

Real-Time Workshop[®] Embedded Coder[™] 5 Reference



MATLAB[®]
& **SIMULINK[®]**

How to Contact The MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Real-Time Workshop® Embedded Coder™ Reference

© COPYRIGHT 2006–2009 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

September 2006 Online only
March 2007 Online only
September 2007 Online only
March 2008 Online only
October 2008 Online only
March 2009 Online only
September 2009 Online only

New for Version 4.5 (Release 2006b)
Revised for Version 4.6 (Release 2007a)
Revised for Version 5.0 (Release 2007b)
Revised for Version 5.1 (Release 2008a)
Revised for Version 5.2 (Release 2008b)
Revised for Version 5.3 (Release 2009a)
Revised for Version 5.4 (Release 2009b)

Function Reference

1

AUTOSAR	1-2
AUTOSAR Component Import	1-2
AUTOSAR Configuration	1-3
C++ Encapsulation Interface Control	1-6
Code Generation Objectives Customization	1-8
Code Generation Verification	1-8
Function Prototype Control	1-9
Model Entry Points	1-11
System Target File Callback Interface	1-12
Target Function Library Table Creation	1-13
Processor-in-the-Loop	1-15
Connectivity Configuration	1-15
Build	1-15
Execution Download, Start and Stop	1-16
Host and Target Communications	1-16
Host-Side Communications	1-16
Target-Side Communications	1-16

Class Reference

2

AUTOSAR	2-1
AUTOSAR Component Import	2-1
AUTOSAR Configuration	2-1
C++ Encapsulation Interface Control	2-2
Code Generation Objectives Customization	2-2
Code Generation Verification	2-2
Function Prototype Control	2-2

Alphabetical List

3

Block Reference

4

AUTOSAR Client-Server Communication	4-2
Configuration Wizards	4-3
Module Packaging	4-4

5

Configuration Parameters

6

Real-Time Workshop Pane: Code Style	6-2
Real-Time Workshop: Code Style Tab Overview	6-4
Parentheses level	6-5
Preserve operand order in expression	6-7
Preserve condition expression in if statement	6-8
Convert if-elseif-else patterns to switch-case statements ..	6-10
Preserve extern keyword in function declarations	6-12
Real-Time Workshop Pane: Templates	6-14
Real-Time Workshop: Templates Tab Overview	6-16
Code templates: Source file (*.c) template	6-17
Code templates: Header file (*.h) template	6-18
Data templates: Source file (*.c) template	6-19
Data templates: Header file (*.h) template	6-20
File customization template	6-21
Generate an example main program	6-22
Target operating system	6-24
Real-Time Workshop Pane: Data Placement	6-26
Real-Time Workshop: Data Placement Tab Overview	6-28
Data definition	6-29
Data definition filename	6-31
Data declaration	6-33
Data declaration filename	6-35
#include file delimiter	6-36
Module naming	6-37
Module name	6-39
Signal display level	6-41
Parameter tune level	6-43
Real-Time Workshop Pane: Data Type Replacement ..	6-45
Real-Time Workshop: Data Type Replacement Tab Overview	6-47

Replace data type names in the generated code	6-48
Replacement Name: double	6-50
Replacement Name: single	6-52
Replacement Name: int32	6-54
Replacement Name: int16	6-56
Replacement Name: int8	6-58
Replacement Name: uint32	6-60
Replacement Name: uint16	6-62
Replacement Name: uint8	6-64
Replacement Name: boolean	6-66
Replacement Name: int	6-68
Replacement Name: uint	6-70
Replacement Name: char	6-72
Real-Time Workshop Pane: Memory Sections	6-74
Real-Time Workshop: Memory Sections Tab Overview ...	6-76
Package	6-77
Refresh package list	6-79
Initialize/Terminate	6-80
Execution	6-81
Constants	6-82
Inputs/Outputs	6-84
Internal data	6-86
Parameters	6-88
Validation results	6-90
Real-Time Workshop Pane: AUTOSAR Code Generation	
Options	6-91
Real-Time Workshop: AUTOSAR Code Generation Options	
Tab Overview	6-92
Generate XML file from schema version	6-93
Configure AUTOSAR Interface	6-94
Parameter Reference	6-95
Recommended Settings Summary	6-95
Parameter Command-Line Information Summary	6-107

Index

Function Reference

AUTOSAR (p. 1-2)	Control AUTOSAR component configuration for import, code generation, and XML file export from Simulink® models
C++ Encapsulation Interface Control (p. 1-6)	Control C++ encapsulation interfaces in generated code for ERT-based Simulink models
Code Generation Objectives Customization (p. 1-8)	Control step function prototypes in generated code for ERT-based Simulink models
Code Generation Verification (p. 1-8)	Compare numerical equivalence of simulation and generated code results
Function Prototype Control (p. 1-9)	Control step function prototypes in generated code for ERT-based Simulink models
Model Entry Points (p. 1-11)	Access entry points in generated code for ERT-based Simulink models
System Target File Callback Interface (p. 1-12)	Control Real-Time Workshop® configuration options in callbacks for ERT-based custom targets
Target Function Library Table Creation (p. 1-13)	Create function replacement tables that make up Real-Time Workshop target function libraries (TFLs)
Processor-in-the-Loop (p. 1-15)	Control processor-in-the-loop (PIL) configuration

AUTOSAR

AUTOSAR Component Import (p. 1-2)	Control import of AUTOSAR components
AUTOSAR Configuration (p. 1-3)	Control and validate AUTOSAR configuration

AUTOSAR Component Import

<code>arxml.importer</code>	Construct <code>arxml.importer</code> object
<code>createCalibrationComponentObjects</code> (<code>arxml.importer</code>)	Create Simulink calibration objects from AUTOSAR calibration component
<code>createComponentAsModel</code> (<code>arxml.importer</code>)	Create AUTOSAR atomic software component as Simulink model
<code>createComponentAsSubsystem</code> (<code>arxml.importer</code>)	Create AUTOSAR atomic software component as Simulink atomic subsystem
<code>createOperationAsConfigurableSubsystem</code> (<code>arxml.importer</code>)	Create configurable Simulink subsystem library for client-server operation
<code>getCalibrationComponentNames</code> (<code>arxml.importer</code>)	Get calibration component names
<code>getComponentNames</code> (<code>arxml.importer</code>)	Get atomic software component names
<code>getDependencies</code> (<code>arxml.importer</code>)	Get list of XML dependency files
<code>getFile</code> (<code>arxml.importer</code>)	Return XML file name for <code>arxml.importer</code> object
<code>setDependencies</code> (<code>arxml.importer</code>)	Set XML file dependencies
<code>setFile</code> (<code>arxml.importer</code>)	Set XML file name for <code>arxml.importer</code> object

AUTOSAR Configuration

<code>addIOConf (RTW.AutosarInterface)</code>	Add AUTOSAR I/O configuration to model
<code>attachToModel (RTW.AutosarInterface)</code>	Attach <code>RTW.AutosarInterface</code> object to model
<code>getComponentName (RTW.AutosarInterface)</code>	Get XML component name
<code>getDataTypePackageName (RTW.AutosarInterface)</code>	Get XML data type package name
<code>getDefaultConf (RTW.AutosarInterface)</code>	Get default configuration
<code>getExecutionPeriod (RTW.AutosarInterface)</code>	Get runnable execution period
<code>getImplementationName (RTW.AutosarInterface)</code>	Get XML implementation name
<code>getInitEventName (RTW.AutosarInterface)</code>	Get initial event name
<code>getInitRunnableName (RTW.AutosarInterface)</code>	Get initial runnable name
<code>getInterfacePackageName (RTW.AutosarInterface)</code>	Get XML interface package name
<code>getInternalBehaviorName (RTW.AutosarInterface)</code>	Get XML internal behavior name
<code>getIOAutosarPortName (RTW.AutosarInterface)</code>	Get I/O AUTOSAR port name
<code>getIODataAccessMode (RTW.AutosarInterface)</code>	Get I/O data access mode
<code>getIODataElement (RTW.AutosarInterface)</code>	Get I/O data element name
<code>getIOErrorStatusReceiver (RTW.AutosarInterface)</code>	Get receiver port name

getIOInterfaceName (RTW.AutosarInterface)	Get I/O interface name
getIOPortNumber (RTW.AutosarInterface)	Get I/O AUTOSAR port number
getIOServiceInterface (RTW.AutosarInterface)	Get port I/O service interface
getIOServiceName (RTW.AutosarInterface)	Get port I/O service name
getIOServiceOperation (RTW.AutosarInterface)	Get port I/O service operation
getIsServerOperation (RTW.AutosarInterface)	Determine whether server is specified
getPeriodicEventName (RTW.AutosarInterface)	Get periodic event name
getPeriodicRunnableName (RTW.AutosarInterface)	Get periodic runnable name
getPortDefaultConf (RTW.AutosarInterface)	Get port default configuration
getServerInterfaceName (RTW.AutosarInterface)	Get name of server interface
getServerOperationPrototype (RTW.AutosarInterface)	Get server operation prototype
getServerPortName (RTW.AutosarInterface)	Get server port name
getServerType (RTW.AutosarInterface)	Determine server type
RTW.AutosarInterface	Construct RTW.AutosarInterface object
runValidation (RTW.AutosarInterface)	Validate RTW.AutosarInterface object against model
setComponentName (RTW.AutosarInterface)	Set XML component name

setInitEventName (RTW.AutosarInterface)	Set initial event name
setInitRunnableName (RTW.AutosarInterface)	Set initial runnable name
setIOAutosarPortName (RTW.AutosarInterface)	Set AUTOSAR port name
setIODataAccessMode (RTW.AutosarInterface)	Set I/O data access mode
setIODataElement (RTW.AutosarInterface)	Set I/O data element
setIOInterfaceName (RTW.AutosarInterface)	Set I/O interface name
setIsServerOperation (RTW.AutosarInterface)	Indicate that server is specified
setPeriodicEventName (RTW.AutosarInterface)	Set periodic event name
setPeriodicRunnableName (RTW.AutosarInterface)	Set periodic runnable name
setServerInterfaceName (RTW.AutosarInterface)	Set name of server interface
setServerOperationPrototype (RTW.AutosarInterface)	Specify operation prototype
setServerPortName (RTW.AutosarInterface)	Set server port name
setServerType (RTW.AutosarInterface)	Specify server type
syncWithModel (RTW.AutosarInterface)	Synchronize configuration with model

C++ Encapsulation Interface Control

<code>attachToModel</code> (<code>RTW.ModelCPPClass</code>)	Attach model-specific C++ encapsulation interface to loaded ERT-based Simulink model
<code>getArgCategory</code> (<code>RTW.ModelCPPArgsClass</code>)	Get argument category for Simulink model port from model-specific C++ encapsulation interface
<code>getArgName</code> (<code>RTW.ModelCPPArgsClass</code>)	Get argument name for Simulink model port from model-specific C++ encapsulation interface
<code>getArgPosition</code> (<code>RTW.ModelCPPArgsClass</code>)	Get argument position for Simulink model port from model-specific C++ encapsulation interface
<code>getArgQualifier</code> (<code>RTW.ModelCPPArgsClass</code>)	Get argument type qualifier for Simulink model port from model-specific C++ encapsulation interface
<code>getClassName</code> (<code>RTW.ModelCPPClass</code>)	Get class name from model-specific C++ encapsulation interface
<code>getDefaultConf</code> (<code>RTW.ModelCPPClass</code>)	Get default configuration information for model-specific C++ encapsulation interface from Simulink model
<code>getNumArgs</code> (<code>RTW.ModelCPPClass</code>)	Get number of step method arguments from model-specific C++ encapsulation interface
<code>getStepMethodName</code> (<code>RTW.ModelCPPClass</code>)	Get step method name from model-specific C++ encapsulation interface
<code>RTW.configSubsystemBuild</code>	Configure C function prototype or C++ encapsulation interface for right-click build of specified subsystem

<code>RTW.getEncapsulationInterfaceSpecificControl</code>	Handle to model-specific C++ encapsulation interface control object
<code>RTW.ModelCPPArgsClass</code>	Create C++ encapsulation interface object for configuring model class with I/O arguments style step method
<code>RTW.ModelCPPVoidClass</code>	Create C++ encapsulation interface object for configuring model class with void-void style step method
<code>runValidation</code> (<code>RTW.ModelCPPArgsClass</code>)	Validate model-specific C++ encapsulation interface against Simulink model
<code>runValidation</code> (<code>RTW.ModelCPPVoidClass</code>)	Validate model-specific C++ encapsulation interface against Simulink model
<code>setArgCategory</code> (<code>RTW.ModelCPPArgsClass</code>)	Set argument category for Simulink model port in model-specific C++ encapsulation interface
<code>setArgName</code> (<code>RTW.ModelCPPArgsClass</code>)	Set argument name for Simulink model port in model-specific C++ encapsulation interface
<code>setArgPosition</code> (<code>RTW.ModelCPPArgsClass</code>)	Set argument position for Simulink model port in model-specific C++ encapsulation interface
<code>setArgQualifier</code> (<code>RTW.ModelCPPArgsClass</code>)	Set argument type qualifier for Simulink model port in model-specific C++ encapsulation interface
<code>setClassName</code> (<code>RTW.ModelCPPClass</code>)	Set class name in model-specific C++ encapsulation interface
<code>setStepMethodName</code> (<code>RTW.ModelCPPClass</code>)	Set step method name in model-specific C++ encapsulation interface

Code Generation Objectives Customization

<code>addCheck</code> (<code>rtw.codegenObjectives.Objective</code>)	Add checks
<code>addParam</code> (<code>rtw.codegenObjectives.Objective</code>)	Add parameters
<code>excludeCheck</code> (<code>rtw.codegenObjectives.Objective</code>)	Exclude checks
<code>modifyInheritedParam</code> (<code>rtw.codegenObjectives.Objective</code>)	Modify inherited parameter values
<code>register</code> (<code>rtw.codegenObjectives.Objective</code>)	Register objective
<code>removeInheritedCheck</code> (<code>rtw.codegenObjectives.Objective</code>)	Remove inherited checks
<code>removeInheritedParam</code> (<code>rtw.codegenObjectives.Objective</code>)	Remove inherited parameters
<code>rtw.codegenObjectives.Objective</code>	Create custom code generation objectives
<code>setObjectiveName</code> (<code>rtw.codegenObjectives.Objective</code>)	Specify objective name

Code Generation Verification

<code>addCallback</code> (<code>cgv.CGV</code>)	Add callback function
<code>addConfigSet</code> (<code>cgv.CGV</code>)	Add configuration set
<code>addInputData</code> (<code>cgv.CGV</code>)	Add input data
<code>addPostLoadFiles</code> (<code>cgv.CGV</code>)	Add files required by model
<code>configModel</code> (<code>cgv.Config</code>)	Determine and change configuration parameter values
<code>displayReport</code> (<code>cgv.Config</code>)	Display results of comparing configuration parameter values

<code>getOutputData (cgv.CGV)</code>	Get output data
<code>getReportData (cgv.Config)</code>	Return results of comparing configuration parameter values
<code>run (cgv.CGV)</code>	Execute CGV object
<code>setOutputDir (cgv.CGV)</code>	Specify folder
<code>setOutputFile (cgv.CGV)</code>	Specify output data file name

Function Prototype Control

<code>addArgConf (RTW.ModelSpecificCPrototype)</code>	Add argument configuration information for Simulink model port to model-specific C function prototype
<code>attachToModel (RTW.ModelSpecificCPrototype)</code>	Attach model-specific C function prototype to loaded ERT-based Simulink model
<code>getArgCategory (RTW.ModelSpecificCPrototype)</code>	Get argument category for Simulink model port from model-specific C function prototype
<code>getArgName (RTW.ModelSpecificCPrototype)</code>	Get argument name for Simulink model port from model-specific C function prototype
<code>getArgPosition (RTW.ModelSpecificCPrototype)</code>	Get argument position for Simulink model port from model-specific C function prototype
<code>getArgQualifier (RTW.ModelSpecificCPrototype)</code>	Get argument type qualifier for Simulink model port from model-specific C function prototype
<code>getDefaultConf (RTW.ModelSpecificCPrototype)</code>	Get default configuration information for model-specific C function prototype from Simulink model

<code>getFunctionName</code> (<code>RTW.ModelSpecificCPrototype</code>)	Get function name from model-specific C function prototype
<code>getNumArgs</code> (<code>RTW.ModelSpecificCPrototype</code>)	Get number of function arguments from model-specific C function prototype
<code>getPreview</code> (<code>RTW.ModelSpecificCPrototype</code>)	Get model-specific C function prototype code preview
<code>RTW.configSubsystemBuild</code>	Configure C function prototype or C++ encapsulation interface for right-click build of specified subsystem
<code>RTW.getFunctionSpecification</code>	Get handle to model-specific C prototype function control object
<code>RTW.ModelSpecificCPrototype</code>	Create model-specific C prototype object
<code>runValidation</code> (<code>RTW.ModelSpecificCPrototype</code>)	Validate model-specific C function prototype against Simulink model
<code>setArgCategory</code> (<code>RTW.ModelSpecificCPrototype</code>)	Set argument category for Simulink model port in model-specific C function prototype
<code>setArgName</code> (<code>RTW.ModelSpecificCPrototype</code>)	Set argument name for Simulink model port in model-specific C function prototype
<code>setArgPosition</code> (<code>RTW.ModelSpecificCPrototype</code>)	Set argument position for Simulink model port in model-specific C function prototype
<code>setArgQualifier</code> (<code>RTW.ModelSpecificCPrototype</code>)	Set argument type qualifier for Simulink model port in model-specific C function prototype
<code>setFunctionName</code> (<code>RTW.ModelSpecificCPrototype</code>)	Set function name in model-specific C function prototype

