver communication between software components by using well defined classic AUTOSAR ports and interfaces.
3	InterRunnable	InterRunnableVariable communication between runnables inside of a software component.
4	Client-Server	Client-Server communication between software components by using well defined classic AUTOSAR ports and interfaces.
5	Operation	Implementation or call of an operation.
6	OperationReturnValue	Operation has return value.
7	Mode-Switch	Reading and switching modes by using well defined classic AUTOSAR ports and interfaces.
8	Activation Reasons Argument	Activation reasons argument of runnables.
9	Port-Defined Argument	Port-Defined argument(s) of runnables.
10	NvData	NvData communication between software components by using well defined classic AUTOSAR ports and interfaces.
11	TransformerError	Generating transformer error parameter
12	DataElementUpdated	Check whether data element has changed after last read access.
13	Event	Event call of adaptive AUTOSAR service interface.
14	Field	Get field of adaptive AUTOSAR service interface.
15	PersistencyDatabase	PersistencyDataElement exchange of adaptive AUTOSAR PersistencyKeyValueDatabaseinterface.

CommunicationModeEnum The table below describes the CommunicationModeEnum API data type.

Value	String	Description
0	Implicit (IWrite)	The data is present at the start of the runnable an does not change during the runnable's execution.The send operation is done with the return of the runnable exactly once.
1	Implicit (IWriteRef)	The data is present at the start of the runnable an does not change during the runnable's execution.The send operation is done with the return of the runnable exactly once.
2	Explicit	The communication is done whenever the data is required or provided, respectively.

ConversionOutputEnum

The table below describes the ConversionOutputEnum API data type.

Value	String	Description
1	Signal copy	Outputs a copy of the incoming bus.
2	Virtual bus	Converts the input bus to a virtual bus.
3	Nonvirtual bus	Converts the input bus to a nonvirtual bus.

DataPortOrderEnum

The table below describes the DataPortOrderEnum API data type.

Value	String	Description
1	Zero-based contiguous	Use zero-based indexing for ordering contiguous data ports.
2	One-based contiguous	Use one-based indexing for ordering contiguous data ports.
3	Specify indices	Use noncontiguous indexing for ordering data ports.

DataStoreAccessEnum

The table below describes the DataStoreAccessEnum API data type.

Value	String	Description
1	rw	Read and write access are allowed.
2	r	Only read access is allowed.
3	w	Only write access is allowed.

DDRefAdaptiveAutosarFuncti on

Describes an adaptive AUTOSAR function. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/AdaptiveAutosarFunctions`.

DDRefAlarm

Describes an OSEK OS alarm. They reside in `/Pool/RTOS/Alarms`.

DDRefArField

Describes a field, a member of a service interface. They reside in `/Pool/Autosar/Interfaces/<ServiceInterface>`.

DDRefBlock

Describes a Simulink block. In the Subsystems area, Block objects describe blocks in the Simulink system that has been processed by the Code Generator. In the Pool area, Block objects can be used for DD-based code generation. They reside in `/Pool/ModelDesign/Blocks`.

DDRefClientPort

Describes an AUTOSAR require client-server port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.

DDRefDataElement	Describes a data element, e.g., a member of a sender/receiver interface. They reside in `/Pool/Autosar/Interfaces`.
DDRefEvent	Describes an operating system event. They reside in `/Pool/RTOS/Events`.
DDRefExchangeableWidth	Defines a set of variable widths used for width invariant code generation. They reside in `/Pool/ExchangeableWidths`.
DDRefFileSpecification	Describes attributes of files which are needed to build an application, e.g., generated code files, legacy code files, etc. They reside in `/Pool/Modules`.
DDRefFunctionClass	Defines C attributes of a function in the production code. They reside in `/Pool/FunctionClasses`.
DDRefInterRunnableVariable	Describes an AUTOSAR interrunnable variable. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/InterRunnableVariables`.
DDRefISR	Describes an interrupt service routine. They reside in `/Pool/RTOS/ISRs`.
DDRefMATLABFunction	Defines properties of a MATLAB function. They reside in `/Pool/MATLABFunctions`.
DDRefMessage	Describes a communication connection. In the Pool area, Message objects specify how communication connections should be generated. In the Subsystems area, Message object describe how communication connections have been implemented by the Code Generator. They reside in `/Pool/RTOS/Messages`.
DDRefModeElement	Describes an element of a sender/receiver interface which is used to communicate ECU modes to a software component. In AUTOSAR, mode elements are called ModeDeclarationGroupPrototypes. They reside in `/Pool/Autosar/Interfaces`.
DDRefModeReceiverPort	Describes an AUTOSAR mode receiver port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.

DDRefModeSenderPort	Describes an AUTOSAR mode sender port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefModule	Specifies a module in the production code. They reside in `/Pool/Modules`.
DDRefNvDataElement	Describes an NvDataElement member of an NvDataInterface. They reside in `/Pool/Autosar/Interfaces`.
DDRefNvReceiverPort	Describes an AUTOSAR require NvData port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefNvSenderPort	Describes an AUTOSAR provide NvData port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefNvSenderReceiverPort	Describes an AUTOSAR provide require NvData port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefOperationArgument	Describes an argument of an operation. Operations are elements of client/server interfaces, that means functions which a client component may invoke on a server component. They reside in `/Pool/Autosar/Interfaces`.
DDRefOperation	Describes an operation. Operations are elements of client/server interfaces, that means functions which a client component may invoke on a server component. They reside in `/Pool/Autosar/Interfaces/<ClientServerInterface>/Operations`.
DDRefPersistencyDataElement	Specifies a piece of data that is subject to persistency in the context of the enclosing DD PersistencyKeyValueDatabaseInterface object. They reside in `/Pool/Autosar/Interfaces/<ServiceInterface>/DataElements`.
DDRefPersistencyProvideRequirePort	Describes an AUTOSAR provide require port whose type is determined by a DD PersistencyKeyValueDatabaseInterface object. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefPortDefinedArgument	A PortDefinedArgumentValue is passed to a RunnableEntity dealing with the ClientServerOperations provided by a given PortPrototype. Note that this is restricted to PortPrototypes of a ClientServerInterface. They reside