Model Entry Points

<code>model_initialize</code>	Initialization entry point in generated code for ERT-based Simulink model
<code>model_SetEventsForThisBaseStep</code>	Set event flags for multirate, multitasking operation before calling <i>model_step</i> for ERT-based Simulink model — not generated as of Version 5.1 (R2008a)
<code>model_step</code>	Step routine entry point in generated code for ERT-based Simulink model
<code>model_terminate</code>	Termination entry point in generated code for ERT-based Simulink model

System Target File Callback Interface

slConfigUIGetVal	Return current value for custom target configuration option
slConfigUISetEnabled	Enable or disable custom target configuration option
slConfigUISetVal	Set value for custom target configuration option

Target Function Library Table Creation

<code>addAdditionalHeaderFile</code>	Add additional header file to array of additional header files for TFL table entry
<code>addAdditionalIncludePath</code>	Add additional include path to array of additional include paths for TFL table entry
<code>addAdditionalLinkObj</code>	Add additional link object to array of additional link objects for TFL table entry
<code>addAdditionalLinkObjPath</code>	Add additional link object path to array of additional link object paths for TFL table entry
<code>addAdditionalSourceFile</code>	Add additional source file to array of additional source files for TFL table entry
<code>addAdditionalSourcePath</code>	Add additional source path to array of additional source paths for TFL table entry
<code>addConceptualArg</code>	Add conceptual argument to array of conceptual arguments for TFL table entry
<code>addEntry</code>	Add table entry to collection of table entries registered in TFL table
<code>copyConceptualArgsToImplementation</code>	Copy conceptual argument specifications to matching implementation arguments for TFL table entry
<code>createAndAddConceptualArg</code>	Create conceptual argument from specified properties and add to conceptual arguments for TFL table entry

<code>createAndAddImplementationArg</code>	Create implementation argument from specified properties and add to implementation arguments for TFL table entry
<code>createAndSetCImplementationReturn</code>	Create implementation return argument from specified properties and add to implementation for TFL table entry
<code>getTflArgFromString</code>	Create TFL argument based on specified name and built-in data type
<code>registerCFunctionEntry</code>	Create TFL function entry based on specified parameters and register in TFL table
<code>registerCPromotableMacroEntry</code>	Create TFL promotable macro entry based on specified parameters and register in TFL table (for <code>abs</code> function replacement only)
<code>setReservedIdentifiers</code>	Register specified reserved identifiers to be associated with TFL table
<code>setTflCFunctionEntryParameters</code>	Set specified parameters for function entry in TFL table
<code>setTflCOperationEntryParameters</code>	Set specified parameters for operator entry in TFL table

Processor-in-the-Loop

Connectivity Configuration (p. 1-15)	Define processor-in-the-loop (PIL) configuration
Build (p. 1-15)	Configure PIL build process
Execution Download, Start and Stop (p. 1-16)	Control downloading, starting and resetting PIL executable on target hardware
Host and Target Communications (p. 1-16)	Configure host-target communications
Host-Side Communications (p. 1-16)	Configure host-side communications channel and drivers
Target-Side Communications (p. 1-16)	Configure target-side communications channel and drivers

Connectivity Configuration

<code>rtw.connectivity.ComponentArgs</code>	Provide parameters to each target connectivity component
<code>rtw.connectivity.Config</code>	Define connectivity implementation, comprising builder, launcher, and communicator components
<code>rtw.connectivity.ConfigRegistry</code>	Register connectivity configuration

Build

<code>rtw.connectivity.MakefileBuilder</code>	Configure makefile-based build process
---	--

Execution Download, Start and Stop

<code>rtw.connectivity.Launcher</code>	Control downloading, starting and resetting executable on target hardware
--	---

Host and Target Communications

<code>rtIOStreamClose</code>	Shut down communications channel with remote processor
<code>rtIOStreamOpen</code>	Initialize communications channel with remote processor
<code>rtIOStreamRecv</code>	Receive data from remote processor
<code>rtIOStreamSend</code>	Send data to remote processor

Host-Side Communications

<code>rtiostream_wrapper</code>	Test <code>rtiostream</code> shared library methods
<code>rtw.connectivity.RtIOStreamHost-Communicator</code>	Configure host-side communications

Target-Side Communications

<code>rtw.pil.RtIOStreamApplication-Framework</code>	Configure target-side communications
--	--------------------------------------

Class Reference

- “AUTOSAR” on page 2-1
- “C++ Encapsulation Interface Control” on page 2-2
- “Code Generation Objectives Customization” on page 2-2
- “Code Generation Verification” on page 2-2
- “Function Prototype Control” on page 2-2

AUTOSAR

In this section...
“AUTOSAR Component Import” on page 2-1
“AUTOSAR Configuration” on page 2-1

AUTOSAR Component Import

arxml.importer

Control import of AUTOSAR components

AUTOSAR Configuration

RTW.AutosarInterface

Control and validate AUTOSAR configuration

C++ Encapsulation Interface Control

RTW.ModelCPPArgsClass	Control C++ encapsulation interfaces for models using I/O arguments style step method
RTW.ModelCPPClass	Control C++ encapsulation interfaces for models
RTW.ModelCPPVoidClass	Control C++ encapsulation interfaces for models using void-void style step method

Code Generation Objectives Customization

rtw.codegenObjectives.Objective	Customize code generation objectives
---------------------------------	--------------------------------------

Code Generation Verification

cgv.CGV	Verify numerical equivalence of results
cgv.Config	Check and modify model configuration parameter values

Function Prototype Control

RTW.ModelSpecificCPrototype	Describe signatures of functions for model
-----------------------------	--

Alphabetical List

addAdditionalHeaderFile

Purpose	Add additional header file to array of additional header files for TFL table entry
Syntax	<code>void addAdditionalHeaderFile(<i>hEntry</i>, <i>headerFile</i>)</code>
Arguments	<i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = RTW.Tf1CFunctionEntry or <i>hEntry</i> = RTW.Tf1COperationEntry. <i>headerFile</i> String specifying an additional header file.
Description	The <code>addAdditionalHeaderFile</code> function adds a specified additional header file to the array of additional header files for a TFL table entry.
Example	<p>In the following example, the <code>addAdditionalHeaderFile</code> function is used along with <code>addAdditionalIncludePath</code>, <code>addAdditionalSourceFile</code>, and <code>addAdditionalSourcePath</code> to fully specify additional header and source files for a TFL table entry.</p> <pre>% Path to external header and source files libdir = fullfile('\${MATLAB_ROOT}','..', '..', 'lib'); op_entry = RTW.Tf1COperationEntry; . . . addAdditionalHeaderFile(op_entry, 'all_additions.h'); addAdditionalIncludePath(op_entry, fullfile(libdir, 'include')); addAdditionalSourceFile(op_entry, 'all_additions.c'); addAdditionalSourcePath(op_entry, fullfile(libdir, 'src'));</pre>
See Also	<code>addAdditionalIncludePath</code> , <code>addAdditionalSourceFile</code> , <code>addAdditionalSourcePath</code>

“Specifying Build Information for Function Replacements” in the Real-Time Workshop® Embedded Coder™ documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

addAdditionalIncludePath

Purpose	Add additional include path to array of additional include paths for TFL table entry
Syntax	<code>void addAdditionalIncludePath(<i>hEntry</i>, <i>path</i>)</code>
Arguments	<i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = RTW.Tf1CFunctionEntry or <i>hEntry</i> = RTW.Tf1COperationEntry. <i>path</i> String specifying the full path to an additional header file.
Description	The <code>addAdditionalIncludePath</code> function adds a specified additional include path to the array of additional include paths for a TFL table entry.
Example	<p>In the following example, the <code>addAdditionalIncludePath</code> function is used along with <code>addAdditionalHeaderFile</code>, <code>addAdditionalSourceFile</code>, and <code>addAdditionalSourcePath</code> to fully specify additional header and source files for a TFL table entry.</p> <pre>% Path to external header and source files libdir = fullfile('\$\{MATLAB_ROOT}', '..', '..', 'lib'); op_entry = RTW.Tf1COperationEntry; . . . addAdditionalHeaderFile(op_entry, 'all_additions.h'); addAdditionalIncludePath(op_entry, fullfile(libdir, 'include')); addAdditionalSourceFile(op_entry, 'all_additions.c'); addAdditionalSourcePath(op_entry, fullfile(libdir, 'src'));</pre>
See Also	<code>addAdditionalHeaderFile</code> , <code>addAdditionalSourceFile</code> , <code>addAdditionalSourcePath</code>

“Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

addAdditionalLinkObj

Purpose	Add additional link object to array of additional link objects for TFL table entry
Syntax	<code>void addAdditionalLinkObj(<i>hEntry</i>, <i>linkObj</i>)</code>
Arguments	<i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = RTW.Tf1CFunctionEntry or <i>hEntry</i> = RTW.Tf1COperationEntry. <i>linkObj</i> String specifying an additional link object.
Description	The <code>addAdditionalLinkObj</code> function adds a specified additional link object to the array of additional link objects for a TFL table entry.
Example	<p>In the following example, the <code>addAdditionalLinkObj</code> function is used along with <code>addAdditionalLinkObjPath</code> to fully specify an additional link object file for a TFL table entry.</p> <pre>% Path to external object files libdir = fullfile('\$\{MATLAB_ROOT}\', '..', '..\', 'lib'); op_entry = RTW.Tf1COperationEntry; ... addAdditionalLinkObj(op_entry, 'addition.o'); addAdditionalLinkObjPath(op_entry, fullfile(libdir, 'bin'));</pre>
See Also	<code>addAdditionalLinkObjPath</code> “Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation “Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

Purpose	Add additional link object path to array of additional link object paths for TFL table entry
Syntax	<code>void addAdditionalLinkObjPath(<i>hEntry</i>, <i>path</i>)</code>
Arguments	<p><i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = <code>RTW.Tf1CFunctionEntry</code> or <i>hEntry</i> = <code>RTW.Tf1COperationEntry</code>.</p> <p><i>path</i> String specifying the full path to an additional link object.</p>
Description	The <code>addAdditionalLinkObjPath</code> function adds a specified additional link object path to the array of additional link object paths for a TFL table entry.
Example	<p>In the following example, the <code>addAdditionalLinkObjPath</code> function is used along with <code>addAdditionalLinkObj</code> to fully specify an additional link object file for a TFL table entry.</p> <pre>% Path to external object files libdir = fullfile('\$MATLAB_ROOT','..', '..', 'lib'); op_entry = RTW.Tf1COperationEntry; ... addAdditionalLinkObj(op_entry, 'addition.o'); addAdditionalLinkObjPath(op_entry, fullfile(libdir, 'bin'));</pre>
See Also	<p><code>addAdditionalLinkObj</code></p> <p>“Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation</p> <p>“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation</p>

addAdditionalSourceFile

- Purpose** Add additional source file to array of additional source files for TFL table entry
- Syntax** `void addAdditionalSourceFile(hEntry, sourceFile)`
- Arguments**
- hEntry*
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry or *hEntry* = RTW.Tf1COperationEntry.
 - sourceFile*
String specifying an additional source file.

Description The `addAdditionalSourceFile` function adds a specified additional source file to the array of additional source files for a TFL table entry.

Example In the following example, the `addAdditionalSourceFile` function is used along with `addAdditionalHeaderFile`, `addAdditionalIncludePath`, and `addAdditionalSourcePath` to fully specify additional header and source files for a TFL table entry.

```
% Path to external header and source files
libdir = fullfile('${MATLAB_ROOT}','..', '..', 'lib');

op_entry = RTW.Tf1COperationEntry;
.
.
.
addAdditionalHeaderFile(op_entry, 'all_additions.h');
addAdditionalIncludePath(op_entry, fullfile(libdir, 'include'));

addAdditionalSourceFile(op_entry, 'all_additions.c');
addAdditionalSourcePath(op_entry, fullfile(libdir, 'src'));
```

See Also `addAdditionalHeaderFile`, `addAdditionalIncludePath`, `addAdditionalSourcePath`

“Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

addAdditionalSourcePath

Purpose Add additional source path to array of additional source paths for TFL table entry

Syntax `void addAdditionalSourcePath(hEntry, path)`

Arguments

hEntry
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry or *hEntry* = RTW.Tf1COperationEntry.

path
String specifying the full path to an additional source file.

Description The `addAdditionalSourcePath` function adds a specified additional source file path to the array of additional source file paths for a TFL table.

Example In the following example, the `addAdditionalSourcePath` function is used along with `addAdditionalHeaderFile`, `addAdditionalIncludePath`, and `addAdditionalSourceFile` to fully specify additional header and source files for a TFL table entry.

```
% Path to external header and source files
libdir = fullfile('${MATLAB_ROOT}','..', '..', 'lib');

op_entry = RTW.Tf1COperationEntry;
.
.
.
addAdditionalHeaderFile(op_entry, 'all_additions.h');
addAdditionalIncludePath(op_entry, fullfile(libdir, 'include'));

addAdditionalSourceFile(op_entry, 'all_additions.c');
addAdditionalSourcePath(op_entry, fullfile(libdir, 'src'));
```

See Also `addAdditionalHeaderFile`, `addAdditionalIncludePath`, `addAdditionalSourceFile`

“Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.addArgConf

Purpose Add argument configuration information for Simulink model port to model-specific C function prototype

Syntax `addArgConf(obj, portName, category, argName, qualifier)`

Description `addArgConf(obj, portName, category, argName, qualifier)` method adds argument configuration information for a port in your ERT-based Simulink model to a model-specific C function prototype. You specify the name of the model port, the argument category ('Value' or 'Pointer'), the argument name, and the argument type qualifier (for example, 'const').

The order of `addArgConf` calls determines the argument position for the port in the function prototype, unless you change the order by other means, such as the `RTW.ModelSpecificCPrototype.setArgPosition` method.

If a port has an existing argument configuration, subsequent calls to `addArgConf` with the same port name overwrite the previous argument configuration of the port.

Inputs	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .
	<i>portName</i>	String specifying the unqualified name of an inport or outport in your Simulink model.
	<i>category</i>	String specifying the argument category, either 'Value' or 'Pointer'.
	<i>argName</i>	String specifying a valid C identifier.
	<i>qualifier</i>	String specifying the argument type qualifier: 'none', 'const', 'const *', or 'const * const'.

Examples

In the following example, you use the `addArgConf` method to add argument configuration information for ports `Input` and `Output` in an ERT-based version of `rtwdemo_counter`. After executing these commands, click the **Configure Model Functions** button on the **Interface** pane of the Configuration Parameters dialog box to open the Model Interface dialog box and confirm that the `addArgConf` commands succeeded.

```
rtwdemo_counter
set_param(gcs,'SystemTargetFile','ert.tlc')

%% Create a function control object
a=RTW.ModelSpecificCPrototype

%% Add argument configuration information for Input and Output ports
addArgConf(a,'Input','Pointer','inputArg','const *')
addArgConf(a,'Output','Pointer','outputArg','none')

%% Attach the function control object to the model
attachToModel(a,gcs)
```

Alternatives

You can specify the argument configuration information in the Model Interface dialog box. See “Configuring Model Function Prototypes” in the Real-Time Workshop Embedded Coder documentation.

See Also

`RTW.ModelSpecificCPrototype.attachToModel`
“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

cgv.CGV.addCallback

Purpose Add callback function

Syntax `cgvObj.addCallback(CallbackFcn)`

Description `cgvObj.addCallback(CallbackFcn)` is an optional method that adds a callback function to the object. `cgvObj` is a handle to a `cgv.CGV` object. Before executing the model, for each input data, the object loads the input data (that you added using `addInputData`) and then calls the callback function.

The declaration of the callback function must receive the following parameters:

```
CallbackFcn(InputIndex, model_name, componentType,  
            connectivity)
```

`model_name`, `componentType`, `connectivity` are identical to the arguments that you specified in the `cgvObj` constructor. `InputIndex` is the index of the input data that the callback function executes.

You can specify only one callback for each object.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”
- “Using Callback Functions”

Purpose	Add checks				
Syntax	<code>addCheck(obj, checkID)</code>				
Description	<code>addCheck(obj, checkID)</code> includes the check, <code>checkID</code> , in the Code Generation Advisor. When a user selects the objective, the Code Generation Advisor includes the check, unless another objective with a higher priority excludes the check.				
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a code generation objective object previously created.</td></tr><tr><td><code>checkID</code></td><td>Unique identifier of the check that you add to the new objective.</td></tr></table>	<code>obj</code>	Handle to a code generation objective object previously created.	<code>checkID</code>	Unique identifier of the check that you add to the new objective.
<code>obj</code>	Handle to a code generation objective object previously created.				
<code>checkID</code>	Unique identifier of the check that you add to the new objective.				
Examples	Add the Disable signal logging check to the objective. <pre>addCheck(obj, 'com.mathworks.MA.CheckDisableSignalLogging');</pre>				
See Also	<code>Simulink.ModelAdvisor</code>				
How To	<ul style="list-style-type: none">• “Creating Custom Objectives”• “About IDs”				

addConceptualArg

Purpose Add conceptual argument to array of conceptual arguments for TFL table entry

Syntax `void addConceptualArg(hEntry, arg)`

Arguments

hEntry
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry or *hEntry* = RTW.Tf1COperationEntry.

arg
Argument of type Tf1Arg, such as returned by Tf1Arg* getTf1ArgFromString(*name*, *datatype*), to be added to the array of conceptual arguments for the TFL table entry.

Description The addConceptualArg function adds a specified conceptual argument to the array of conceptual arguments for a TFL table entry.

Example In the following example, the addConceptualArg function is used to add conceptual arguments for the output port and the two input ports for an addition operation.

```
hLib = RTW.Tf1Table;

% Create entry for addition of built-in uint8 data type
op_entry = RTW.Tf1COperationEntry;
op_entry.setTf1COperationEntryParameters( ...
    'Key', 'RTW_OP_ADD', ...
    'Priority', 90, ...
    'SaturationMode', 'RTW_SATURATE_ON_OVERFLOW', ...
    'RoundingMode', 'RTW_ROUND_UNSPECIFIED', ...
    'ImplementationName', 'u8_add_u8_u8', ...
    'ImplementationHeaderFile', 'u8_add_u8_u8.h', ...
    'ImplementationSourceFile', 'u8_add_u8_u8.c' );

arg = hLib.getTf1ArgFromString('y1','uint8');
arg.IOType = 'RTW_IO_OUTPUT';
op_entry.addConceptualArg( arg );
```

```
arg = hLib.getTf1ArgFromString('u1','uint8');
op_entry.addConceptualArg( arg );

arg = hLib.getTf1ArgFromString('u2','uint8');
op_entry.addConceptualArg( arg );

op_entry.copyConceptualArgsToImplementation();

hLib.addEntry( op_entry );
```

See Also

[getTf1ArgFromString](#)

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

cgv.CGV.addConfigSet

Purpose Add configuration set

Syntax

```
cgvObj.addConfigSet(configSet)
cgvObj.addConfigSet('configSetName')
cgvObj.addConfigSet('file', 'configSetFileName')
cgvObj.addConfigSet('file', 'configSetFileName', 'variable',
    'configSetName')
```

Description `cgvObj.addConfigSet(configSet)` is an optional method that adds the configuration set to the object. `cgvObj` is a handle to a `cgv.CGV` object. `cgvObj.configSet` is a variable that specifies the configuration set.

`cgvObj.addConfigSet('configSetName')` is an optional method that adds the configuration set to the object. `configSetName` is a string that specifies the name of the configuration set in the workspace.

`cgvObj.addConfigSet('file', 'configSetFileName')` is an optional method that adds the configuration set to the object. `configSetFileName` is a string that specifies the name of the file that contains only one configuration set.

`cgvObj.addConfigSet('file', 'configSetFileName', 'variable', 'configSetName')` is an optional method that adds the configuration set to the object. The file contains one or more configuration sets. Specify the name of the configuration set to use.

This method replaces all configuration parameter values in the model with the values from the configuration set that you add. The object applies the configuration set when you call the run method. You can add only one configuration set for each `cgv.CGV` object.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”
- “Managing Configuration Sets”

Purpose Add table entry to collection of table entries registered in TFL table

Syntax `void addEntry(hTable, entry)`

Arguments

hTable

Handle to a TFL table previously returned by *hTable* = RTW.TflTable.

entry

Handle to a function or operator entry that you have constructed after calling *hEntry* = RTW.TflCFunctionEntry or *hEntry* = RTW.TflCOperationEntry

Description

The addEntry function adds a function or operator entry that you have constructed to the collection of table entries registered in a TFL table.

Example

In the following example, the addEntry function is used to add an operator entry to a TFL table after the entry is constructed.

```

hLib = RTW.TflTable;

% Create an entry for addition of built-in uint8 data type
op_entry = RTW.TflCOperationEntry;
op_entry.setTflCOperationEntryParameters( ...
    'Key', 'RTW_OP_ADD', ...
    'Priority', 90, ...
    'SaturationMode', 'RTW_SATURATE_ON_OVERFLOW', ...
    'RoundingMode', 'RTW_ROUND_UNSPECIFIED', ...
    'ImplementationName', 'u8_add_u8_u8', ...
    'ImplementationHeaderFile', 'u8_add_u8_u8.h', ...
    'ImplementationSourceFile', 'u8_add_u8_u8.c' );

arg = hLib.getTflArgFromString('y1','uint8');
arg.IOType = 'RTW_IO_OUTPUT';
op_entry.addConceptualArg( arg );

arg = hLib.getTflArgFromString('u1','uint8');

```

addEntry

```
op_entry.addConceptualArg( arg );

arg = hLib.getTf1ArgFromString('u2','uint8');
op_entry.addConceptualArg( arg );

op_entry.copyConceptualArgsToImplementation();

addEntry(hLib, op_entry);
```

See Also

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

Purpose Add input data

Syntax `cgvObj.addInputData(InputIndex, 'DataFileName')`

Description `cgvObj.addInputData(InputIndex, 'DataFileName')` is a required method that adds an input data file to the object. `cgvObj` is a handle to a `cgv.CGV` object. `InputIndex` is a unique numeric identifier that you assign to input data that you want to use when the model executes. `DataFileName` is the name of the input file, with or without the `.mat` extension. If the input file is in the working folder, the `cgvObj` does not require the path.

The `cgvObj` uses the `InputIndex` to identify the input data associated with output data and output data files. `cgvObj` passes `InputIndex` to a callback function to identify the input data that the callback function uses.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”

RTW.AutosarInterface.addIOConf

Purpose Add AUTOSAR I/O configuration to model

Syntax

```
autosarInterfaceObj.addIOConf(SimulinkPort, DataAccessMode, autosarPort, InterfaceName, DataElement)  
autosarInterfaceObj.addIOConf(SimulinkErrorStatusPort, 'ErrorStatus', CorrespondingSimulinkReceiverPort)  
autosarInterfaceObj.addIOConf(SimulinkBasicSoftwarePort, 'BasicSoftwarePort', ServiceName, ServiceOperation, ServiceInterfacePath)
```

Description You can designate inports and outports to be data sender/receiver ports, error status receivers, or access points to AUTOSAR Basic Software using the method `addIOConf`:

```
autosarInterfaceObj.addIOConf(SimulinkPort, DataAccessMode, autosarPort, InterfaceName, DataElement)
```

```
autosarInterfaceObj.addIOConf(SimulinkErrorStatusPort, 'ErrorStatus', CorrespondingSimulinkReceiverPort)
```

```
autosarInterfaceObj.addIOConf(SimulinkBasicSoftwarePort, 'BasicSoftwarePort', ServiceName, ServiceOperation, ServiceInterfacePath)
```

Each call adds an AUTOSAR I/O configuration to *autosarInterfaceObj*, a model-specific `RTW.AutosarInterface` object.

Inputs	<i>SimulinkPort</i>	Inport/outport name (string)
	<i>DataAccessMode</i>	Data access mode of the port. You can designate inports and outports to be data sender/receiver ports by specifying <i>DataAccessMode</i> to be one of the following:

- 'ImplicitSend'
- 'ImplicitReceive'
- 'ExplicitSend'
- 'ExplicitReceive'

Use 'Implicit...' where data is buffered by the run-time environment (RTE), or 'Explicit...' where data is not buffered and hence not deterministic.

autosarPort

AUTOSAR port name
(string)

InterfaceName

Interface name (string)

DataElement

Data element name
(string)

SimulinkErrorStatusPort

The port you choose to receive error status.

'ErrorStatus'

The data access mode for ports chosen to be error status receivers.

CorrespondingSimulinkReceiverPort

The port that is listened to for error status. The data access mode for this port must be either 'ImplicitReceive' or 'ExplicitReceive'.

RTW.AutosarInterface.addIOConf

<i>SimulinkBasicSoftwarePort</i>	The port that you specify as an access point to AUTOSAR Basic Software.
'BasicSoftwarePort'	The data access mode for ports chosen to be access points to AUTOSAR Basic Software.
<i>ServiceName</i>	The service name you specify. Must be a valid AUTOSAR identifier.
<i>ServiceOperation</i>	The service operation you specify. Must be a valid AUTOSAR identifier.
<i>ServiceInterfacePath</i>	The service interface you specify. Must be a valid path of the form <i>AUTOSAR/Service/servicename</i> .

See Also

“Using the Configure AUTOSAR Interface Dialog Box” and “Configuring Ports for Basic Software and Error Status Receivers” in the Real-Time Workshop Embedded Coder documentation

Purpose Add parameters

Syntax `addParam(obj, paramName, value)`

Description `addParam(obj, paramName, value)` adds a parameter to the objective, and defines the value of the parameter that the Code Generation Advisor verifies in **Check model configuration settings against code generation objectives**.

Inputs

<i>obj</i>	Handle to a code generation objective object previously created.
<i>paramName</i>	Parameter that you add to the objective.
<i>value</i>	Value of the parameter.

Examples Add Inlineparameters to the objective, and specify the parameter value as on.

```
addParam(obj, 'InlineParams', 'on');
```

See Also `get_param`

How To

- “Creating Custom Objectives”
- “Parameter Command-Line Information Summary”

cgv.CGV.addPostLoadFiles

Purpose Add files required by model

Syntax `cgvObj.addPostLoadfiles({FileList})`

Description `cgvObj.addPostLoadfiles({FileList})` is an optional method that adds M- and MAT-files to the object. `cgvObj` is a handle to a `cgv.CGV` object. `cgvObj` executes and loads the files after opening the model and before running tests. `FileList` is a cell array of names of M- and MAT-files in the testing directory that the model requires to run.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”
- “Using Callback Functions”

Purpose	Control import of AUTOSAR components																
Description	You can use methods of the <code>arxml.importer</code> class to import AUTOSAR components in a controlled manner. For example, you can parse an AUTOSAR software component description file exported by DaVinci System Architect (from Vector Informatik GmbH), and import the component into a Simulink model for subsequent configuration, code generation, and export to XML.																
Construction	<code>arxml.importer</code> Construct <code>arxml.importer</code> object																
Methods	<table><tr><td><code>createCalibrationComponentObjects</code></td><td>Create Simulink calibration objects from AUTOSAR calibration component</td></tr><tr><td><code>createComponentAsModel</code></td><td>Create AUTOSAR atomic software component as Simulink model</td></tr><tr><td><code>createComponentAsSubsystem</code></td><td>Create AUTOSAR atomic software component as Simulink atomic subsystem</td></tr><tr><td><code>createOperationAsConfigurableSubsystem</code></td><td>Create configurable Simulink subsystem library for client-server operation</td></tr><tr><td><code>getCalibrationComponentNames</code></td><td>Get calibration component names</td></tr><tr><td><code>getComponentNames</code></td><td>Get atomic software component names</td></tr><tr><td><code>getDependencies</code></td><td>Get list of XML dependency files</td></tr><tr><td><code>getFile</code></td><td>Return XML file name for <code>arxml.importer</code> object</td></tr></table>	<code>createCalibrationComponentObjects</code>	Create Simulink calibration objects from AUTOSAR calibration component	<code>createComponentAsModel</code>	Create AUTOSAR atomic software component as Simulink model	<code>createComponentAsSubsystem</code>	Create AUTOSAR atomic software component as Simulink atomic subsystem	<code>createOperationAsConfigurableSubsystem</code>	Create configurable Simulink subsystem library for client-server operation	<code>getCalibrationComponentNames</code>	Get calibration component names	<code>getComponentNames</code>	Get atomic software component names	<code>getDependencies</code>	Get list of XML dependency files	<code>getFile</code>	Return XML file name for <code>arxml.importer</code> object
<code>createCalibrationComponentObjects</code>	Create Simulink calibration objects from AUTOSAR calibration component																
<code>createComponentAsModel</code>	Create AUTOSAR atomic software component as Simulink model																
<code>createComponentAsSubsystem</code>	Create AUTOSAR atomic software component as Simulink atomic subsystem																
<code>createOperationAsConfigurableSubsystem</code>	Create configurable Simulink subsystem library for client-server operation																
<code>getCalibrationComponentNames</code>	Get calibration component names																
<code>getComponentNames</code>	Get atomic software component names																
<code>getDependencies</code>	Get list of XML dependency files																
<code>getFile</code>	Return XML file name for <code>arxml.importer</code> object																

arxml.importer class

setDependencies

Set XML file dependencies

setFile

Set XML file name for
arxml.importer object

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB® Programming Fundamentals documentation.

See Also

Importing an AUTOSAR Software Component in the Real-Time Workshop Embedded Coder documentation

Purpose Construct arxml.importer object

Syntax `importer_obj = importer(filename)`

Description `importer_obj = importer(filename)` constructs an arxml.importer object and parses the atomic software component described in the XML file specified by *filename*.

Note Only the atomic software components described in this XML file can be imported.

Inputs

<i>filename</i>	XML file containing description of atomic software component.
-----------------	---

Outputs

<i>importer_obj</i>	Handle to newly created arxml.importer object.
---------------------	--

See Also “Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.attachToModel

Purpose	Attach RTW.AutosarInterface object to model		
Syntax	<code>autosarInterfaceObj.attachToModel(modelName)</code>		
Description	<code>autosarInterfaceObj.attachToModel(modelName)</code> attaches <code>autosarInterfaceObj</code> , an RTW.AutosarInterface object, to a loaded Simulink model with an ERT-based target.		
Inputs	<table><tr><td><code>modelName</code></td><td>Name of a loaded Simulink model to which the object is going to be attached (string).</td></tr></table>	<code>modelName</code>	Name of a loaded Simulink model to which the object is going to be attached (string).
<code>modelName</code>	Name of a loaded Simulink model to which the object is going to be attached (string).		
See Also	“Modifying and Validating an Existing AUTOSAR Interface” in the Real-Time Workshop Embedded Coder documentation		

Purpose Attach model-specific C++ encapsulation interface to loaded ERT-based Simulink model

Syntax `attachToModel(obj, modelName)`

Description `attachToModel(obj, modelName)` attaches a model-specific C++ encapsulation interface to a loaded ERT-based Simulink model.

Inputs

<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> .
<i>modelName</i>	String specifying the name of a loaded ERT-based Simulink model to which the object is going to be attached.

Alternatives The **Configure C++ Encapsulation Interface** button on the **Interface** pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. Once you validate and apply your changes, you can generate code based on your C++ encapsulation interface modifications. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.

See Also

- “Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation
- “Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation
- “Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.attachToModel

Purpose	Attach model-specific C function prototype to loaded ERT-based Simulink model				
Syntax	<code>attachToModel(obj, modelName)</code>				
Description	<code>attachToModel(obj, modelName)</code> attaches a model-specific C function prototype to a loaded ERT-based Simulink model.				
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code>.</td></tr><tr><td><i>modelName</i></td><td>String specifying the name of a loaded ERT-based Simulink model to which the object is going to be attached.</td></tr></table>	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> .	<i>modelName</i>	String specifying the name of a loaded ERT-based Simulink model to which the object is going to be attached.
<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> .				
<i>modelName</i>	String specifying the name of a loaded ERT-based Simulink model to which the object is going to be attached.				
Alternatives	Click the Configure Model Functions button on the Configuration Parameters > Real-Time Workshop > Interface pane for flexible control over the model function prototypes that are generated for your model. Once you validate and apply your changes, you can generate code based on your function prototype modifications. See “Configuring Model Function Prototypes” in the Real-Time Workshop Embedded Coder documentation.				
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code				

Purpose	Verify numerical equivalence of results
Description	<p>Verify numerical equivalence of multiple execution results.</p> <p>Creates a handle to a <code>cgv.CGV</code> object that supports executing the model in different environments such as, simulation, Software-In-the-Loop (SIL), or Processor-In-the-Loop (PIL) and stores numerical results. Use the <code>cgv.CGV</code> class to create a script to verify that the model and the Real-Time Workshop Embedded Coder generated code produce numerically equivalent results.</p>
Construction	<p><code>cgvObj = cgv.CGV('model_name')</code> creates a handle to a code generation verification object using the default properties. <i>model_name</i> is the name of the model that you are verifying. Returns a handle to a <code>cgv.CGV</code> object, <i>cgvObj</i>.</p> <p><code>cgvObj = cgv.CGV('model_name', 'PropertyName', 'PropertyValue')</code> constructs the object using options, specified as property name and value pairs. Property names and values are not case sensitive.</p> <p><code>cgv.Config</code> and <code>cgv.CGV</code> use many of the same properties. When the <code>cgv.CGV</code> object executes, it uses <code>cgv.Config</code> to verify that the model is configured correctly for the mode of execution that you specify.</p>
Properties	<p><code>ComponentType</code></p> <p>Define the SIL or PIL approach</p> <p>If mode of execution is simulation (<code>connectivity</code> is <code>sim</code>), choosing either value for <code>ComponentType</code> has no effect on simulation results. However, <code>cgv.CGV</code> recommends configuration parameter values based on the value of <code>ComponentType</code>.</p>

cgv.CGV class

Value	Description
'topmodel' (default)	Top-model SIL or PIL simulation and standalone code interface mode.
'modelblock'	Model block SIL or PIL simulation and model reference Real-Time Workshop target code interface mode.

connectivity

Specify mode of execution

Value	Description
'sim' (default)	Mode of execution is simulation. Recommends changes to a proper subset of the configuration parameters that all PIL targets require.
'sil'	Mode of execution is SIL. Recommends changes to the configuration parameters that SIL targets require.
'tasking'	Mode of execution is PIL using the Embedded IDE Link™ software and Altium® TASKING tools configuration. Requires that you specify 'processor'. Recommends changes to the configuration parameters that PIL targets using the Embedded IDE

Value	Description
	Link software and Altium TASKING tools require.
'custom'	Mode of execution is PIL with custom connectivity that you provide using the PIL Connectivity API. Recommends changes to the configuration parameters that PIL targets with custom connectivity require.