	in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports/<ServerPort>/PortDefinedArguments`.
DDRefPropertyValueList	List of property name/value pairs. All property values are strings. Additional PropertyValueList objects can be inserted to create a hierarchical composition. They reside in `/Pool/CodegenOptionSets`.
DDRefProtectionMechanism	Defines a mechanism used to protect code sections against preemption by tasks or ISRs. They reside in `/Pool/RTOS/ProtectionMechanisms`.
DDRefReceiverPort	Describes an AUTOSAR require sender-receiver port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefReplaceableDataItem	Specifies that a macro variable should be generated for a block variable. The definition of these macro variables can be generated in a separate code generation run independently of the current code generation unit. They reside in `/Pool/CodeGenerationUnitInterfaces/<CodeGenerationUnitInterface>/ReplaceableDataItems`.
DDRefRunnable	Describes an AUTOSAR runnable. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Runnables`.
DDRefScaling	Specifies scaling parameters. They reside in `/Pool/Scalings`.
DDRefSenderPort	Describes an AUTOSAR provide sender-receiver port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefServerPort	Describes an AUTOSAR provide client-server port. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports`.
DDRefServiceDiscovery	Specify settings for the service discovery process. They reside in `/Pool/Autosar/SoftwareComponents/<SoftwareComponent>/Ports/<ServiceRequirePort>`.
DDRefSignature	Describes the signature of a Simulink system. Can be used to have a Simulink system (model or subsystem) generated. They reside in `/Pool/ModelDesign/Signatures`.

DDRefSoftwareComponent

Describes an AUTOSAR software component and its internal behavior. They reside in `/Pool/Autosar/SoftwareComponents`.

DDRefTask

Describes an operating system task. They reside in `/Pool/RTOS/Tasks`.

DDRefTypedef

Specifies a data type. In the Pool area, Typedef objects serve as templates that describe how a data type should be generated. In the Subsystems area, Typedef objects describe how a data type was generated in the production code. They reside in `/Pool/Typedefs`.

DDRefVariableClass

Defines C attributes of variables in the production code. They reside in `/Pool/VariableClasses`.

DDRefVariable

Variable objects describe variables in production code. In the Pool area, Variable objects serve as templates that specify how a variable should be generated. By associating a TargetLink block variable (for example, a block's output) with a Variable object in the Pool area, the properties of the Variable object determine how the block variable should be represented in production code. In the Subsystems area, Variable objects describe how an instance of a variable was generated in production code. They reside in `/Pool/Variables`.

DefaultDataPortEnum

The table below describes the DefaultDataPortEnum API data type.

Value	String	Description
1	Last data port	Use the last data port.
2	Additional data port	Use the additional data port with a * label.

DelayLengthSourceEnum

The table below describes the DelayLengthSourceEnum API data type.

Value	String	Description
1	Dialog	The delay length is given by the property state.delaylength.
2	Input port	The delay length is fed into the block via an additional inport. The property state.delaylength is ignored.

DiagnosticForOORShiftEnum

The table below describes the DiagnosticForOORShiftEnum API data type.

Value	String	Description
1	None	Out-of-range will not be checked.
2	Warning	A warning message is issued, if value is out of range.
3	Error	An error occurs, if value is out of range.

DocumentationHookEnum The table below describes the DocumentationHookEnum API data type.

Value	String	Description
1	default	Allow TargetLink to determine the location automatically based on the AutoDocCustomization block placement.
2	Create(Cover)	Insert a cover page of the generated documentation.
3	Create(Entry)	Insert the additional information at the beginning of the generated documentation.
4	Overview(Entry)	Insert the additional information at the beginning of the Overview chapter.
5	Overview(ModelImage)	Insert the additional information within the Overview chapter, below the screen shot of the model.
6	Overview(ModelComment)	Insert the additional information within the Overview chapter, below the model comment.
7	Overview(Exit)	Insert the additional information at the end of the Overview chapter.
8	TlSubsystems(Entry)	Insert the additional information at the beginning of the TLSubsystems chapter.
9	TlSubsystems(TlSubsystemEntry)	Insert the additional information at the beginning of the chapter that documents a TargetLink subsystem.
10	TlSubsystems(TlSubsystemImage)	Insert the additional information within the chapter that documents the parent TargetLink subsystem, below the screen shot of the TargetLink subsystem.
11	TlSubsystems(TlSubsystemInfo)	Insert the additional information within the chapter that documents the parent TargetLink subsystem, below the general information such as generation date and code generation options.
12	TlSubsystems(FunctionHierarchy)	Insert the additional information within the chapter that documents the parent TargetLink subsystem, below the listed functions hierarchy.
13	TlSubsystems(SourceFiles)	Insert the additional information within the chapter that documents the parent TargetLink subsystem, below the list of the generated source files.
14	TlSubsystems(BlockStatistics)	Insert the additional information within the chapter that documents the parent TargetLink subsystem, below the list of TargetLink blocks.
15	TlSubsystems(TypeInfo)	Insert the additional information within the chapter that documents the parent TargetLink subsystem, below the list of the used data types, variable and function classes that are used.
16	TlSubsystems(FunctionEntry)	Insert the additional information at the beginning of the chapter that documents the step function.
17	TlSubsystems(FunctionImage)	Insert the additional information within the chapter that documents the step function, below the screen shot of the function subsystem.
18	TlSubsystems(FunctionComment)	Insert the additional information within the chapter that documents the step function, below the function comment.
19	TlSubsystems(FunctionInterface)	Insert the additional information within the chapter that documents the step function, below the function interface.
20	TlSubsystems(MeasurableVariables)	Insert the additional information within the chapter that documents the step function, below the list of measurable variables.
21	TlSubsystems(CalibratableVariables)	Insert the additional information within the chapter that documents the step function, below the list of calibratable variables.
22	TlSubsystems(MacroVariables)	Insert the additional information within the chapter that documents the step function, below the list of macros.
23	TlSubsystems(OtherVariables)	Insert the additional information within the chapter that documents the step function, below the list of other global variables.

Value	String	Description
24	TlSubsystems(1DTables)	Insert the additional information within the chapter that documents the step function, below the list of 1-D look-up table blocks.
25	TlSubsystems(2DTables)	Insert the additional information within the chapter that documents the step function, below the list of 2-D look-up table blocks.
26	TlSubsystems(InterpolationBlocks)	Insert the additional information within the chapter that documents the step function, below the list of interpolation blocks.
27	TlSubsystems(IndexSearchBlocks)	Insert the additional information within the chapter that documents the step function, below the list of index search blocks.
28	TlSubsystems(FunctionExit)	Insert the additional information at the end of the chapter that documents the step function.
29	TlSubsystems(TlSubsystemExit)	Insert the additional information at the end of the chapter that documents a TargetLink subsystem.
30	TlSubsystems(Exit)	Insert the additional information at the end of the TLSubsystems chapter.

ExternalResetDelayEnum

The table below describes the ExternalResetDelayEnum API data type.

Value	String	Description
1	None	No external reset
2	Rising	External reset with rising trigger edge
3	Falling	External reset with falling trigger edge
4	Either	External reset with either trigger edge
5	Level	External reset on level
6	Level hold	External reset when the reset signal is nonzero at the current time step

ExternalResetDTIEnum

The table below describes the ExternalResetDTIEnum API data type.

Value	String	Description
1	none	No external reset
2	rising	External reset with rising trigger edge
3	falling	External reset with falling trigger edge
4	either	External reset with either trigger edge
5	level	not supported value: level
6	sampled level	not supported value: sampled level

FunctionRoleEnum

The table below describes the FunctionRoleEnum API data type.

Value	String	Description
1	Runnable	The subsystem where the Function block resides is implemented as a runnable.
2	Operation call with Runnable implementation	The subsystem where the Function block resides is implemented as a operation call system which also implements a runnable.
3	Operation call (synchronous)	The subsystem where the Function block resides is implemented as an operation call system.

Value	String	Description
4	Operation call (asynchronous)	The subsystem where the Function block resides is implemented as an asynchronous operation call system.
5	Operation result provider	The subsystem where the Function block resides is implemented as an operation result system.
6	Adaptive AUTOSAR Function	The subsystem where the Function block resides is implemented as adaptive AUTOSAR system.
7	Method Behavior	The subsystem where the Function block resides is implemented as method implementation system.
8	Method Call	The subsystem where the Function block resides is implemented as method call system.

GlobalLoggingModeEnum

The table below describes the GlobalLoggingModeEnum API data type.

Value	String	Description
1	Log according to block data	Outputs and states will be logged according to block settings.
2	Log signal histories	All outputs and states will be logged. For large models, this will severely increase simulation time.
3	Log min/max values	The min/max values of all outputs and states will be logged. (This may be convenient for autoscaling).
4	Do not log anything	No variables will be logged in the model. (Execution time and stacksize measurement during target simulations are not affected by this setting).

IndexOptions2Enum

The table below describes the IndexOptions2Enum API data type.

Value	String	Description
0	not used	Number of dimension is 1.
1	Assign all	All elements are assigned.
2	Index vector (dialog)	The element indices are given via dialog.
3	Index vector (port)	The element indices are given via index port.
4	Starting index (dialog)	The starting index of the range of elements to be assigned is given via dialog.
5	Starting index (port)	The starting index of the range of elements to be assigned is given via index port.

IndexOptionsEnum

The table below describes the IndexOptionsEnum API data type.

Value	String	Description
1	Assign all	All elements are assigned.
2	Index vector (dialog)	The element indices are given via dialog.
3	Index vector (port)	The element indices are given via index port.
4	Starting index (dialog)	The starting index of the range of elements to be assigned is given via dialog.
5	Starting index (port)	The starting index of the range of elements to be assigned is given via index port.

IndexSearchEnum

The table below describes the IndexSearchEnum API data type.

Value	String	Description
1	Linear search, start low	Linear search, starting at the lower end (simple and fast for small tables)
2	Linear search, start high	Linear search, starting at the upper end (simple and fast for small tables)
3	Local search	Local search (complex code, but fast for particularly large tables if input signal is smooth)
4	Binary search	Binary search (complex code, but fast for particularly large tables)
5	Equidistant implementation	Equidistant implementation (no search needed, only start point, stepsize, number of points kept in memory)

InitialConditionSourceDelayEnum

The table below describes the InitialConditionSourceDelayEnum API data type.

Value	String	Description
1	Dialog	The initial condition of the state is given by the property state.value.
2	Input port	The initial condition of the state is passed in with an additional input port.

InitialConditionSourceDTIEnum

The table below describes the InitialConditionSourceDTIEnum API data type.

Value	String	Description
1	internal	The initial condition of the state is given by the property state.initial.
2	external	The initial condition of the state is fed into the block via an additional inport. The property state.initial is ignored.

InputSelectEnum

The table below describes the InputSelectEnum API data type.

Value	String	Description
1	Element	Output is a single data element. The number of input ports is equal with the number of table dimensions.
2	Vector	Output is a vector. The number of input ports is equal with the number of table dimensions minus 1.
3	2-D Matrix	Output is a matrix table. The number of input ports is equal with the number of table dimensions minus 1.

IntegrationMethodEnum

The table below describes the IntegrationMethodEnum API data type.

Value	String	Description
1	Integration: Forward Euler	Forward Euler method with `y(k) = y(k-1) + T * u(k-1)`
2	Integration: Backward Euler	Backward Euler method with `y(k) = y(k-1) + T * u(k)`
3	Integration: Trapezoidal	Tustin's method with `y(k) = y(k-1) + T/2 * (u(k) + u(k-1))`
4	Accumulation: Forward Euler	not supported value: Accumulation: Forward Euler
5	Accumulation: Backward Euler	not supported value: Accumulation: Backward Euler
6	Accumulation: Trapezoidal	not supported value: Accumulation: Trapezoidal

LoggingModeEnum

The table below describes the LoggingModeEnum API data type.