LogMode

Specify how to save signal data

Value	Description
'SignalLogging' (default)	Save signal data to the MATLAB workspace variable during execution. To save the data, in the Configuration Parameters dialog box, the object uses the Data Import/Export > Signal logging parameter. In the Signal logging name field, define the variable name.
'SaveOutput'	Save signal data to the MATLAB workspace variable during execution. To save the data, in the Configuration Parameters Dialog box,

Value	Description
	<p>the object uses the Data Import/Export > Output parameter. In the Output name field, define the variable name.</p> <p>When you use the Output parameter to save signal data, Simulink does not save bus outputs.</p>

processor

Defines the processor type.

Use processor only when the value of connectivity is tasking.

Value	Description
'ARM'	ARM® processor
'TriCore'	Infineon® TriCore® processor
'C166'	Infineon® C166® processor
'8051'	Intel® 8051 processor
'M16C'	Renesas® M16C processor
'DSP563xx'	Freescale™ DSP566xx processor

SaveModel

Specify whether to save the model

Value	Description
'off' (default)	Do not save the model.
'on'	Save the model in the working folder.

Methods

addCallback	Add callback function
addConfigSet	Add configuration set
addInputData	Add input data
addPostLoadFiles	Add files required by model
getOutputData	Get output data
run	Execute CGV object
setOutputDir	Specify folder
setOutputFile	Specify output data file name

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples

Configure and execute the `rtwdemo_iec61508` model in simulation and SIL modes. Compare the outputs of the two modes of execution. Create a different version of the model to illustrate comparing mismatched outputs.

Note To compare the results, you must create the `compareResults.m` and `setupAndPlot.m` files as described in “Comparing Outputs”.

```
%% Configure the Model
% Save a local copy of rtwdemo_iec61508.
srcModel = 'rtwdemo_iec61508';
cgvModel = 'rtwdemo_iec61508_cgv';
load_system(srcModel);
save_system(srcModel,[pwd filesep cgvModel]);
% Close the rtwdemo_iec61508 model.
bdclose(srcModel);
clear srcModel;
```

cgv.CGV class

```
% Create a cgV.Config object and configure the model for top-model SIL.
cgvCfg = cgV.Config( cgvModel, 'LogMode', 'SaveOutput', ...
    'connectivity', 'sil', 'SaveModel', 'on');
cgvCfg.configModel();
% Display the results of what the cgV.Config object changed.
cgvCfg.displayReport();

load_system(cgvModel);
% Configure the format of the outport data.
set_param( cgvModel, 'SaveFormat', 'StructureWithTime');

% Configure the inports to receive data that is configured in the MAT-file.
set_param( cgvModel, 'LoadExternalInput', 'on');
set_param( cgvModel, 'ExternalInput', 'uIn');

% Do not display the code generation report.
set_param( cgvModel, 'GenerateReport', 'off');

% Save these changes.
save_system(cgvModel);

% Create the input data
InputData = 'rtwdemo_cgvScript_data';
x.time = (0:.1:4)';
values = sin( 2.5* x.time) + 1;
x.signals(1).values = uint16(values * 32767/2);
x.signals(2).values = uint16(values * 32767/4);
x.signals(3).values = int16(values * 32767/2);
x.signals(4).values = int16(values * 11000/2);
uIn = x;
save( InputData, 'uIn');

%% Execute rtwdemo_iec61508_cgv in SIM mode
cgvObjSim = cgV.CGV(cgvModel, 'LogMode', 'SaveOutput');
cgvObjSim.addInputData(1,InputData);
result = cgvObjSim.run();
% Verify that the execution completed without error.
```



```
if result == true
    outputDataSim = cgvObjSim.getOutputData(1);
end

%% Execute rtdemo_iec61508_cgv in SIL mode
cgvObjSil = cgv.CGV(cgvModel, 'LogMode', 'SaveOutput', 'connectivity', 'sil');
cgvObjSil.addInputData(1, InputData);
result = cgvObjSil.run();
% Verify that the execution completed without error.
if result == true
    outputDataSil = cgvObjSil.getOutputData(1);
end

%% Compare and display results appropriate to the model output format
% Note: You must create compareResults.m and setupAndPlot.m as directed in
% the user's guide.
% The compareResults function displays a message in the MATLAB Command Window
% for each matching output, and creates a plot of each mismatched output.

% Since there are no differences in the output data,
% compareResults only displays a message in the MATLAB Command Window.
tolerance = 1e-10;
compareResults( tolerance, outputDataSim, outputDataSil);

% Manually plot the data. The results appear as a single line, with the
% second plot overlaying the first.
setupAndPlot('outputDataSim', 'outputDataSil', ...
    outputDataSim.signals(1).blockName, outputDataSim.time, ...
    outputDataSim.signals(1).values, outputDataSil.signals(1).values, 50, 50);

%% Create a version of rtdemo_iec61508_cgv
% In the Add2 block, select 'Saturate on integer overflow'.
cgvModelSaturate = 'rtdemo_iec61508_cgv_saturate';
load_system(cgvModel);
set_param([cgvModel '/Add2'], 'SaturateOnIntegerOverflow', 'on');
save_system(cgvModel, cgvModelSaturate);
```

cgv.CGV class

```
%% Execute rtwdemo_iec61508_cgv_saturate in SIL mode
cgvObjSilErr = cgv.CGV('rtwdemo_iec61508_cgv_saturate', ...
    'LogMode', 'SaveOutput', 'connectivity', 'sil');
cgvObjSilErr.addInputData(1,InputData);
result = cgvObjSilErr.run();
% Verify that the execution completed without error.
if result == true
    outputDataSilErr = cgvObjSilErr.getOutputData(1);
end

%% Compare and display results appropriate to the model output format
% Note: You must create compareResults.m and setupAndPlot.m as directed in
% the user's guide.
% The compareResults function displays a message in the MATLAB Command Window
% for each matching output, and creates a plot of each mismatched output.

% compareResults creates a plot of the mismatched output.
compareResults( tolerance, outputDataSim, outputDataSilErr);
```

See Also

cgv.Config

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”
- “Verifying Compiled Object Code with Processor-in-the-Loop Simulation”

Purpose

Check and modify model configuration parameter values

Description

Creates a handle to a `cgv.Config` object that supports checking and optionally modifying models for compatibility with various modes of execution, such as simulation, Software-In-the-Loop (SIL), or Processor-In-the-Loop (PIL).

To execute the model successfully in the mode that you specify, you might need to make additional modifications to the configuration parameter values or the model.

By default, `cgv.Config` modifies configuration parameter values to the value that it recommends, but does not save the model. Alternatively, you can specify that `cgv.Config` use one of the following approaches:

- Modify configuration parameter values to the values that `cgv.Config` recommends, and save the model. Specify this approach using the `SaveModel` property.
- List the values that `cgv.Config` recommends for the configuration parameters, but do not modify the configuration parameters or the model. Specify this approach using the `ReportOnly` property.

Do not use referenced configuration sets in models that you are modifying using `cgv.Config`. If the model uses a referenced configuration set, update the model with a copy of the configuration set. Use the `Simulink.ConfigSetRef.getRefConfigSet` method. For more information, see `Simulink.ConfigSetRef` in the Simulink documentation.

If you use `cgv.Config` on a model that executes a callback function, the callback function might modify configuration parameter values each time the model loads. The callback function might revert changes that `cgv.Config` made. When this change occurs, the model might no longer be set up correctly for SIL or PIL. For more information, see “Using Callback Functions”.

Construction

`cfgObj = cgv.Config('model_name')` creates a handle to a `cgv.Config` object, `cfgObj`, using default values for properties.

cgv.Config class

model_name is the name of the model that you are checking and optionally configuring.

cfgObj =
`cgv.Config('model_name', 'PropertyName', 'PropertyValue')`
constructs the object using options, specified as property name and value pairs. Property names and values are not case sensitive.

`cgv.Config` and `cgv.CGV` use many of the same properties. When the `cgv.CGV` object executes, it uses `cgv.Config` to verify that the model is configured correctly for the mode of execution that you specify.

Properties

`ComponentType`

Define the SIL or PIL approach

If mode of execution is simulation (`connectivity` is `sim`), choosing either value for `ComponentType` has no effect on simulation results. However, `cgv.Config` recommends configuration parameter values based on the value of `ComponentType`.

Value	Description
'topmodel' (default)	Top-model SIL or PIL simulation and standalone code interface mode.
'modelblock'	Model block SIL or PIL simulation and model reference Real-Time Workshop target code interface mode.

`connectivity`

Specify mode of execution

Value	Description
'sim' (default)	Mode of execution is simulation. Recommends changes to a proper subset of the configuration parameters that all PIL targets require.
'sil'	Mode of execution is SIL. Recommends changes to the configuration parameters that SIL targets require.
'tasking'	Mode of execution is PIL using the Embedded IDE Link software and Altium TASKING tools configuration. Requires that you specify 'processor'. Recommends changes to the configuration parameters that PIL targets using the Embedded IDE Link software and Altium TASKING tools require.
'custom'	Mode of execution is PIL with custom connectivity that you provide using the PIL Connectivity API. Recommends changes to the configuration parameters that PIL targets with custom connectivity require.

LogMode

Specify how to save signal data

cgv.Config class

Value	Description
'SignalLogging' (default)	<p>Save signal data to the MATLAB workspace variable during execution.</p> <p>To save the data, in the Configuration Parameters dialog box, the object uses the Data Import/Export > Signal logging parameter . In the Signal logging name field, define the variable name.</p>
'SaveOutput'	<p>Save signal data to the MATLAB workspace variable during execution.</p> <p>To save the data, in the Configuration Parameters dialog box, the object uses the Data Import/Export > Output parameter . In the Output name field, define the variable name.</p> <p>When you use the Output parameter to save signal data, Simulink does not save bus outputs.</p>

processor

Defines the processor type

Use processor only when the value of connectivity is tasking.

Value	Description
'ARM'	ARM processor
'TriCore'	Infineon TriCore processor
'C166'	Infineon C166 processor
'8051'	Intel 8051 processor
'M16C'	Renesas M16C processor
'DSP563xx'	Freescale DSP563xx processor

ReportOnly

Specify whether to create a report

Specify whether to create a report of the values that `cgv.Config` recommends for the configuration parameters, without modifying the configuration parameters or the model. If you set `ReportOnly` to 'on', `SaveModel` must be 'off'.

Value	Description
'off' (default)	Do not create a report.
'on'	Create a report.

SaveModel

Specify whether to save the model

Specify whether to modify configuration parameter values to the values that `cgv.Config` recommends, and save the model. If you set `SaveModel` to 'on', `ReportOnly` must be 'off'.

Value	Description
'off' (default)	Do not save the model.
'on'	Save the model in the working folder.

cgv.Config class

Methods

<code>configModel</code>	Determine and change configuration parameter values
<code>displayReport</code>	Display results of comparing configuration parameter values
<code>getReportData</code>	Return results of comparing configuration parameter values

Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples

Configure the `rtwdemo_iec61508` model for top-model SIL. Then view the changes at the MATLAB Command Window:

```
% Create a cgv.Config object and configure the model for top-model SIL.
cgvCfg = cgv.Config('rtwdemo_iec61508', 'LogMode', 'SaveOutput', ...
    'connectivity', 'sil');
cgvCfg.configModel();
% Display the results of what the cgv.Config object changed.
cgvCfg.displayReport();
% Close the rtwdemo_iec61508 model.
bdclose('rtwdemo_iec61508');
```

See Also

`cgv.CGV`

Tutorials

- “How To Verify a SIL or PIL Configuration”

How To

- “Configuring a PIL Simulation”
- “Choosing a PIL Simulation Approach”
- “Managing Configuration Sets”

Purpose

Determine and change configuration parameter values

Syntax

`cfgObj.configModel()`

Description

`cfgObj.configModel()` determines the recommended values for the configuration parameters in the model. `cfgObj` is a handle to a `cgv.Config` object. The `ReportOnly` property of the object determines whether `configModel` reports or changes the configuration parameter values.

Tutorials

- “How To Verify a SIL or PIL Configuration”

How To

- “Configuring a PIL Simulation”
- “Managing Configuration Sets”

copyConceptualArgsToImplementation

- Purpose** Copy conceptual argument specifications to matching implementation arguments for TFL table entry
- Syntax** `void copyConceptualArgsToImplementation(hEntry)`
- Arguments** *hEntry*
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry or *hEntry* = RTW.Tf1COperationEntry.
- Description** The `copyConceptualArgsToImplementation` function provides a quick way to copy conceptual argument specifications to matching implementation arguments. This function can be used when the conceptual arguments and the implementation arguments are the same for a TFL table entry.
- Example** In the following example, the `copyConceptualArgsToImplementation` function is used to copy conceptual argument specifications to matching implementation arguments for an addition operation.

```
hLib = RTW.Tf1Table;

% Create an entry for addition of built-in uint8 data type
op_entry = RTW.Tf1COperationEntry;
op_entry.setTf1COperationEntryParameters( ...
    'Key',                'RTW_OP_ADD', ...
    'Priority',           90, ...
    'SaturationMode',    'RTW_SATURATE_ON_OVERFLOW', ...
    'RoundingMode',      'RTW_ROUND_UNSPECIFIED', ...
    'ImplementationName', 'u8_add_u8_u8', ...
    'ImplementationHeaderFile', 'u8_add_u8_u8.h', ...
    'ImplementationSourceFile', 'u8_add_u8_u8.c' );

arg = hLib.getTf1ArgFromString('y1','uint8');
arg.IOType = 'RTW_IO_OUTPUT';
op_entry.addConceptualArg( arg );
```

```
arg = hLib.getTf1ArgFromString('u1','uint8');
op_entry.addConceptualArg( arg );

arg = hLib.getTf1ArgFromString('u2','uint8');
op_entry.addConceptualArg( arg );

op_entry.copyConceptualArgsToImplementation();

hLib.addEntry( op_entry );
```

See Also

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

createAndAddConceptualArg

Purpose Create conceptual argument from specified properties and add to conceptual arguments for TFL table entry

Syntax `void createAndAddConceptualArg(hEntry, argType, varargin)`

Arguments

hEntry
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry or *hEntry* = RTW.Tf1COperationEntry.

argType
String specifying the argument type to create:
'RTW.Tf1ArgNumeric' for numeric or 'RTW.Tf1ArgMatrix' for matrix.

varargin
Parameter/value pairs for the conceptual argument. See varargin Parameters.

varargin Parameters

The following argument properties can be specified to the createAndAddConceptualArg function using parameter/value argument pairs. For example,

```
createAndAddConceptualArg(..., 'DataTypeMode', 'double', ...);
```

Name
String specifying the argument name, for example, 'y1' or 'u1'.

I/OType
String specifying the I/O type of the argument: 'RTW_IO_INPUT' for input or 'RTW_IO_OUTPUT' for output. The default is 'RTW_IO_INPUT'.

IsSigned
Boolean value that, when set to true, indicates that the argument is signed. The default is true.

WordLength
Integer specifying the word length, in bits, of the argument. The default is 16.

CheckSlope

Boolean flag that, when set to `true` for a fixed-point argument, causes TFL replacement request processing to check that the slope value of the argument exactly matches the call-site slope value. The default is `true`.

Specify `true` if you are matching a specific [slope bias] scaling combination or a specific binary-point-only scaling combination on fixed-point operator inputs and output. Specify `false` if you are matching relative scaling or relative slope and bias values across fixed-point operator inputs and output.

CheckBias

Boolean flag that, when set to `true` for a fixed-point argument, causes TFL replacement request processing to check that the bias value of the argument exactly matches the call-site bias value. The default is `true`.

Specify `true` if you are matching a specific [slope bias] scaling combination or a specific binary-point-only scaling combination on fixed-point operator inputs and output. Specify `false` if you are matching relative scaling or relative slope and bias values across fixed-point operator inputs and output.

DataTypeMode

String specifying the data type mode of the argument: `'boolean'`, `'double'`, `'single'`, `'Fixed-point: binary point scaling'`, or `'Fixed-point: slope and bias scaling'`. The default is `'Fixed-point: binary point scaling'`.

Note You can specify either `DataType` (with `Scaling`) or `DataTypeMode`, but do not specify both.

DataType

String specifying the data type of the argument: `'boolean'`, `'double'`, `'single'`, or `'Fixed'`. The default is `'Fixed'`.

createAndAddConceptualArg

Scaling

String specifying the data type scaling of the argument: 'BinaryPoint' for binary-point scaling or 'SlopeBias' for slope and bias scaling. The default is 'BinaryPoint'.

Slope

Floating-point value specifying the slope of the argument, for example, 15.0. The default is 1.

If you are matching a specific [slope bias] scaling combination on fixed-point operator inputs and output, specify either this parameter or a combination of the SlopeAdjustmentFactor and FixedExponent parameters

SlopeAdjustmentFactor

Floating-point value specifying the slope adjustment factor (F) part of the slope, $F2^E$, of the argument. The default is 1.0.

If you are matching a specific [slope bias] scaling combination on fixed-point operator inputs and output, specify either the Slope parameter or a combination of this parameter and the FixedExponent parameter.

FixedExponent

Integer value specifying the fixed exponent (E) part of the slope, $F2^E$, of the argument. The default is -15.

If you are matching a specific [slope bias] scaling combination on fixed-point operator inputs and output, specify either the Slope parameter or a combination of this parameter and the SlopeAdjustmentFactor parameter.

Bias

Floating-point value specifying the bias of the argument, for example, 2.0. The default is 0.0.

Specify this parameter if you are matching a specific [slope bias] scaling combination on fixed-point operator inputs and output.

FractionLength

Integer value specifying the fraction length for the argument, for example, 3. The default is 15.

Specify this parameter if you are matching a specific binary-point-only scaling combination on fixed-point operator inputs and output.

BaseType

String specifying the base data type for which a matrix argument is valid, for example, 'double'.