Value	String	Description
1	Signal history	The signal history will be logged by the TLDS.
2	Min / Max values	The minima/maxima will be logged by the TLDS.
3	None	No data will be logged by the TLDS.

LookupMethEnum

The table below describes the LookupMethEnum API data type.

Value	String	Description
1	Interpolation-Extrapolation	Find value by linear interpolation and extrapolate if the input exceeds the table boundaries.
2	Interpolation-Use End Values	Find value by linear interpolation and use end values if the input exceeds the table boundaries.
3	Use Input Nearest	Use value which is nearest to the current input.
4	Use Input Below	Use value which is nearest and below to the current input.
5	Use Input Above	Use value which is nearest and above to the current input.

ModelSavingEnum

The table below describes the ModelSavingEnum API data type.

Value	String	Description
1	Never	Do not save the model automatically before code generation starts.
2	Prompt	Inquire me about saving the model before code generations starts.
3	Always	Save the model automatically before code generation starts.

MultiplicationGainEnum

The table below describes the MultiplicationGainEnum API data type.

Value	String	Description
1	Element-wise(K.*u)	The block is in Element-wise mode, in which it operates on the individual numeric elements of any nonscalar inputs.
2	Matrix(K*u)	Multiplies gain and input with the input as the second operand.
3	Matrix(u*K)	Multiplies input and gain with the input as the first operand.
4	Matrix(K*u) (u vector)	Multiplies gain and input with the input as the second operand and it processes nonscalar inputs as matrices.

MultiplicationProductEnum

The table below describes the MultiplicationProductEnum API data type.

Value	String	Description
1	Element-wise(.*)	The block is in Element-wise mode, in which it operates on the individual numeric elements of any nonscalar inputs.
2	Matrix(*)	The block is in Matrix mode, in which it processes nonscalar inputs as matrices.

NameMacro

NameMacro properties can be set to strings which are C identifiers and may additionally contain TargetLink-specific name macros. These macros start with the $ character and are expanded during code generation, resulting in identifiers for variables, functions, and data types. For example, the $B macro is expanded to the name of the block TargetLink block a variable is associated with.

OutOfRangeActionEnum

The table below describes the OutOfRangeActionEnum API data type.

Value	String	Description
1	None	No action if input is out of range.
2	Warning	Emit warning if input is out of range (this setting has no impact on generated prodcution code).
3	Error	Throw error if input is out of range (this setting has no impact on generated prodcution code).

OutputModeEnum

The table below describes the OutputModeEnum API data type.

Value	String	Description
1	Output only the index	Only the index signal is written to the block output, i.e. the block output is a scalar integer value in the range of the output's datatype. In Simulink, the block's output datatype is uint32.
2	Separate output variables for index and fraction	Both the index and the fraction are written to the output as separate signals, i.e. the block output is a double vector with two elements where the first element is the index and the second element is the fraction.
3	Common output for index and fraction	The index and fraction are combined to a single output variable. The data format is determined by the configuration file tl_get_block_config. In Simulink, the block's output datatype is uint32.

OutputSourceEnum

The table below describes the OutputSourceEnum API data type.

Value	String	Description
1	Dialog	The initial output value is specified by the Initial output parameter on the dialog.
2	Input signal	The initial output value is inherited from the input signal.

OutputWhenDisabledEnum

The table below describes the OutputWhenDisabledEnum API data type.

Value	String	Description
1	held	Output is held when the subsystem is disabled.
2	reset	Output is reset to the value given by Initial output when the subsystem is disabled.

PreprocessorModeEnum

The table below describes the PreprocessorModeEnum API data type.

Value	String	Description
1	#if	The code section will start with a preprocessor "#if" directive, resulting in an #ifdef / #elif / #endif section.
2	#ifdef	The code section will start with a preprocessor "#ifdef" directive, resulting in an #ifdef / #endifdef section

Value	String	Description
3	#if defined	The code section will start with a preprocessor "#if defined" directive, resulting in an #if defined / #elif / #endif section.

RefModelLoggingModeEnum

The table below describes the RefModelLoggingModeEnum API data type.

Value	String	Description
1	Enable in all referenced models	Enables logging in all referenced models.
2	Disable in all referenced models	Disables Logging in all referenced models.
3	Enable in selected models	Enables logging only in referenced models that are selected for logging.

SampleTimeModeEnum

The table below describes the SampleTimeModeEnum API data type.

Value	String	Description
1	continuous	Continuous sample time.
2	inherited	Inherit sample time from the driving block.

SearchEnum

The table below describes the SearchEnum API data type.

Value	String	Description
1	Linear search, start low	Linear search, starting at the lower end (simple and fast for small tables)
2	Linear search, start high	Linear search, starting at the upper end (simple and fast for small tables)
3	Local search	Local search (complex code, but fast for particularly large tables if input signal is smooth)
4	Binary search	Binary search (complex code, but fast for particularly large tables)
5	Equidistant implementation	Equidistant implementation (no search needed, only start point, stepsize, number of points kept in memory)

SimModeEnum

The table below describes the SimModeEnum API data type.

Value	String	Description
1	Without frame	No simulation frame will be generated for the incrementally generated production code.
2	SIL frame	A Software-in-the-Loop frame will be generated for the incrementally generated production code.
3	PIL frame	A Processor-in-the-Loop frame will be generated for the incrementally generated production code.

TableSourceEnum

The table below describes the TableSourceEnum API data type.

Value	String	Description
1	Dialog	The table data is given by the property table.value.
2	Input port	The table data is passed in with an additional input port.

ThresholdCriteriaEnum

The table below describes the ThresholdCriteriaEnum API data type.