DimRange

Dimensions for which a matrix argument is valid, for example, [2 2]. You can also specify a range of dimensions specified in the format [Dim1Min Dim2Min ... DimNMin; Dim1Max Dim2Max ... DimNMax]. For example, [2 2; inf inf] means any two-dimensional matrix of size 2x2 or larger.

Description

The `createAndAddConceptualArg` function creates a conceptual argument from specified properties and adds the argument to the conceptual arguments for a TFL table entry.

Examples

In the following example, the `createAndAddConceptualArg` function is used to specify conceptual output and input arguments for a TFL operator entry.

```
op_entry = RTW.Tf1COperationEntry;  
.   
.   
.   
createAndAddConceptualArg(op_entry, 'RTW.Tf1ArgNumeric', ...  
                           'Name',      'y1', ...  
                           'IOType',    'RTW_IO_OUTPUT', ...  
                           'IsSigned',  true, ...  
                           'WordLength', 32, ...  
                           'FractionLength', 0);
```

createAndAddConceptualArg

```
createAndAddConceptualArg(op_entry, 'RTW.Tf1ArgNumeric',...
                           'Name',      'u1', ...
                           'IOType',    'RTW_IO_INPUT',...
                           'IsSigned',  true,...
                           'WordLength', 32, ...
                           'FractionLength', 0 );
```

```
createAndAddConceptualArg(op_entry, 'RTW.Tf1ArgNumeric',...
                           'Name',      'u2', ...
                           'IOType',    'RTW_IO_INPUT',...
                           'IsSigned',  true,...
                           'WordLength', 32, ...
                           'FractionLength', 0 );
```

The following examples show some common type specifications using `createAndAddConceptualArg`.

```
% uint8:
createAndAddConceptualArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                           'Name',      'u1', ...
                           'IOType',    'RTW_IO_INPUT', ...
                           'IsSigned',  false, ...
                           'WordLength', 8, ...
                           'FractionLength', 0 );
```

```
% single:
createAndAddConceptualArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                           'Name',      'u1', ...
                           'IOType',    'RTW_IO_INPUT', ...
                           'DataTypeMode', 'single' );
```

```
% double:
createAndAddConceptualArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                           'Name',      'y1', ...
                           'IOType',    'RTW_IO_OUTPUT', ...
                           'DataTypeMode', 'double' );
```



```
% boolean:
createAndAddConceptualArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                           'Name',      'u1', ...
                           'IOType',    'RTW_IO_INPUT', ...
                           'DataTypeMode', 'boolean' );

% Fixed-point using binary-point-only scaling:
createAndAddConceptualArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                           'Name',      'y1', ...
                           'IOType',    'RTW_IO_OUTPUT', ...
                           'CheckSlope', true, ...
                           'CheckBias', true, ...
                           'DataTypeMode', 'Fixed-point: binary point scaling', ...
                           'IsSigned',   true, ...
                           'WordLength', 32, ...
                           'FractionLength', 28);

% Fixed-point using [slope bias] scaling:
createAndAddConceptualArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                           'Name',      'y1', ...
                           'IOType',    'RTW_IO_OUTPUT', ...
                           'CheckSlope', true, ...
                           'CheckBias', true, ...
                           'DataTypeMode', 'Fixed-point: slope and bias scaling', ...
                           'IsSigned',   true, ...
                           'WordLength', 16, ...
                           'Slope',      15, ...
                           'Bias',       2);
```

For examples of fixed-point arguments that use relative scaling or relative slope/bias values, see “Example: Creating Fixed-Point Operator Entries for Relative Scaling (Multiplication and Division)” and “Example: Creating Fixed-Point Operator Entries for Equal Slope and Zero Net Bias (Addition and Subtraction)” in the Real-Time Workshop Embedded Coder documentation.

createAndAddConceptualArg

See Also

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

createAndAddImplementationArg

Purpose	Create implementation argument from specified properties and add to implementation arguments for TFL table entry
Syntax	<pre>void createAndAddImplementationArg(<i>hEntry</i>, <i>argType</i>, <i>varargin</i>)</pre>
Arguments	<p><i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = RTW.Tf1CFunctionEntry or <i>hEntry</i> = RTW.Tf1COperationEntry.</p> <p><i>argType</i> String specifying the argument type to create: 'RTW.Tf1ArgNumeric' for numeric.</p> <p><i>varargin</i> Parameter/value pairs for the implementation argument. See <i>varargin</i> Parameters.</p>
varargin Parameters	<p>The following argument properties can be specified to the <code>createAndAddImplementationArg</code> function using parameter/value argument pairs. For example,</p> <pre>createAndAddImplementationArg(..., 'DataTypeMode', 'double', ...);</pre> <p>Name String specifying the argument name, for example, 'u1'.</p> <p>IOType String specifying the I/O type of the argument: 'RTW_IO_INPUT' for input.</p> <p>IsSigned Boolean value that, when set to true, indicates that the argument is signed. The default is true.</p> <p>WordLength Integer specifying the word length, in bits, of the argument. The default is 16.</p>

createAndAddImplementationArg

DataTypeMode

String specifying the data type mode of the argument: 'boolean', 'double', 'single', 'Fixed-point: binary point scaling', or 'Fixed-point: slope and bias scaling'. The default is 'Fixed-point: binary point scaling'.

Note You can specify either `DataType` (with `Scaling`) or `DataTypeMode`, but do not specify both.

DataType

String specifying the data type of the argument: 'boolean', 'double', 'single', or 'Fixed'. The default is 'Fixed'.

Scaling

String specifying the data type scaling of the argument: 'BinaryPoint' for binary-point scaling or 'SlopeBias' for slope and bias scaling. The default is 'BinaryPoint'.

Slope

Floating-point value specifying the slope of the argument, for example, 15.0. The default is 1.

You can optionally specify either this parameter or a combination of the `SlopeAdjustmentFactor` and `FixedExponent` parameters, but do not specify both.

SlopeAdjustmentFactor

Floating-point value specifying the slope adjustment factor (F) part of the slope, $F2^E$, of the argument. The default is 1.0.

You can optionally specify either the `Slope` parameter or a combination of this parameter and the `FixedExponent` parameter, but do not specify both.

FixedExponent

Integer value specifying the fixed exponent (E) part of the slope, $F2^E$, of the argument. The default is -15.

You can optionally specify either the `Slope` parameter or a combination of this parameter and the `SlopeAdjustmentFactor` parameter, but do not specify both.

Bias

Floating-point value specifying the bias of the argument, for example, 2.0. The default is 0.0.

FractionLength

Integer value specifying the fraction length of the argument, for example, 3. The default is 15.

Description

The `createAndAddImplementationArg` function creates an implementation argument from specified properties and adds the argument to the implementation arguments for a TFL table entry.

Note Implementation arguments must describe fundamental numeric data types, such as `double`, `single`, `int32`, `int16`, `int8`, `uint32`, `uint16`, `uint8`, or `boolean` (not fixed point data types).

Example

In the following example, the `createAndAddImplementationArg` function is used along with the `createAndSetCImplementationReturn` function to specify the output and input arguments for an operator implementation.

```
op_entry = RTW.Tf1COperationEntry;
.
.
.
createAndSetCImplementationReturn(op_entry, 'RTW.Tf1ArgNumeric', ...
                                   'Name',      'y1', ...
                                   'IOType',    'RTW_IO_OUTPUT', ...
                                   'IsSigned',  true, ...
                                   'WordLength', 32, ...
                                   'FractionLength', 0);
```

createAndAddImplementationArg

```
createAndAddImplementationArg(op_entry, 'RTW.Tf1ArgNumeric', ...
                              'Name',      'u1', ...
                              'IOType',    'RTW_IO_INPUT',...
                              'IsSigned',   true,...
                              'WordLength', 32, ...
                              'FractionLength', 0 );
```

```
createAndAddImplementationArg(op_entry, 'RTW.Tf1ArgNumeric', ...
                              'Name',      'u2', ...
                              'IOType',    'RTW_IO_INPUT',...
                              'IsSigned',   true,...
                              'WordLength', 32, ...
                              'FractionLength', 0 );
```

The following examples show some common type specifications using `createAndAddImplementationArg`.

```
% uint8:
createAndAddImplementationArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                              'Name',      'u1', ...
                              'IOType',    'RTW_IO_INPUT', ...
                              'IsSigned',   false, ...
                              'WordLength', 8, ...
                              'FractionLength', 0 );
```

```
% single:
createAndAddImplementationArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                              'Name',      'u1', ...
                              'IOType',    'RTW_IO_INPUT', ...
                              'DataTypeMode', 'single' );
```

```
% double:
createAndAddImplementationArg(hEntry, 'RTW.Tf1ArgNumeric', ...
                              'Name',      'u1', ...
                              'IOType',    'RTW_IO_INPUT', ...
                              'DataTypeMode', 'double' );
```

```
% boolean:  
createAndAddImplementationArg(hEntry, 'RTW.Tf1ArgNumeric', ...  
                                'Name',      'u1', ...  
                                'IOType',    'RTW_IO_INPUT', ...  
                                'DataTypeMode', 'boolean' );
```

See Also

`createAndSetCImplementationReturn`

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

createAndSetCImplementationReturn

Purpose Create implementation return argument from specified properties and add to implementation for TFL table entry

Syntax `void createAndSetCImplementationReturn(hEntry,
argType, varargin)`

Arguments

hEntry
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry or *hEntry* = RTW.Tf1COperationEntry.

argType
String specifying the argument type to create:
'RTW.Tf1ArgNumeric' for numeric.

varargin
Parameter/value pairs for the implementation return argument.
See varargin Parameters.

varargin Parameters The following argument properties can be specified to the createAndSetCImplementationReturn function using parameter/value argument pairs. For example,

```
createAndSetCImplementationReturn(..., 'DataTypeMode', 'double', ...);
```

Name String specifying the argument name, for example, 'y1'.

IOType String specifying the I/O type of the argument: 'RTW_IO_OUTPUT' for output.

IsSigned Boolean value that, when set to true, indicates that the argument is signed. The default is true.

WordLength Integer specifying the word length, in bits, of the argument. The default is 16.

DataTypeMode

String specifying the data type mode of the argument: 'boolean', 'double', 'single', 'Fixed-point: binary point scaling', or 'Fixed-point: slope and bias scaling'. The default is 'Fixed-point: binary point scaling'.

Note You can specify either `DataType` (with `Scaling`) or `DataTypeMode`, but do not specify both.

DataType

String specifying the data type of the argument: 'boolean', 'double', 'single', or 'Fixed'. The default is 'Fixed'.

Scaling

String specifying the data type scaling of the argument: 'BinaryPoint' for binary-point scaling or 'SlopeBias' for slope and bias scaling. The default is 'BinaryPoint'.

Slope

Floating-point value specifying the slope for a fixed-point argument, for example, 15.0. The default is 1.

You can optionally specify either this parameter or a combination of the `SlopeAdjustmentFactor` and `FixedExponent` parameters, but do not specify both.

SlopeAdjustmentFactor

Floating-point value specifying the slope adjustment factor (F) part of the slope, $F2^E$, of the argument. The default is 1.0.

You can optionally specify either the `Slope` parameter or a combination of this parameter and the `FixedExponent` parameter, but do not specify both.

FixedExponent

Integer value specifying the fixed exponent (E) part of the slope, $F2^E$, of the argument. The default is -15.

createAndSetCImplementationReturn

You can optionally specify either the `Slope` parameter or a combination of this parameter and the `SlopeAdjustmentFactor` parameter, but do not specify both.

Bias

Floating-point value specifying the bias of the argument, for example, 2.0. The default is 0.0.

FractionLength

Integer value specifying the fraction length of the argument, for example, 3. The default is 15.

Description

The `createAndSetCImplementationReturn` function creates an implementation return argument from specified properties and adds the argument to the implementation for a TFL table.

Note Implementation return arguments must describe fundamental numeric data types, such as `double`, `single`, `int32`, `int16`, `int8`, `uint32`, `uint16`, `uint8`, or `boolean` (not fixed point data types).

Example

In the following example, the `createAndSetCImplementationReturn` function is used along with the `createAndAddImplementationArg` function to specify the output and input arguments for an operator implementation.

```
op_entry = RTW.Tf1COperationEntry;
.
.
.
createAndSetCImplementationReturn(op_entry, 'RTW.Tf1ArgNumeric', ...
                                   'Name',      'y1', ...
                                   'IOType',    'RTW_IO_OUTPUT', ...
                                   'IsSigned',  true, ...
                                   'WordLength', 32, ...
                                   'FractionLength', 0);
```

createAndSetCImplementationReturn

```
createAndAddImplementationArg(op_entry, 'RTW.Tf1ArgNumeric',...
                              'Name',      'u1', ...
                              'IOType',    'RTW_IO_INPUT',...
                              'IsSigned',  true,...
                              'WordLength', 32, ...
                              'FractionLength', 0 );
```

```
createAndAddImplementationArg(op_entry, 'RTW.Tf1ArgNumeric',...
                              'Name',      'u2', ...
                              'IOType',    'RTW_IO_INPUT',...
                              'IsSigned',  true,...
                              'WordLength', 32, ...
                              'FractionLength', 0 );
```

The following examples show some common type specifications using `createAndSetCImplementationReturn`.

```
% uint8:
createAndSetCImplementationReturn(hEntry, 'RTW.Tf1ArgNumeric', ...
                                   'Name',      'y1', ...
                                   'IOType',    'RTW_IO_OUTPUT', ...
                                   'IsSigned',  false, ...
                                   'WordLength', 8, ...
                                   'FractionLength', 0 );
```

```
% single:
createAndSetCImplementationReturn(hEntry, 'RTW.Tf1ArgNumeric', ...
                                   'Name',      'y1', ...
                                   'IOType',    'RTW_IO_OUTPUT', ...
                                   'DataTypeMode', 'single' );
```

```
% double:
createAndSetCImplementationReturn(hEntry, 'RTW.Tf1ArgNumeric', ...
                                   'Name',      'y1', ...
                                   'IOType',    'RTW_IO_OUTPUT', ...
                                   'DataTypeMode', 'double' );
```

createAndSetCImplementationReturn

```
% boolean:  
createAndSetCImplementationReturn(hEntry, 'RTW.Tf1ArgNumeric', ...  
                                   'Name',      'y1', ...  
                                   'IOType',    'RTW_IO_OUTPUT', ...  
                                   'DataTypeMode', 'boolean' );
```

See Also

`createAndAddImplementationArg`

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

arxml.importer.createCalibrationComponentObjects

Purpose	Create Simulink calibration objects from AUTOSAR calibration component	
Syntax	<pre>importerObj.createCalibrationComponentObjects(componentName) [success] = createCalibrationComponentObjects(importerObj. com ponentName, CreateSimulinkObject, 'true')</pre>	
Description	<p><code>importerObj.createCalibrationComponentObjects(componentName)</code> creates Simulink calibration objects from an AUTOSAR calibration component. This imports all your parameters into the Workspace and you can then assign them to block parameters in your Simulink model.</p>	
Inputs	<p><i>componentName</i></p> <p>CreateSimulinkObject, 'true'</p>	<p>Absolute short name path of calibration parameter component.</p> <p>Optional property/value pair. The property CreateSimulinkObject can be either true or false (default is true). If it is true, then:</p> <pre>[success] = createCalibrationComponentObjects(importerObj. componentName, CreateSimulinkObject, 'true') creates the Simulink.AliasType and Simulink.NumericType corresponding to the AUTOSAR data types described in the XML file imported by importerObj.</pre>
Outputs	<p><i>success</i></p>	<p>True if function is successful. False otherwise.</p>
Examples	<pre>importer_obj.createCalibrationComponentObjects('/package/autosar_component2')</pre>	
See Also	<p>“Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation</p>	

arxml.importer.createComponentAsModel

Purpose Create AUTOSAR atomic software component as Simulink model

Syntax

```
[modelH,  
    success] = importerObj.createComponentAsModel(ComponentName  
    )  
[modelH,  
    success] = importerObj.createComponentAsModel(ComponentName  
    , Property1, Value1, Property2, Value2, ...)
```

Description `[modelH, success] = importerObj.createComponentAsModel(ComponentName)` creates a Simulink model corresponding to the AUTOSAR atomic software component 'COMPONENT' described in the XML file imported by the `arxml.importer` object `importerObj`.

You can also specify optional property/value pairs when creating this Simulink model:

```
[modelH, success] =  
importerObj.createComponentAsModel(ComponentName,  
Property1, Value1, Property2, Value2, ...)
```

Inputs

<i>ComponentName</i>	Absolute short name path of the atomic software component.
<i>PropertyN, ValueN</i>	Optional property/value pairs. You can specify values for the following properties: <code>CreateSimulinkObject</code> 'true' (default) or 'false'. If 'true', then the function creates the <code>Simulink.AliasType</code> and <code>Simulink.NumericType</code> corresponding to the AUTOSAR data types in the XML file.

NameConflictAction

'overwrite' (default) or
'makenameunique' or 'error'.

Use this property to determine the action if a Simulink model with the same name as the component already exists.

AutoSave

'true' or 'false' (default). If 'true', then the function automatically saves the generated Simulink model.

Outputs

modelH

Model handle.

success

True if the function is successful. Otherwise, it is false.

Examples

```
importer_obj.createComponentAsModel('/package/autosar_component2')
```

See Also

“Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

arxml.importer.createComponentAsSubsystem

Purpose Create AUTOSAR atomic software component as Simulink atomic subsystem

Syntax

```
[susbsysH,  
 success] = importerObj.createComponentAsSubsystem(Component  
 Name)  
[susbsysH,  
 success] = importerObj.createComponentAsSubsystem(Component  
 Name, Property1, Value1, Property2, Value2, ...)
```

Description [susbsysH, success] = *importerObj*.createComponentAsSubsystem(*ComponentName*) creates a Simulink subsystem corresponding to the AUTOSAR atomic software component 'COMPONENT' described in the XML file imported by the arxml.importer object *importerObj*.