Value	String	Description
1	u2 >= Threshold	Pass through input 1 if input 2 >= Threshold, otherwise, pass through input 3
2	u2 > Threshold	Pass through input 1 if input 2 > Threshold, otherwise, pass through input 3
3	u2 ~= 0	Pass through input 1 if input 2 != 0, otherwise, pass through input 3

Custom Look-Up Functions (API)

Where to go from here

Information in this section

Information in other sections

Using Custom Look-Up Functions (📖 TargetLink Preparation and Simulation Guide)

Definition of Custom Look-Up Functions

Where to go from here

Information in this section

Definition of Custom Look-Up Functions

Purpose

To replace TargetLink look-up functions with custom look-up functions.

Syntax

```
lookup1d_script(blockData, cgData)
lookup2d_script(blockData, cgData)
indexsearch_script(blockData, cgData)
interpolation_script(blockData, cgData)
```

Result

TargetLink will use your custom look-up functions instead of the built-in functions for look-up tables (1-D and 2-D), index search and interpolation.

Description

The replacement of TargetLink's look-up functions by custom look-up functions is defined in the following user-written M files:These M files must be placed on the MATLAB search path. They are evaluated during code generation whenever a Look-Up Table, Look-Up Table (2-D), PreLook-Up Index Search, or Interpolation (n-D) using PreLook-Up block is found. The Code Generator invokes the script with two input arguments:

`lookup1d_script.m`	For Look-Up Table blocks
`lookup2d_script.m`	For Look-Up Table (2-D) blocks
`indexsearch_script.m`	For PreLook-Up Index Search blocks
`interpolation_script.m`	For Interpolation (n-D) using PreLook-Up blocks

blockData is a MATLAB struct containing the settings of the Look-Up Table or Look-Up Table (2-D) block. The fields of the `blockData` struct are identical to the property names used in the TargetLink API, for example, `blockData.output.name` for the name of the block output variable (see tl_get on page 78 and tl_set on page 87). In your script you can evaluate the `blockData` struct to select the appropriate look-up function and its parameters.

cgData is a MATLAB struct containing additional information about the current block provided by the Code Generator. Currently, only the field `cgData.blockHandle` is supported, which is the handle of the current Look-Up Table, Look-Up Table (2-D), PreLook-Up Index Search, or Interpolation (n-D) using PreLook-Up block. This handle can be used with the Simulink `get_param` and `set_param` commands as a general approach to get more information about the block or the Simulink model (refer to *get_param* and *set_param* in the *Simulink User's Guide* by MathWorks®).

Tip

If your script for custom look-up functions needs additional configuration data, which is not available as standard TargetLink block data, you can attach this information to the Documentation page of the block dialog. The contents of this user data section can be accessed with the API function tl_get_userdata on page 81.

tlscript subcommands

The `tlscript` utility is called in the M file named `lookup1d_script/lookup2d_script/indexsearch_script/interpolation_script` and defines custom table data structures and look-up functions. Two subcommands are supported for `tlscript`:

-
  ```
  tlscript('TemplateCode', <code>);
  ```

  definition of the template code for custom look-up functions (refer to Template Code on page 338)

-
  ```
  tlscript('Set', <object_name>, ...
           'Property', <value>,  ...);
  ```

  specification of attributes for objects in the template code (refer to Variable Initialization and Setting Attributes on page 341)

Related topics

References

Template Code

Purpose

To identify the necessary header files, variables, data structures and prototypes of your look-up function.

Result

The Code Generator parses the template code and reads the necessary information to embed custom look-up functions in the generated code.

Description

The template code is written like C code and is passed to the Code Generator with the command `tlscript('TemplateCode', <code>)`, where `<code>` is a MATLAB cell array of strings containing the desired C code lines.

Tip

For examples of the definition of template code for custom look-up functions, refer to Basics on Customizing Table Maps (📖 TargetLink Preparation and Simulation Guide) or to the look-up table demo in `<DocumentsFolder>\dSPACE\TargetLink\<Version>\Demos\table`.

Contents of the template code

You can include the following statements in the template code:

#include The specified header file will be included in each generated source file containing code for custom look-up functions.

```
...
   '/* header file for lookup functions */'
   '#include "tab_fcn.h"'
...
```

Note

Multiple `#include` statements are possible. Other C preprocessor statements, for example, `#ifdef` or `#define`, are not supported in the template code. The C code file containing the actual implementations of the look-up functions can be linked to the generated code with the **Addfile Block** (📖 TargetLink Model Element Reference). Alternatively, if you want to avoid an extra block in the model, you can use a DD module object, refer to How to Build and Link External Modules via File Specifications (📖 TargetLink Customization and Optimization Guide)).

typedef Allows you to create type definitions for data structures, for example, structs with table data. Nested struct statements within type definitions are not allowed. Instead, nested structures can be implemented by creating a separate `typedef` for the substruct and using it as the type of a field in the parent struct.

```
...
   '/* typedef for 1D table struct */'
   'typedef struct {'
   '   UInt8  size;        /* Number of x,y pairs */'
   '   Int16  *xval;       /* Pointer to start of x array */'
   '   Int16  *yval;       /* Pointer to start of y array */'
   '} TAB1D_S16_tp;'
   ''
...
```

Variables, pointers, and struct definitions These objects cannot be initialized in the template code. Initializations have to be specified via the `tlscript('Set',...)` command. Variables may be scalars, vectors or matrices. Up to two dimensions are possible for variables. Pointers and structs have to be scalars.

```
...
'UInt16 xval[];          /* x array */          '
'UInt16 yval[];          /* y array */          '
'UInt8 fixed_vec[2];     /* array with two elements*/'
'const TAB1D_U16_tp map; /* 1D table struct */  '
...
```

Function prototypes For each custom look-up function a prototype must be defined in the template code. For example:

```
...
'/* 1D lookup function */'
'extern UInt16 Tab1D_U16('
'   TAB1D_U16_tp *map,     /* pointer to table struct */'
'   UInt16        x        /* x input */'
');'
''
...
```