You can also specify optional property/value pairs when creating this Simulink subsystem:

```
[susbsysH, success] =  
importerObj.createComponentAsSubsystem(ComponentName,  
Property1, Value1, Property2, Value2, ...)
```

You can perform AUTOSAR configuration and code generation on atomic subsystems or function call subsystems. These subsystems must be convertible to model reference blocks by using the method:

```
Simulink.SubSystem.convertToModelReference
```

Note The AUTOSAR target automatically checks that the subsystem meets this requirement when you perform a subsystem build.

You do not have to convert your subsystem to a model reference block; it is optional. If you convert your subsystem to a referenced model, you can configure AUTOSAR options within the referenced model.

You can *export functions* for a single function-call subsystem. First configure your function-call subsystem AUTOSAR options (e.g., using the GUI from the Configuration Parameters dialog or by calling `autosar_gui_launch(subsystemName)`). Then right-click the subsystem and select **Real-Time Workshop > Export Functions**.

Inputs

<i>ComponentName</i>	Absolute short name path of the atomic software component .
<i>PropertyN, ValueN</i>	Optional property/value pairs. You can specify values for the following properties: CreateSimulinkObject Boolean value, true or false (default is true). If true, the function creates the <code>Simulink.AliasType</code> and <code>Simulink.NumericType</code> corresponding to the AUTOSAR data types in the XML file. NameConflictAction 'overwrite' (default) or 'makeunique' or 'error' . Use this property to determine the action to take if a Simulink model with the same name as the component already exists. AutoSave Boolean value, true or false (default is false). If true, the function automatically saves the generated Simulink model.

arxml.importer.createComponentAsSubsystem

Outputs

<i>subsysH</i>	Subsystem handle.
<i>success</i>	True if the function is successful. Otherwise, it is false.

Examples

```
importer_obj.createComponentAsSubsystem('/package/autosar_component2')
```

See Also

“Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

arxml.importer.createOperationAsConfigurableSubsystem

Purpose Create configurable Simulink subsystem library for client-server operation

Syntax

```
[modelH,  
 success] = importerObj.createOperationAsConfigurableSubsystems(interfaceName)  
[modelH,  
 success] = importerObj.createOperationAsConfigurableSubsystems(InterfaceName, Property1, Value1, Property2, Value2,  
 ...)
```

Description

`[modelH, success] = importerObj.createOperationAsConfigurableSubsystems(interfaceName)` creates a configurable Simulink subsystem library corresponding to the AUTOSAR client-server interface 'INTERFACE'. This interface is described in the XML file imported by the `arxml.importer` object `importerObj`.

You can also specify optional property/value pairs when creating this Simulink subsystem library:

```
[modelH, success] =  
importerObj.createOperationAsConfigurableSubsystems(InterfaceName,  
Property1, Value1, Property2, Value2, ...)
```

Inputs

`interfaceName` Absolute short name path of the client-server interface.

`PropertyN, ValueN` Optional property/value pairs. You can specify values for the following properties:

`CreateSimulinkObject`

True (default) or false. If true, then the function creates the `Simulink.AliasType` and `Simulink.NumericType` corresponding

arxml.importer.createOperationAsConfigurableSubsystems

to the AUTOSAR data types in the XML file.

NameConflictAction

'overwrite' (default) or 'makenameunique' or 'error'.

Use this property to determine the action if a Simulink model with the same name as the component already exists.

AutoSave

True or false (default). If true, then the function automatically saves the generated Simulink subsystem library.

ForceClientBlkForBSP

True or false (default). If true, an Invoke AUTOSAR Server Operation block is created for a single argument operation that accesses Basic Software.

Outputs

modelH

Model handle.

success

True if the function is successful. False otherwise.

Examples

```
obj.createOperationAsConfigurableSubsystems('/PortInterface/csinterface')
```

See Also

“Importing an AUTOSAR Software Component” and “Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation

Purpose	Display results of comparing configuration parameter values
Syntax	<code>cfgObj.displayReport()</code>
Description	<code>cfgObj.displayReport()</code> displays the results at the MATLAB Command Window of comparing the configuration parameter values for the model with the values that the object recommends. <code>cfgObj</code> is a handle to a <code>cgv.Config</code> object.
Tutorials	<ul style="list-style-type: none">• “How To Verify a SIL or PIL Configuration”
How To	<ul style="list-style-type: none">• “Configuring a PIL Simulation”

rtw.codegenObjectives.Objective.excludeCheck

Purpose Exclude checks

Syntax `excludeCheck(obj, checkID)`

Description `excludeCheck(obj, checkID)` excludes a check from the Code Generation Advisor when a user specifies the objective. When a user selects multiple objectives, if the user specifies an additional objective that includes this check as a higher priority objective, the Code Generation Advisor displays this check.

Inputs

<i>obj</i>	Handle to a code generation objective object previously created.
<i>checkID</i>	Unique identifier of the check that you exclude from the new objective.

Examples Exclude the **Disable signal logging** check from the objective.

```
excludeCheck(obj, 'com.mathworks.MA.CheckDisableSignalLogging');
```

See Also `Simulink.ModelAdvisor`

How To

- “Creating Custom Objectives”
- “About IDs”

RTW.ModelCPPArgsClass.getArgCategory

Purpose	Get argument category for Simulink model port from model-specific C++ encapsulation interface				
Syntax	<code>category = getArgCategory(obj, portName)</code>				
Description	<code>category = getArgCategory(obj, portName)</code> gets the category — 'Value', 'Pointer', or 'Reference' — of the argument corresponding to a specified Simulink model inport or output from a specified model-specific C++ encapsulation interface.				
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code>.</td></tr><tr><td><code>portName</code></td><td>String specifying the name of an inport or output in your Simulink model.</td></tr></table>	<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .	<code>portName</code>	String specifying the name of an inport or output in your Simulink model.
<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .				
<code>portName</code>	String specifying the name of an inport or output in your Simulink model.				
Outputs	<table><tr><td><code>category</code></td><td>String specifying the argument category — 'Value', 'Pointer', or 'Reference' — for the specified Simulink model port.</td></tr></table>	<code>category</code>	String specifying the argument category — 'Value', 'Pointer', or 'Reference' — for the specified Simulink model port.		
<code>category</code>	String specifying the argument category — 'Value', 'Pointer', or 'Reference' — for the specified Simulink model port.				
Alternatives	To view argument categories in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display step method argument categories. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.				

RTW.ModelCPPArgsClass.getArgCategory

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getArgCategory

Purpose	Get argument category for Simulink model port from model-specific C function prototype	
Syntax	<code>category = getArgCategory(obj, portName)</code>	
Description	<code>category = getArgCategory(obj, portName)</code> gets the category, 'Value' or 'Pointer', of the argument corresponding to a specified Simulink model inport or outport from a specified model-specific C function prototype.	
Inputs	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .
	<i>portName</i>	String specifying the name of an inport or outport in your Simulink model.
Outputs	<i>category</i>	String specifying the argument category, 'Value' or 'Pointer', for the specified Simulink model port.
Alternatives	Click the Get Default Configuration button in the Model Interface dialog box to get argument categories. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code	

RTW.ModelCPPArgsClass.getArgName

Purpose Get argument name for Simulink model port from model-specific C++ encapsulation interface

Syntax `argName = getArgName(obj, portName)`

Description `argName = getArgName(obj, portName)` gets the argument name corresponding to a specified Simulink model inport or outport from a specified model-specific C++ encapsulation interface.

Inputs

<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.

Outputs

<code>argName</code>	String specifying the argument name for the specified Simulink model port.
----------------------	--

Alternatives To view argument names in the Simulink Configuration Parameters graphical user interface, go to the **Interface** pane and click the **Configure C++ Encapsulation Interface** button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments **step method** view of this dialog box, click the **Get Default Configuration** button to display step method argument names. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also “Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class”
in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getArgName

Purpose	Get argument name for Simulink model port from model-specific C function prototype	
Syntax	<code>argName = getArgName(obj, portName)</code>	
Description	<code>argName = getArgName(obj, portName)</code> gets the argument name corresponding to a specified Simulink model inport or outport from a specified model-specific C function prototype.	
Inputs	<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .
	<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.
Outputs	<code>argName</code>	String specifying the argument name for the specified Simulink model port.
Alternatives	Click the Get Default Configuration button in the Model Interface dialog box to get argument names. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code	

RTW.ModelCPPArgsClass.getArgPosition

Purpose	Get argument position for Simulink model port from model-specific C++ encapsulation interface	
Syntax	<code>position = getArgPosition(obj, portName)</code>	
Description	<code>position = getArgPosition(obj, portName)</code> gets the position — 1 for first, 2 for second, etc. — of the argument corresponding to a specified Simulink model inport or outport from a specified model-specific C++ encapsulation interface.	
Inputs	<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
	<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.
Outputs	<code>position</code>	Integer specifying the argument position — 1 for first, 2 for second, etc. — for the specified Simulink model port. If there is no argument for the specified port, the function returns 0.
Alternatives	To view argument positions in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display step method argument positions. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.	

RTW.ModelCPPArgsClass.getArgPosition

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getArgPosition

Purpose	Get argument position for Simulink model port from model-specific C function prototype				
Syntax	<code>position = getArgPosition(obj, portName)</code>				
Description	<code>position = getArgPosition(obj, portName)</code> gets the position — 1 for first, 2 for second, etc. — of the argument corresponding to a specified Simulink model inport or outport from a specified model-specific C function prototype.				
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code>.</td></tr><tr><td><code>portName</code></td><td>String specifying the name of an inport or outport in your Simulink model.</td></tr></table>	<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .	<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.
<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .				
<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.				
Outputs	<table><tr><td><code>position</code></td><td>Integer specifying the argument position — 1 for first, 2 for second, etc. — for the specified Simulink model port. If no argument is found for the specified port, the function returns 0.</td></tr></table>	<code>position</code>	Integer specifying the argument position — 1 for first, 2 for second, etc. — for the specified Simulink model port. If no argument is found for the specified port, the function returns 0.		
<code>position</code>	Integer specifying the argument position — 1 for first, 2 for second, etc. — for the specified Simulink model port. If no argument is found for the specified port, the function returns 0.				
Alternatives	Click the Get Default Configuration button in the Model Interface dialog box to get argument positions. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.				
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code				

RTW.ModelCPPArgsClass.getArgQualifier

Purpose Get argument type qualifier for Simulink model port from model-specific C++ encapsulation interface

Syntax `qualifier = getArgQualifier(obj, portName)`

Description `qualifier = getArgQualifier(obj, portName)` gets the type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — of the argument corresponding to a specified Simulink model inport or outport from a specified model-specific C++ encapsulation interface.

Inputs

<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.

Outputs

<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — for the specified Simulink model port.
------------------------	--

Alternatives To view argument qualifiers in the Simulink Configuration Parameters graphical user interface, go to the **Interface** pane and click the **Configure C++ Encapsulation Interface** button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the **Get Default Configuration** button to display step method argument qualifiers. For more information, see “Configuring the Step Method

for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getArgQualifier

Purpose Get argument type qualifier for Simulink model port from model-specific C function prototype

Syntax `qualifier = getArgQualifier(obj, portName)`

Description `qualifier = getArgQualifier(obj, portName)` gets the type qualifier — 'none', 'const', 'const *', or 'const * const'— of the argument corresponding to a specified Simulink model inport or outport from a specified model-specific C function prototype.

Inputs

<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .
<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.

Outputs

<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const'— for the specified Simulink model port.
------------------------	--

Alternatives Click the **Get Default Configuration** button in the Model Interface dialog box to get argument qualifiers. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.

See Also “Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

arxml.importer.getCalibrationComponentNames

Purpose

Get calibration component names

Syntax

```
calibrationComponentNames = importerObj.getCalibrationComponentNames
```

Description

calibrationComponentNames = *importerObj*.getCalibrationComponentNames returns the list of calibration component names found in the XML files associated with the arxml.importer object, *importerObj*.

Outputs

calibrationComponentNames

Cell array of strings in which each element is the absolute short name path of the corresponding calibration parameter component :

```
'/root_package_name[/sub_package_name]/component_short_name'
```

See Also

“Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPClass.getClassName

Purpose	Get class name from model-specific C++ encapsulation interface	
Syntax	<code>clsName = getClassName(obj)</code>	
Description	<code>clsName = getClassName(obj)</code> gets the name of the class described by the specified model-specific C++ encapsulation interface.	
Inputs	<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
Outputs	<code>clsName</code>	A string specifying the name of the class described by the specified model-specific C++ encapsulation interface.
Alternatives	To view the model class name in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, which displays the model class name and allows you to display and configure the step method for your model class. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation “Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation “Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation	

RTW.AutosarInterface.getComponentName

Purpose Get XML component name

Syntax `componentName = autosarInterfaceObj.getComponentName`

Description `componentName = autosarInterfaceObj.getComponentName` gets the XML component name of the model-specific RTW.AutosarInterface object defined by `autosarInterfaceObj`.

Outputs `componentName` Name of XML component object defined by `autosarInterfaceObj`.

See Also “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

arxml.importer.getComponentNames

Purpose Get atomic software component names

Syntax `componentNames = importerObj.getComponentNames`

Description `componentNames = importerObj.getComponentNames` returns the list of atomic software component names in the XML file associated with the `arxml.importer` object, `importerObj`.

Note `getComponentNames` finds only the atomic software component defined in the XML file specified when constructing the `arxml.importer` object or the XML file specified by the method `setFile`. All atomic software components described in the XML file dependencies are ignored.

Outputs `componentNames` Cell array of strings in which each element is the absolute short name path of the corresponding atomic software component :

`'/root_package_name[/sub_package_name]/component_short_name'`

See Also “Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getDataTypePackageName

- Purpose** Get XML data type package name
- Syntax** `dataTypePackageName = getDataTypePackageName(autosarInterfaceObj)`
- Description** `dataTypePackageName = getDataTypePackageName(autosarInterfaceObj)` gets the XML data type package name of *autosarInterfaceObj*, a model-specific RTW.AutosarInterface object.
- Outputs** `dataTypePackageName` Name of the data type package specified by *autosarInterfaceObj*.
- See Also** “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getDefaultConf

Purpose Get default configuration

Syntax `autosarInterfaceObj.getDefaultConf`

Description `autosarInterfaceObj.getDefaultConf` gets the model's default configuration for `autosarInterfaceObj`, using information from the model to which `autosarInterfaceObj` is attached.

`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object. You must attach the object to a model using `attachToModel` before calling `getDefaultConf`.

When you initially invoke `getDefaultConf` (or the GUI button equivalent, **Get Default Configuration** in the Model Interface dialog), the runnable names, XML properties, and I/O configuration are initialized. If you invoke the command (or click the button) again, only the I/O configurations are reset to default values.

See Also “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

Purpose	Get default configuration information for model-specific C++ encapsulation interface from Simulink model		
Syntax	<code>getDefaultConf(obj)</code>		
Description	<p><code>getDefaultConf(obj)</code> initializes the specified model-specific C++ encapsulation interface to a default configuration, based on information from the ERT-based Simulink model to which the interface is attached. On the first invocation, class and step method names and step method properties are set to default values. On subsequent invocations, only step method properties are reset to default values.</p> <p>Before calling this function, you must call <code>attachToModel</code>, to attach the C++ encapsulation interface to a loaded model.</p>		
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.ModelCPPVoidClass</code>.</td></tr></table>	<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> .
<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> .		
Alternatives	To view C++ encapsulation interface default configuration information in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display default configuration information. In the void-void step method view, you can see the default configuration information without clicking a button. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.		

RTW.ModelCPPClass.getDefaultConf

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getDefaultConf

Purpose	Get default configuration information for model-specific C function prototype from Simulink model		
Syntax	<code>getDefaultConf(obj)</code>		
Description	<p><code>getDefaultConf(obj)</code> invokes the specified model-specific C function prototype to initialize the properties and the step function name of the function argument to a default configuration based on information from the ERT-based Simulink model to which it is attached. If you invoke the command again, only the properties of the function argument are reset to default values.</p> <p>Before calling this function, you must call <code>attachToModel</code>, to attach the function prototype to a loaded model.</p>		
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code>.</td></tr></table>	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> .
<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> .		
Alternatives	Click the Get Default Configuration button in the Model Interface dialog box to get the default configuration. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.		
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code		

arxml.importer.getDependencies

Purpose	Get list of XML dependency files
Syntax	<i>Dependencies</i> = <i>importerObj</i> .getDependencies()
Description	<i>Dependencies</i> = <i>importerObj</i> .getDependencies() returns the list of XML dependency files associated with the arxml.importer object, <i>importerObj</i> .
Outputs	<i>Dependencies</i> Cell array of strings.
See Also	“Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getExecutionPeriod

Purpose	Get runnable execution period		
Syntax	<code>EP = autosarInterfaceObj.getExecutionPeriod</code>		
Description	<p><code>EP = autosarInterfaceObj.getExecutionPeriod</code> gets the execution period in the configuration.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p>		
Outputs	<table><tr><td>EP</td><td>Execution period in the configuration</td></tr></table>	EP	Execution period in the configuration
EP	Execution period in the configuration		
See Also	“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation		

arxml.importer.getFile

Purpose Return XML file name for `arxml.importer` object

Syntax `filename = importerObj.getFile`

Description `filename = importerObj.getFile` returns the name of the XML file associated with the `arxml.importer` object, `importerObj`.

Outputs `filename` XML file name

See Also “Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getFunctionName

Purpose	Get function name from model-specific C function prototype	
Syntax	<code>fcnName = getFunctionName(obj, fcnType)</code>	
Description	<code>fcnName = getFunctionName(obj, fcnType)</code> gets the name of the step or initialize function described by the specified model-specific C function prototype.	
Inputs	<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .
	<code>fcnType</code>	Optional string specifying which function name to get. Valid strings are 'step' and 'init'. If <code>fcnType</code> is not specified, gets the step function name.
Outputs	<code>fcnName</code>	A string specifying the name of the function described by the specified model-specific C function prototype.
Alternatives	Click the Get Default Configuration button in the Model Interface dialog box to get function names. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code	

RTW.AutosarInterface.getImplementationName

Purpose Get XML implementation name

Syntax `implementationName = autosarInterfaceObj.getImplementationName`

Description `implementationName = autosarInterfaceObj.getImplementationName` gets the XML implementation name of `autosarInterfaceObj`, a model-specific `RTW.AutosarInterface` object.

See Also “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getInitEventName

Purpose	Get initial event name
Syntax	<code>initEventName = autosarInterfaceObj.getInitEventName</code>
Description	<code>initEventName = autosarInterfaceObj.getInitEventName</code> gets the initial event name of <code>autosarInterfaceObj</code> , a model-specific RTW.AutosarInterface object.
Outputs	<code>initEventName</code> Name of the initial event specified by <code>autosarInterfaceObj</code> .
See Also	“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getInitRunnableName

Purpose Get initial runnable name

Syntax `initRunnableName = autosarInterfaceObj.getInitRunnableName`

Description `initRunnableName = autosarInterfaceObj.getInitRunnableName` gets the initial runnable name of `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.

Outputs `initRunnableName` Name of the initial runnable specified by `autosarInterfaceObj`.

See Also “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getInterfacePackageName

- Purpose** Get XML interface package name
- Syntax** `interfacePkgName = autosarInterfaceObj.getInterfacePackageName`
- Description** `interfacePkgName = autosarInterfaceObj.getInterfacePackageName` gets the XML interface package name of `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.
- Outputs** `interfacePkgName` Name of the interface package specified by `autosarInterfaceObj`.
- See Also** “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getInternalBehaviorName

Purpose	Get XML internal behavior name
Syntax	<code>internalBehaviorName = autosarInterfaceObj.getInternalBehaviorName</code>
Description	<code>internalBehaviorName = autosarInterfaceObj.getInternalBehaviorName</code> gets the XML internal behavior name of <code>autosarInterfaceObj</code> , a model-specific RTW.AutosarInterface object.
Outputs	<code>internalBehaviorName</code> Name of the internal behavior specified by <code>autosarInterfaceObj</code> .
See Also	“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getIOAutosarPortName

Purpose Get I/O AUTOSAR port name

Syntax `ioAutosarName = autosarInterfaceObj.getIOAutosarPortName(portName)`

Description `ioAutosarName = autosarInterfaceObj.getIOAutosarPortName(portName)` gets the I/O AUTOSAR port name in the configuration for the port corresponding to `portName`.

`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.

By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.

Inputs `portName` Name of inport/outport name (string).

Outputs `ioAutosarName` AUTOSAR port name of `portName`

See Also “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getIODataAccessMode

Purpose	Get I/O data access mode	
Syntax	<code>dataAccessMode = autosarInterfaceObj.getIODataAccessMode(portName)</code>	
Description	<code>dataAccessMode = autosarInterfaceObj.getIODataAccessMode(portName)</code> gets the data access mode of the I/O corresponding to <code>portName</code> , for <code>autosarInterfaceObj</code> , a model-specific RTW.AutosarInterface object.	
Inputs	<code>portName</code>	Name of inport/outport (string).
Outputs	<code>dataAccessMode</code>	Data access mode of the given port. Can be one of the following: <ul style="list-style-type: none">• 'ImplicitSend'• 'ImplicitReceive'• 'ExplicitSend'• 'ExplicitReceive'
See Also	“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation	

RTW.AutosarInterface.getIODataElement

Purpose Get I/O data element name

Syntax `ioDataElement = autosarInterfaceObj.getIODataElement(portName)`
`)`

Description `ioDataElement = autosarInterfaceObj.getIODataElement(portName)` gets the I/O data element name in the configuration for the port corresponding to `portName`.

`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.

By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.

Inputs `portName` Name of inport/outport (string).

Outputs `ioDataElement` Data element of the given port (string).

See Also “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getIOErrorStatusReceiver

Purpose Get receiver port name

Syntax `ESR = autosarInterfaceObj.getIOErrorStatusReceiver(PortName)`

Description `ESR = autosarInterfaceObj.getIOErrorStatusReceiver(PortName)` gets the receiver port name in the configuration for the port corresponding to `PortName` .

`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.

Inputs `PortName` Name of inport/outport (string).

Outputs `ESR` Name of receiver port for `PortName`.

See Also “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getIOInterfaceName

Purpose	Get I/O interface name		
Syntax	<pre><i>ioInterfaceName</i> = <i>autosarInterfaceObj</i>.getIOInterfaceName(<i>portName</i>)</pre>		
Description	<p><i>ioInterfaceName</i> = <i>autosarInterfaceObj</i>.getIOInterfaceName(<i>portName</i>) gets the I/O interface name in the configuration for the port corresponding to <i>portName</i>.</p> <p><i>autosarInterfaceObj</i> is a model-specific RTW.AutosarInterface object.</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>		
Inputs	<table><tr><td><i>portName</i></td><td>Name of the inport/outport (string).</td></tr></table>	<i>portName</i>	Name of the inport/outport (string).
<i>portName</i>	Name of the inport/outport (string).		
Outputs	<table><tr><td><i>ioInterfaceName</i></td><td>Name of the I/O interface for <i>portName</i>.</td></tr></table>	<i>ioInterfaceName</i>	Name of the I/O interface for <i>portName</i> .
<i>ioInterfaceName</i>	Name of the I/O interface for <i>portName</i> .		
See Also	“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation		

RTW.AutosarInterface.getIOPortNumber

Purpose Get I/O AUTOSAR port number

Syntax `IOPortNumber= autosarInterfaceObj.getIOPortNumber(PortName)`

Description `IOPortNumber= autosarInterfaceObj.getIOPortNumber(PortName)` gets the I/O AUTOSAR port number in the configuration for the port corresponding to *PortName*.

autosarInterfaceObj is a model-specific RTW.AutosarInterface object.

Inputs *PortName* Name of the inport/output (string).

Outputs *IOPortNumber* Port number of *PortName*.

See Also “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getIOServiceInterface

Purpose	Get port I/O service interface	
Syntax	<i>SI</i> = <i>autosarInterfaceObj</i> .getIOServiceInterface(<i>PortName</i>)	
Description	<i>SI</i> = <i>autosarInterfaceObj</i> .getIOServiceInterface(<i>PortName</i>) gets the I/O service interface in the configuration for the port corresponding to <i>PortName</i> . <i>autosarInterfaceObj</i> is a model-specific RTW.AutosarInterface object.	
Inputs	<i>PortName</i>	Name of the inport/outport (string).
Outputs	<i>SI</i>	I/O service interface of <i>PortName</i> .
See Also	“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation	

RTW.AutosarInterface.getIOServiceName

Purpose	Get port I/O service name	
Syntax	<i>SN</i> = <i>autosarInterfaceObj</i> .getIOServiceName(<i>PortName</i>)	
Description	<i>SN</i> = <i>autosarInterfaceObj</i> .getIOServiceName(<i>PortName</i>) gets the I/O service name in the configuration for the port corresponding to <i>PortName</i> . <i>autosarInterfaceObj</i> is a model-specific RTW.AutosarInterface object.	
Inputs	<i>PortName</i>	Name of the inport/outport (string).
Outputs	<i>SN</i>	Name of I/O service for <i>PortName</i> .
See Also	“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation	

RTW.AutosarInterface.getIOServiceOperation

Purpose Get port I/O service operation

Syntax `SO = autosarInterfaceObj.getIOServiceOperation(PortName)`

Description `SO = autosarInterfaceObj.getIOServiceOperation(PortName)` gets the I/O service operation in the configuration for the port corresponding to `PortName`.

`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.

Inputs

<code>PortName</code>	Inport/outport name (string).
-----------------------	-------------------------------

Outputs

<code>SO</code>	I/O service operation of <code>PortName</code> .
-----------------	--

See Also “Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getIsServerOperation

Purpose	Determine whether server is specified		
Syntax	<code>isServerOperation = autosarInterfaceObj.getIsServerOperation</code>		
Description	<p><code>isServerOperation = autosarInterfaceObj.getIsServerOperation</code> returns the value of the property '<code>isServerOperation</code>' in <code>autosarInterfaceObj</code>.</p> <p><code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.</p>		
Outputs	<table><tr><td><code>isServerOperation</code></td><td>True or false. If true, a server is specified in <code>autosarInterfaceObj</code>.</td></tr></table>	<code>isServerOperation</code>	True or false. If true, a server is specified in <code>autosarInterfaceObj</code> .
<code>isServerOperation</code>	True or false. If true, a server is specified in <code>autosarInterfaceObj</code> .		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

Purpose	Get number of step method arguments from model-specific C++ encapsulation interface		
Syntax	<code>num = getNumArgs(obj)</code>		
Description	<code>num = getNumArgs(obj)</code> gets the number of arguments for the step method described by the specified model-specific C++ encapsulation interface.		
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code>.</td></tr></table>	<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .		
Outputs	<table><tr><td><code>num</code></td><td>An integer specifying the number of step method arguments.</td></tr></table>	<code>num</code>	An integer specifying the number of step method arguments.
<code>num</code>	An integer specifying the number of step method arguments.		
Alternatives	To view the number of step method arguments in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display the step method arguments. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.		
See Also	“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation		

RTW.ModelCPPClass.getNumArgs

“Sample M-Script for Configuring the Step Method for a Model Class”
in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.getNumArgs

Purpose	Get number of function arguments from model-specific C function prototype		
Syntax	<code>num = getNumArgs(obj)</code>		
Description	<code>num = getNumArgs(obj)</code> gets the number of function arguments for the function described by the specified model-specific C function prototype.		
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code>.</td></tr></table>	<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .
<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .		
Outputs	<table><tr><td><code>num</code></td><td>An integer specifying the number of function arguments.</td></tr></table>	<code>num</code>	An integer specifying the number of function arguments.
<code>num</code>	An integer specifying the number of function arguments.		
Alternatives	Click the Get Default Configuration button in the Model Interface dialog box to get arguments. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.		
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code		

cgv.CGV.getOutputData

Purpose Get output data

Syntax `out = cgvObj.getOutputData(InputIndex)`

Description `out = cgvObj.getOutputData(InputIndex)` is the method that you use to retrieve the output data that the object creates during execution of the model. `out` is the output data that the object returns. `cgvObj` is a handle to a `cgv.CGV` object. `InputIndex` is a unique numeric identifier that specifies which output data to retrieve. The `InputIndex` is associated with specific input data.

How To • “Verifying Numerical Equivalence of Results with Code Generation Verification”

RTW.AutosarInterface.getPeriodicEventName

Purpose	Get periodic event name
Syntax	<code>periodicEventName = autosarInterfaceObj.getPeriodicEventName</code>
Description	<code>periodicEventName = autosarInterfaceObj.getPeriodicEventName</code> gets the periodic event name specified by the model-specific RTW.AutosarInterface object, <code>autosarInterfaceObj</code> .
Outputs	<code>periodicEventName</code> Name of the periodic event specified by <code>autosarInterfaceObj</code>
Examples	For multiple runnables, use the Children property to access each individual runnable after building or GUI update, for example: <code>autosarInterfaceObj.Children(1).getPeriodicEventName()</code>
See Also	“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getPeriodicRunnableName

Purpose

Get periodic runnable name

Syntax

```
periodicRunnableName = autosarInterfaceObj.getPeriodicRunnableName
```

Description

```
periodicRunnableName =  
autosarInterfaceObj.getPeriodicRunnableName
```

gets the name of the periodic runnable specified in `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.

Outputs

<i>periodicRunnableName</i>	Name of the periodic runnable specified by <i>autosarInterfaceObj</i> .
-----------------------------	---

Examples

For multiple runnables, use the Children property to access each individual runnable after building or GUI update, for example:

```
autosarInterfaceObj.Children(1).getPeriodicRunnableName()
```

See Also

“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getPortDefaultConf

Purpose	Get port default configuration	
Syntax	<pre>[autosarPort, interfaceName, dataElement, dataAccessMode]=autosarInterfaceObj.getPortDefaultConf(portH)</pre>	
Description	<pre>[autosarPort, interfaceName, dataElement, dataAccessMode]=autosarInterfaceObj.getPortDefaultConf(portH)</pre> gets the port's default configuration for <i>autosarInterfaceObj</i> , a model-specific RTW.AutosarInterface object.	
Inputs	<i>portH</i>	Port of <i>autosarInterfaceObj</i> .
Outputs	<i>autosarPort</i>	AUTOSAR port name for <i>portH</i> .
	<i>interfaceName</i>	Name I/O interface for <i>portH</i> (string).
	<i>dataElement</i>	Data element of <i>portH</i> (string).
	<i>dataAccessMode</i>	Data access mode of <i>portH</i> , and is one of the following: <ul style="list-style-type: none">• 'ImplicitSend'• 'ImplicitReceive'• 'ExplicitSend'• 'ExplicitReceive'
See Also	“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation	

RTW.ModelSpecificCPrototype.getPreview

Purpose	Get model-specific C function prototype code preview	
Syntax	<code>preview = getPreview(obj, fcnType)</code>	
Description	<code>preview = getPreview(obj, fcnType)</code> gets the model-specific C function prototype code preview.	
Inputs	<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modelName)</code> .
	<code>fcnType</code>	Optional. String specifying which function to preview. Valid strings are 'step' and 'init'. If <code>fcnType</code> is not specified, previews the step function.
Outputs	<code>preview</code>	String specifying the function prototype for the step or initialization function.
Alternatives	Use the Step function preview subpane in the Model Interface dialog box to preview how your step function prototype is interpreted in generated code. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code	

Purpose Return results of comparing configuration parameter values

Syntax `rpt_data = cfgObj.getReportData()`

Description `rpt_data = cfgObj.getReportData()` compares the original configuration parameter values with the values that the object recommends. `cfgObj` is a handle to a `cgv.Config` object. Returns a cell array of strings with the model, parameter, previous value, and recommended or new value.

Tutorials

- “How To Verify a SIL or PIL Configuration”

How To

- “Configuring a PIL Simulation”

RTW.AutosarInterface.getServerInterfaceName

Purpose Get name of server interface

Syntax `serverInterfaceName = autosarInterfaceObj.getServerInterfaceName`

Description `serverInterfaceName = autosarInterfaceObj.getServerInterfaceName` returns the name of the server interface specified in `autosarInterfaceObj`.
`autosarInterfaceObj` is a model-specific RTW.AutosarInterface object.

Outputs `serverInterfaceName` Name of the server interface in `autosarInterfaceObj`.

See Also “Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getServerOperationPrototype

Purpose	Get server operation prototype	
Syntax	<code>operation_prototype = autosarInterfaceObj.getServerOperationPr ototype</code>	
Description	<code>operation_prototype = autosarInterfaceObj.getServerOperationPrototype</code> returns the server operation prototype in <code>autosarInterfaceObj</code> . <code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.	
Outputs	<code>operation_prototype</code>	String with names of prototype and arguments: <code>operation_name(dir1 datatype1 arg1, dir2 datatype2 arg2, ..., dirN datatypeN argN, ...)</code> <ul style="list-style-type: none">• <code>operation_name</code> — Name of the operation• <code>dirN</code> — Either IN or OUT, which indicates whether data is passed in or out of the function.• <code>datatypeN</code> — Data type, which can be an AUTOSAR basic data type or record, Simulink data type, or array.• <code>argN</code> — Name of the argument
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation	

RTW.AutosarInterface.getServerPortName

Purpose Get server port name

Syntax `serverPortName = autosarInterfaceObj.getServerPortName`

Description `serverPortName = autosarInterfaceObj.getServerPortName` returns the server port name of the model-specific RTW.AutosarInterface object defined by `autosarInterfaceObj`.

Outputs

<code>serverPortName</code>	Name of the server port defined by <code>autosarInterfaceObj</code> .
-----------------------------	---

See Also “Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.getServerType

Purpose	Determine server type		
Syntax	<code>serverType = autosarInterfaceObj.getServerType</code>		
Description	<p><code>serverType = autosarInterfaceObj.getServerType</code> determines the type of the server in <code>autosarInterfaceObj</code>, that is, whether it is application software or Basic software.</p> <p><code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.</p>		
Outputs	<table><tr><td><code>serverType</code></td><td>Either 'Application software' or 'Basic software'.</td></tr></table>	<code>serverType</code>	Either 'Application software' or 'Basic software'.
<code>serverType</code>	Either 'Application software' or 'Basic software'.		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

RTW.ModelCPPClass.getStepMethodName

Purpose Get step method name from model-specific C++ encapsulation interface

Syntax `fcnName = getStepMethodNameName(obj)`

Description `fcnName = getStepMethodNameName(obj)` gets the name of the step method described by the specified model-specific C++ encapsulation interface.