Output computation A statement with instructions to the Code Generator on computing the block output (function results). Only function calls are supported in the output code. Arithmetic operations are not possible. You may call multiple functions in the output code, for example, to compute intermediate results of index search functions. The output computation must be the last part of the template code. For example:

```
'/* evaluate 1D lookup function (replaces dSPACE Tab1D...) */'
'out1 = Tab1D_U16(&map,in1);'
```

In this statement the following predefined variables can be referenced depending on the block type:

Block Type	**Object**	**Description**
Look-Up Table PreLook-Up Index Search Interpolation (n-D) using PreLook-Up (1-dimensional)	`out1` `in1`	output signal input signal

Look-Up Table (2-D)
Interpolation (n-D) using PreLook-Up (2-dimensional)
	`out1`
	`in1`
	`in2`
		output signal
		first input signal (row)
		second input signal (column)

Block Type	Object	Description

Call of user-written function in Stateflow action language	`out1`
	`arg1`
	`argN`
		function result
		first function input argument
		n-th function input argument

Related topics

Basics

Basics on Customizing Table Maps (📖 TargetLink Preparation and Simulation Guide)

Variable Initialization and Setting Attributes

Where to go from here

Information in this section

Variable Initialization and Setting Attributes

Syntax

```
tlscript('Set', <object_name>, ...
         'Property', <value>,  ...);
```

Description

This command is used to define all instance-specific object properties that do not result directly from the template code, such as variable classes, scaling factors, initial values, etc.

Object names

- For simple variables the `<object_name>` to be used with the `Set` command results directly from the variable name in the template code. For example, if the template code contains the line `Int16 xval[]`, the corresponding object name is `xval`.
- Components of structures are referenced with the dot notation used in the C language:

  `<object_name> = <struct_name>.<component_name>`

  For example, if properties should be set for the component `nx` of the structure `map`, then use

  `tlscript('Set','map.nx', ...);`
- Function arguments are referenced with the notation

  `<object_name> = <function_name>(<argument_name>)`

  For example:

  `tlscript('Set','lookup_fcn(x)', ...);`

Tip

To obtain a list of all objects in the template code, you can call the following commands:

- `test_lookup1d_script`
- `test_lookup2d_script`
- `test_indexsearch_script`
- `test_interpolation_script`.

Default values for properties

Default values are assigned to all properties when an object is created in the template code. It is only necessary to call `tlscript('Set', ...)` for properties that differ from the default. The default values are the same as in TargetLink's dialog boxes: for example, `'Lsb'=1`, `'Offset'=0`, `'Class'='default'`, etc.

Permissible properties and keywords

Depending on the kind of object, `tlscript` supports the following sets of properties:

- Permissible Properties for Variables on page 343
- Permissible Properties for Pointers on page 345
- Permissible Properties for Structs on page 346
- Permissible Properties for Functions on page 347

To describe the contents of a variable or the destination of a pointer for a calibration system, `tlscript` supports certain ASAM MCD-2 MC keywords:

- Keywords for Variables and Pointers on page 348

Related topics

References

Permissible Properties for Variables

Variable properties

The following table lists the properties that can be set for variables.

Property Name	Description
`'Variable'`	References a Data Dictionary variable.
`'Name'`	Variable name in the generated C code. This name can be constructed via name macros (see Basics on Using Name Macros (📖 TargetLink Customization and Optimization Guide)). Default: `$S_$B_<objName>`, i.e., subsystem ID, block name and the object name in the template code.
`'Description'`	Variable description string (comment) for the generated code and ASAM MCD-2 MC file. Default: as specified in the template code.
`'Type'`	Data type of the variable (basic or user-defined). Default: as specified in the template code.
`'Class'`	Variable class name, e.g., `CAL` for calibratable variables. Default: the most efficient implementation (the variable might even be eliminated later on by interblock optimization).
`'TypePrefix'`	A string which is placed at the beginning of the code line with the type definition of the variable, for example, '`far`' or '`near`'. The type prefix must not conflict with the type prefix specified in the variable class definition.
`'Address'`	Fixed program address as a string, e.g., '`0x4711'`, or `'&other_var'`
`'Width'`	Dimension of the variable (empty matrix for scalar variables). If `Value` is specified, `Width` is set automatically. `Width` can also be predefined in the template cotlscriptde, e.g., `UInt8 var[4];`. Up to two dimensions are possible. For example:`width=[]` scalar variable`width=3` vector of 3 elements`width=[4 5]` matrix with 4 x 5 elements
`'Value'`	Value(s) of the variable (floating-point; scalar, vector or matrix). The variable value must match the specified width.
`'LSB'`	Scaling factor(s) for conversion from floating-point to integer representation. `LSB` must be a scalar or have the same dimension as the variable value.
`'Offset'`	Offset value(s) for conversion from floating-point to integer representation. `Offset` must be a scalar or have the same dimension as the variable value.
`'ScalingAdjust'`	Enable/disable scaling adjust when assigning a fixed-point value to the variable. This flag is useful to specify functions which are independent of the scaling. `Default = '`yes`'`.
`'Min'`	Lower limit of the adjustable range of the variable as floating-point number(s). `Min` must be a scalar or have the same dimension as the variable value.
`'Max'`	Upper limit of the adjustable range of the variable as floating-point number(s). `Max` must be a scalar or have the same dimension as the variable value.
`'Unit'`	Physical unit of the variable (string) that appears as a comment in the generated code and ASAM MCD-2 MC file.
`'MergeVariable'`	Merge variable definitions, if a variable with the same name and attributes is already declared by another script. `Default = 'no'`.
`'Plotchannels'`	Elements of the variable to be plotted during simulations, e.g. [1 3] to plot the first and third element of the variable. Should only be used for block output variables.
`'Keywords'`	Keyword(s) to describe the contents of the variable, e.g., `AXIS_PTS_X` (see Keywords for Variables and Pointers on page 348 and example below). The `Keyword` is required mainly for ASAM MCD-2 MC file generation.

Property Name	Description
`'InheritDimensionFromInput'`	Specifies the table input signal (one-based index) which a specific variable inherits its dimensions from. This lets you implement, for example, a last index state variable for the Local search table search method. Refer to Basics on Using Custom Look-Up Functions (📖 TargetLink Preparation and Simulation Guide)

Example 1

```
tlscript('Set','xval',    ...
   'Class',     'CAL',     ...
   'Lsb',       2^-7,      ...
   'Offset',    -3.1415,   ...
   'Min',       -2.1,      ...
   'Max',       2.1,       ...
   'Unit',      'A',       ...
   'Value',     [-2:0.2:2] ...
   'Keywords'    {'AXIS_PTS_X' 'INDEX_INCR' 'ABSOLUTE'} ...
);
```

```
% store number of table points in element 'size'
% of struct 'map'
tlscript('Set',     'map.size',                  ...
         'Value',    length(blockData.input.value), ...
         'Class',    blockData.input.class,         ...
         'Keywords', 'NO_AXIS_PTS_X'                 ...
);
if ds_error_check, return; end
```

Example 2

The `lastIdx_x_$S_$B` variable inherits its dimension from the 1st table input signal:

```
tlscript('Set','lastIdx_x', ...
   'Name', 'lastIdx_x_$S_$B', ...
   'InheritDimensionFromInput', 1, ...
   );
if ds_error_check, return; end
```

Related topics

Basics

Basics on Using Name Macros (📖 TargetLink Customization and Optimization Guide)

References

Keywords for Variables and Pointers.......... 348

Permissible Properties for Pointers

Pointer properties

The following table lists the properties that can be set for pointers.

Property Name	Description
`'Name'`	Pointer name in the generated C code. This name can be constructed via name macros (see Basics on Using Name Macros (TargetLink Customization and Optimization Guide)). Default: `$S_$B_<objName>`, i.e., subsystem ID, block name and the template code's object name.
`'Description'`	Pointer description string (comment) for the generated code/ASAM MCD-2 MC file. Default: as specified in the template code.
`'Destination'`	Destination object of the pointer (variable, pointer or struct defined in the template code). For example: `x_array` reference to the variable x_array
`'Class'`	Variable class name, e.g., `GLOBAL` for a pointer with global scope. Default: the most efficient implementation (the pointer might even be eliminated later on by interblock optimization).
`'TypePrefix'`	A string which is placed at the beginning of the code line with the type definition of the pointer, for example, `'far'` or `'near'`. The type prefix must not conflict with the type prefix specified in the variable class definition.
`'Width'`	Dimension of the pointer must be an empty matrix because only scalar pointers are supported. For example: `width=[]` scalar pointer
`'MergeVariable'`	Merge pointer definitions, if a pointer with the same name and attributes is already declared by another script. `Default = 'no'`.
`'Keywords'`	Keyword(s) to describe the contents of the referenced variable, e.g., `AXIS_PTS_X` (see Keywords for Variables and Pointers on page 348 and example below). The `Keyword` is required mainly for ASAM MCD-2 MC file generation.
`'typecast'`	Insert a cast to the pointer type. Use this property if the destination of a pointer is of a different type.

Example

```
% set pointer to x-axis
tlscript('Set',          'map.xval',  ...
         'Destination', 'xval',      ...
         'Keywords',     'AXIS_PTS_X' ...
);
```

Related topics

Basics

Basics on Using Name Macros (TargetLink Customization and Optimization Guide)

References

Keywords for Variables and Pointers.. 348

Permissible Properties for Structs

Struct properties

The following table lists the properties that can be set for structs.

Property Name	Description	
'Variable'	References a Data Dictionary variable.	
'Name'	Struct name in the generated C code. This name can be constructed via name macros (see Generating Unique Names via Name Macros (TargetLink Customization and Optimization Guide)). Default: `$S_$B_<objName>`, i.e., subsystem ID, block name and the template code's object name.	
'Description'	Struct description string (comment) for the generated code and ASAM MCD-2 MC file. Default: as specified in the template code.	
'Class'	Variable class name, e.g., `GLOBAL` for a struct with global scope. Default: static global.	
'TypePrefix'	A string which is placed at the beginning of the code line with the type definition of the struct, for example, '`far`' or '`near`'. The type prefix must not conflict with the type prefix specified in the variable class definition.	
'Width'	Dimension of the struct; must be an empty matrix because only scalar structs are supported. For example: `width=[]` scalar struct	
'CreateTypedef'	= 'yes'	Creates a `typedef` for the struct.
	= 'no'	Does not generate a `typedef` (`typedef` must be supplied in an external header file, which is included with the ADDFILE block).
'MergeVariable'	Merge struct definitions, if a struct with the same name and attributes is already declared by another script. `Default = 'no'`.	
'Keywords'	Keyword(s) to describe the contents of the struct. Currently, the following `keywords` are recognized:	
	LOOKUP1D_STRUCT	data structure for 1-D look-up tables
	LOOKUP2D_STRUCT	data structure for 2-D look-up tables (if this keyword is set, the TargetLink Data Dictionary's ASAM MCD-2 MC export interface automatically creates appropriate RECORD_LAYOUTs).

Example

```
% set properties of table struct 'map'
tlscript('Set',            'map', ...
         'CreateTypedef', 'no',  ...
         'Keywords',       'LOOKUP1D_STRUCT' ...
);
```

Related topics

Basics

Generating Unique Names via Name Macros (TargetLink Customization and Optimization Guide)

Permissible Properties for Functions

Function properties

The following table lists the properties that can be set for functions.

Property Name	Description	
`'Name'`	Function name in the generated C code. This name can be constructed via name macros (see Basics on Using Name Macros (TargetLink Customization and Optimization Guide)). Default: object name in the template code, e.g., `my_tabfcn`.	
`'Description'`	Function description string (comment) for the generated code and ASAM MCD-2 MC file. Default: as specified in the template code.	
`'Type'`	Data type of the function result (basic or user-defined). Default: as specified in the template code's function prototype.	
`'TypePrefix'`	A string which is placed at the beginning of the code line with the definition of the function prototype, for example, `'far'` or `'near'`. The type prefix must not conflict with the type prefix specified in the function class definition.	
`'LSB'`	Scaling factor for conversion of the function result from floating-point to integer representation.	
`'Offset'`	Offset value for conversion of the function result from floating-point to integer representation.	
`'ScalingAdjust'`	Enable/disable scaling adjust when assigning the result to a fixed-point variable. This flag is useful to specify functions which are independent of the scaling. `Default = 'yes'`.	
`'Movable'`	`'yes'`	The custom look-up function call can be moved into conditional statements, there are no side effects.
	`'no'` (default)	The TargetLink optimizer assumes possible side effects and would not move the custom look-up function call into conditional statements.
`'FunctionClass'`	Function class name, e.g., `STATIC_FCN`.	
`'CreatePrototype'`	`'yes'`	Creates a prototype for the function.
	`'no'`	Does not generate a prototype (the prototype must be supplied in an external header file).
`'SampleTime'`	Sample time of the function as a positive floating-point number, or –1 for inherited sample time (default).	
`'Keywords'`	Keyword(s) to describe the contents of the function. Currently, the following keywords are recognized:	
	`LOOKUP1D_FUNCTION`	Function for 1-D look-up table
	`LOOKUP2D_FUNCTION`	Function for 2-D look-up table
	`INDEXSEARCH_FUNCTION`	Function for axis search
	`INTERPOLATION_FUNCTION`	Function for interpolation (n-D)

Note

It is recommended to specify one of the keywords listed above for each custom look-up function. The optimization algorithm which identifies common index search operations on look-up blocks is dependent on these keywords. Only functions marked with the `INDEXSEARCH_FUNCTION` keyword are combined.

Example

```
tlscript('Set', 'lookup_fcn', ...
         'CreatePrototype', 'no', ...
         'LSB', blockData.table.lsb, ...
         'Offset', blockData.table.offset, ...
         'Keywords', 'LOOKUP1D_FUNCTION' ...
);
```

Related topics

Basics

Basics on Using Name Macros (📖 TargetLink Customization and Optimization Guide)

Keywords for Variables and Pointers

Keywords

The following table lists the keywords that describe the contents of a variable or the destination of a pointer. These are the same keyword names as defined in the specification of the ASAM MCD-2 MC standard. They are recognized by the TargetLink Data Dictionary's ASAM MCD-2 MC export interface to generate appropriate record layouts for table structures. Some of the keywords can be combined with sub-keywords to provide a full description of a variable. You can also specify your own keywords to be written directly to the TargetLink Data Dictionary. The TargetLink Data Dictionary's ASAM MCD-2 MC export interface will ignore such user-specific keywords.

Keyword		Description
`FNC_VALUES`		Table values, i.e., the look-up results. Sub-keywords for 2-D look-up tables:
	`ROW_DIR`	Values of 2-D map deposited in rows
	`COLUMN_DIR`	Values of 2-D map deposited in columns
`IDENTIFICATION`		Placeholder for an identifier marking the table struct for the calibration system
`AXIS_PTS_X`		x-axis points, i.e., the row values for 2-D tables. Sub-keywords:
	`INDEX_INCR`	Increasing index of axis points with increasing address
	`INDEX_DECR`	Decreasing index of axis points with increasing address
	`ABSOLUTE`	Axis points deposited as absolute values
	`DIFFERENCE`	Axis points deposited as difference values
`AXIS_PTS_Y`		y-axis points, i.e., the column values for 2D tables. Sub-keywords:
	`INDEX_INCR`	Increasing index of axis points with increasing address
	`INDEX_DECR`	Decreasing index of axis points with increasing address
	`ABSOLUTE`	Axis points deposited as absolute values
	`DIFFERENCE`	Axis points deposited as difference values
`NO_AXIS_PTS_X`		Number of x-axis points. Do not use together with `FIX_NO_AXIS_PTS_X`.
`NO_AXIS_PTS_Y`		Number of y-axis points. Do not use together with `FIX_NO_AXIS_PTS_Y`.

Keyword	Description
FIX_NO_AXIS_PTS_X	Specifies that the number of x-axis points cannot be changed. Do not use together with NO_AXIS_PTS_X.
FIX_NO_AXIS_PTS_Y	Specifies that the number of y-axis points cannot be changed. Do not use together with NO_AXIS_PTS_Y.
SRC_ADDR_X	Pointer to the X input signal
SRC_ADDR_Y	Pointer to the Y input signal
SHIFT_OP_X	Shift operand to compute the x-axis points for an equidistant axis if the distance is a power of two
SHIFT_OP_Y	Shift operand to compute the y-axis points for an equidistant axis if the distance is a power of two
DIST_OP_X	Distance parameter to compute the x-axis points for an equidistant axis
DIST_OP_Y	Distance parameter to compute the y-axis points for an equidistant axis
OFFSET_X	Offset to compute the x-axis points for an equidistant axis
OFFSET_Y	Offset to compute the y-axis points for an equidistant axis
RIP_ADDR_X	Address of intermediate interpolation result for x-axis (see ASAM MCD-2 MC specification)
RIP_ADDR_Y	Address of intermediate interpolation result for y-axis (see ASAM MCD-2 MC specification)
RIP_ADDR_W	Address of final result of the internal ECU interpolation algorithm, i.e., table value
AXIS_RESCALE_X	Memory location of the rescale mapping for the x-axis
AXIS_RESCALE_Y	Memory location of the rescale mapping for the y-axis
NO_RESCALE_X	Memory location of the current number of rescale pairs for the x-axis
NO_RESCALE_Y	Memory location of the current number of rescale pairs for the y-axis
RESERVED	Placeholder for variables not covered by one of the keywords above (optional)