Inputs

<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
------------------	---

Outputs

<code>fcnName</code>	A string specifying the name of the step method described by the specified model-specific C++ encapsulation interface.
----------------------	--

Alternatives To view the step method name in the Simulink Configuration Parameters graphical user interface, go to the **Interface** pane and click the **Configure C++ Encapsulation Interface** button. This button launches the Configure C++ encapsulation interface dialog box, which displays the step method name and allows you to display and configure the step method for your model class. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also “Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation
“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPClass.getStepMethodName

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

getTflArgFromString

Purpose Create TFL argument based on specified name and built-in data type

Syntax `TflArg* getTflArgFromString(hTable, name, datatype)`

Arguments

hTable
Handle to a TFL table previously returned by *hTable* = RTW.TflTable.

name
String specifying the name to use for the TFL argument, for example, 'y1'.

datatype
String specifying the built-in data type to use for the TFL argument, among the following: 'int8', 'int16', 'int32', 'uint8', 'uint16', 'uint32', 'single', 'double', or 'boolean'.

Returns Handle to the created TFL argument, which can be specified to the `addConceptualArg` function. See the example below.

Description The `getTflArgFromString` function creates a TFL argument that is based on a specified name and built-in data type.

Note The `IOType` property of the created argument defaults to 'RTW_IO_INPUT', indicating an input argument. For an output argument, you must change the `IOType` value to 'RTW_IO_OUTPUT' by directly assigning the argument property. See the example below.

Example In the following example, `getTflArgFromString` is used to create an `int16` output argument named `y1`, which is then added as a conceptual argument for a TFL table entry.

```
hLib = RTW.TflTable;  
op_entry = RTW.TflCOperationEntry;
```

```
.  
. .  
. .  
arg = hLib.getTflArgFromString('y1', 'int16');  
arg.IOType = 'RTW_IO_OUTPUT';  
op_entry.addConceptualArg( arg );
```

See Also

`addConceptualArg`

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

model_initialize

Purpose Initialization entry point in generated code for ERT-based Simulink model

Syntax
`void model_initialize(void)`
`void model_initialize(boolean_T firstTime)`

Argument *firstTime*

The Real-Time Workshop Embedded Coder software generates the *firstTime* argument to *model_initialize* only if both of the following conditions are true:

- Your selected target supports *firstTime* argument control — that is, target configuration parameter `ERTFirstTimeCompliant` is set to on. (ERT targets supplied by The MathWorks™ support *firstTime* argument control.)
- The `IncludeERTFirstTime` model configuration parameter, which is off by default, is set to on.

The *firstTime* argument specifies value 0 (FALSE) or 1 (TRUE). If *firstTime* equals 1, *model_initialize* initializes `rtModel` and other data structures private to the model. If *firstTime* equals 0, *model_initialize* resets the model's states, but does not initialize other data structures. Call *model_initialize* with *firstTime* set to 0 to reset the model's states at a time greater than start time.

Note In a future release, the Real-Time Workshop Embedded Coder software will discontinue use of the *firstTime* argument in a model's generated *model_initialize* function.

Description The *model_initialize* function contains all model initialization code. The generated code for a Simulink model calls *model_initialize* once, at the beginning of model execution. If your selected target supports

firstTime argument control and `IncludeERTFirstTime` is set to on, the generated code passes in *firstTime* as 1 (TRUE).

See Also

`model_SetEventsForThisBaseStep`, `model_step`, `model_terminate`
“Model Entry Points” in the Real-Time Workshop Embedded Coder documentation

Command Line Information for `IncludeERTFirstTime` and `ERTFirstTimeCompliant` in the Real-Time Workshop Embedded Coder reference documentation

model_SetEventsForThisBaseStep

Purpose Set event flags for multirate, multitasking operation before calling *model_step* for ERT-based Simulink model — not generated as of Version 5.1 (R2008a)

Syntax

```
void model_SetEventsForThisBaseStep(boolean_T *eventFlags)
void model_SetEventsForThisBaseStep(boolean_T *eventFlags,
RT_MODEL_model *model_M)
```

Arguments

eventFlags
Pointer to the model's event flags array.

model_M
Pointer to the real-time model object. The Real-Time Workshop Embedded Coder software generates this argument only if **Generate reusable code** is on.

Description Versions of the Real-Time Workshop Embedded Coder software prior to Version 5.1 (R2008a) generate the *model_SetEventsForThisBaseStep* function for multirate, multitasking models. The function maintains model event flags that determine which subrate tasks need to run on a given base rate time step. In a multirate, multitasking application, the program code must call *model_SetEventsForThisBaseStep* before calling the *model_step* function.

Note The macro `MODEL_SETEVENTS`, defined in the static `ert_main.c` module, provides a way to call *model_SetEventsForThisBaseStep* from a static main program.

Note Real-Time Workshop Embedded Coder no longer generates this function and you should avoid using it. The model event flags are now maintained by code in a model's generated example main program (`ert_main.c`). For more information, see “Optimizing Task Scheduling for Multirate Multitasking Models on RTOS Targets”.

See Also

`model_initialize`, `model_step`, `model_terminate`

“Model Entry Points” in the Real-Time Workshop Embedded Coder documentation

model_step

Purpose Step routine entry point in generated code for ERT-based Simulink model

Syntax

```
void model_step(void)
void model_step(int_T tid)
void model_stepN(void)
```

Arguments *tid*
Task identifier. The Real-Time Workshop Embedded Coder software generates this argument only for multirate, single-tasking models.

Calling Interfaces The *model_step* default function prototype varies depending on the number of rates in the model and the solver mode, as shown below:

Rates/Solver Mode	Function Prototype
Single-rate/SingleTasking	void <i>model_step</i> (void);
Multirate/SingleTasking	void <i>model_step</i> (int_T <i>tid</i>);
Multirate/MultiTasking (rate grouping)	void <i>model_stepN</i> (void); (<i>N</i> is a task identifier)

If you generate reusable, reentrant code for an ERT-based model using the **Generate reusable code** option, the generated code passes the model's root-level inputs and outputs, block states, parameters, and external outputs to *model_step* using a function prototype that generally resembles the following:

```
void model_step(inport_args, outport_args, BlockIO_arg,
DWork_arg, RT_model_arg);
```

The manner in which the inport and outport arguments are passed is determined by the setting of the **Pass root-level I/O as** parameter, which appears on the **Interface** pane of the Configuration Parameters dialog box only if **Generate reusable code** is selected.

For greater control over the *model_step* function prototype, you can use the **Configure Model Functions** button on the **Interface** pane to launch a Model Interface dialog box (see “Configuring Model Function Prototypes” in the Real-Time Workshop Embedded Coder documentation). Based on the **Function specification** value you specify for your *model_step* function (supported values include `Default model initialize` and `step` functions and `Model specific C` prototypes), you can preview and modify the function prototype. Once you validate and apply your changes, you can generate code based on your function prototype modifications. For more information about controlling the *model_step* function prototype, see the sections “Configuring the Target Hardware Environment” and “Controlling Generation of Function Prototypes” in the Real-Time Workshop Embedded Coder documentation.

Description

The Real-Time Workshop Embedded Coder software generates the *model_step* function for a Simulink model when the **Single output/update function** configuration option is selected (the default) in the Configuration Parameters dialog box. *model_step* contains the output and update code for all blocks in the model.

model_step is designed to be called at interrupt level from `rt_OneStep`, which is assumed to be invoked as a timer ISR. `rt_OneStep` calls *model_step* to execute processing for one clock period of the model. See “`rt_OneStep` and Scheduling Considerations” in the Real-Time Workshop Embedded Coder documentation for a description of how calls to *model_step* are generated and scheduled.

model_step

Note If the **Single output/update function** configuration option is not selected, the Real-Time Workshop Embedded Coder software generates the following model entry point functions in place of *model_step*:

- *model_output*: Contains the output code for all blocks in the model
 - *model_update*: Contain the update code for all blocks in the model
-

The *model_step* function computes the current value of all blocks. If logging is enabled, *model_step* updates logging variables. If the model's stop time is finite, *model_step* signals the end of execution when the current time equals the stop time.

In cases where a *tid* is passed in, the caller (*rt_OneStep*) assigns each task a *tid*, and *model_step* uses the *tid* argument to determine which blocks have a sample hit (and, therefore, should execute).

Under any of the following conditions, *model_step* does not check the current time against the stop time:

- The model's stop time is set to *inf*.
- Logging is disabled.
- The **Terminate function required** option is not selected.

Therefore, if any of these conditions are true, the program runs indefinitely.

See Also

model_initialize, *model_SetEventsForThisBaseStep*,
model_terminate

“Model Entry Points” in the Real-Time Workshop Embedded Coder documentation

Purpose	Termination entry point in generated code for ERT-based Simulink model
Syntax	<code>void model_terminate(void)</code>
Description	<p>The Real-Time Workshop Embedded Coder software generates the <code>model_terminate</code> function for a Simulink model when the Terminate function required configuration option is selected (the default) in the Configuration Parameters dialog box. <code>model_terminate</code> contains all model termination code and should be called as part of system shutdown.</p> <p>When <code>model_terminate</code> is called, blocks that have a terminate function execute their terminate code. If logging is enabled, <code>model_terminate</code> ends data logging.</p> <p>The <code>model_terminate</code> function should be called only once.</p> <p>If your application runs indefinitely, you do not need the <code>model_terminate</code> function. To suppress the function, clear the Terminate function required configuration option in the Configuration Parameters dialog box.</p>
See Also	<code>model_initialize</code> , <code>model_SetEventsForThisBaseStep</code> , <code>model_step</code> “Model Entry Points” in the Real-Time Workshop Embedded Coder documentation

rtw.codegenObjectives.Objective.modifyInheritedParam

Purpose Modify inherited parameter values

Syntax `modifyInheritedParam(obj, paramName, value)`

Description `modifyInheritedParam(obj, paramName, value)` changes the value of an inherited parameter that the Code Generation Advisor verifies in **Check model configuration settings against code generation objectives**. Use this method when you create a new objective from an existing objective.

Inputs

<i>obj</i>	Handle to a code generation objective object previously created.
<i>paramName</i>	Parameter that you modify in the objective.
<i>value</i>	Value of the parameter.

Examples Change the value of `InlineParameters` to off in the objective.

```
modifyInheritedParam(obj, 'InlineParams', 'off');
```

See Also `get_param`

How To

- “Creating Custom Objectives”
- “Parameter Command-Line Information Summary”

Purpose	Register objective		
Syntax	<code>register(obj)</code>		
Description	<code>register(obj)</code> registers <i>obj</i> Register and add <i>obj</i> to the end of the list of available objectives that you can use with the Code Generation Advisor.		
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a code generation objective object previously created.</td></tr></table>	<i>obj</i>	Handle to a code generation objective object previously created.
<i>obj</i>	Handle to a code generation objective object previously created.		
Examples	Register the objective: <pre>register(obj);</pre>		
See Also	<code>DASudio.CustomizationManager.ObjectiveCustomizer</code>		
How To	<ul style="list-style-type: none">• “Creating Custom Objectives”• “Registering Customizations”		

registerCFunctionEntry

Purpose Create TFL function entry based on specified parameters and register in TFL table

Syntax `TflCFunctionEntry* registerCFunctionEntry(hTable, priority, numInputs, functionName, inputType, implementationName, outputType, headerFile, genCallback, genFileName)`

Arguments

hTable
Handle to a TFL table previously returned by *hTable* = RTW.TflTable.

priority
Positive integer specifying the function entry's search priority, 0-100, relative to other entries of the same function name and conceptual argument list within this table. Highest priority is 0, and lowest priority is 100. If the table provides two implementations for a function, the implementation with the higher priority will shadow the one with the lower priority.

numInputs
Positive integer specifying the number of input arguments.

functionName
String specifying the name of the function to be replaced. The name must match one of the functions supported for replacement:

Floating-Point Math Functions			
abs	cos	log10	tan
acos	cosh	pow (Simulink)/power (Embedded MATLAB™)	tanh
asin	exp	sin	
atan	floor	sinh	

ceil	log	sqrt	
Copy Utility Function			
memcpy			
Nonfinite Support Utility Functions			
getInf	getMinusInf	getNaN	

inputType

String specifying the data type of the input arguments, for example, 'double'. (This function requires that all input arguments are of the same type.)

implementationName

String specifying the name of your implementation. For example, if *functionName* is 'sqrt', *implementationName* can be 'sqrt' or a different name of your choosing.

outputType

String specifying the data type of the return argument, for example, 'double'.

headerFile

String specifying the header file in which the implementation function is declared, for example, '<math.h>'.

genCallback

String specifying '' or 'RTW.copyFileToBuildDir'. If you specify 'RTW.copyFileToBuildDir', and if this function entry is matched and used, the function RTW.copyFileToBuildDir will be called after code generation to copy additional header, source, or object files that you have specified for this function entry to the build directory. For more information, see “Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation.

genFileName

String specifying ''. (This argument is for use only by MathWorks™ developers.)

registerCFunctionEntry

Returns Handle to the created TFL function entry.

Description The registerCFunctionEntry function provides a quick way to create and register a TFL function entry. This function can be used only if your TFL function entry meets the following conditions:

- All input arguments are of the same type.
- All input argument names and the return argument name follow the default Simulink naming convention:
 - For input argument names, u_1, u_2, \dots, u_n
 - For return argument, y_1

Example In the following example, the registerCFunctionEntry function is used to create a function entry for sqrt in a TFL table.

```
hLib = RTW.TflTable;  
  
hLib.registerCFunctionEntry(100, 1, 'sqrt', 'double', 'sqrt', ...  
                           'double', '<math.h>', '', '');
```

See Also registerCPromotableMacroEntry
“Alternative Method for Creating Function Entries” in the Real-Time Workshop Embedded Coder documentation
“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation
“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

Purpose	Create TFL promotable macro entry based on specified parameters and register in TFL table (for abs function replacement only)
Syntax	<pre>TflFunctionEntry* registerCPromotableMacroEntry(<i>hTable</i>, <i>priority</i>, <i>numInputs</i>, <i>functionName</i>, <i>inputType</i>, <i>implementationName</i>, <i>outputType</i>, <i>headerFile</i>, <i>genCallback</i>, <i>genFileName</i>)</pre>
Arguments	<p><i>hTable</i> Handle to a TFL table previously returned by <i>hTable</i> = RTW.TflTable.</p> <p><i>priority</i> Positive integer specifying the function entry's search priority, 0-100, relative to other entries of the same function name and conceptual argument list within this table. Highest priority is 0, and lowest priority is 100. If the table provides two implementations for a function, the implementation with the higher priority will shadow the one with the lower priority.</p> <p><i>numInputs</i> Positive integer specifying the number of input arguments.</p> <p><i>functionName</i> String specifying the name of the function to be replaced. Specify 'abs'. (This function should be used only for abs function replacement.)</p> <p><i>inputType</i> String specifying the data type of the input arguments, for example, 'double'. (This function requires that all input arguments are of the same type.)</p>

registerCPromotableMacroEntry

implementationName

String specifying the name of your implementation. For example, assuming *functionName* is 'abs', *implementationName* can be 'abs' or a different name of your choosing.

outputType

String specifying the data type of the return argument, for example, 'double'.

headerFile

String specifying the header file in which the implementation function is declared, for example, '<math.h>'.

genCallback

String specifying '' or 'RTW.copyFileToBuildDir'. If you specify 'RTW.copyFileToBuildDir', and if this function entry is matched and used, the function RTW.copyFileToBuildDir will be called after code generation to copy additional header, source, or object files that you have specified for this function entry to the build directory. For more information, see “Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation.

genFileName

String specifying ''. (This argument is for use only by MathWorks developers.)

Returns

Handle to the created TFL promotable macro entry.

Description

The registerCPromotableMacroEntry function creates a TFL promotable macro entry based on specified parameters and registers the entry in the TFL table. A promotable macro entry will promote the output data type based on the target word size.

This function provides a quick way to create and register a TFL promotable macro entry. This function can be used only if your TFL function entry meets the following conditions:

- All input arguments are of the same type.

- All input argument names and the return argument name follow the default Simulink naming convention:
 - For input argument names, u_1, u_2, \dots, u_n
 - For return argument, y_1

Note This function should be used only for `abs` function replacement. Other functions supported for replacement should use `registerCFunctionEntry`.

Example

In the following example, the `registerCPromotableMacroEntry` function is used to create a function entry for `abs` in a TFL table.

```
hLib = RTW.TflTable;  
  
hLib.registerCPromotableMacroEntry(100, 1, 'abs', 'double', 'abs_prime', ...  
                                   'double', '<math>prime</math>', '', '');
```

See Also

`registerCFunctionEntry`

“Alternative Method for Creating Function Entries” in the Real-Time Workshop Embedded Coder documentation

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

rtw.codegenObjectives.Objective.removeInheritedCheck

Purpose Remove inherited checks

Syntax `removeInheritedCheck(obj, checkID)`

Description `removeInheritedCheck(obj, checkID)` removes an inherited check from the objective definition. Use this method when you create a new objective from an existing objective.

When the user selects multiple objectives, if another selected objective includes this check, the Code Generation Advisor displays the check.

Inputs

<i>obj</i>	Handle to a code generation objective object previously created.
<i>checkID</i>	Unique identifier of the check that you remove from the new objective.

Examples Remove the **Disable signal logging** check from the objective.

```
removeInheritedCheck(obj, 'com.mathworks.MA.CheckDisableSignalLogging');
```

See Also `Simulink.ModelAdvisor`

How To

- “Creating Custom Objectives”
- “About IDs”

rtw.codegenObjectives.Objective.removeInheritedParam

Purpose Remove inherited parameters

Syntax `removeInheritedParam(obj, paramName)`

Description `removeInheritedParam(obj, paramName)` removes an inherited parameter from this objective. Use this method when you create a new objective from an existing objective.

When the user selects multiple objectives, if another objective includes the parameter, the Code Generation Advisor reviews the parameter value using **Check model configuration settings against code generation objectives**.

Inputs	<i>obj</i>	Handle to a code generation objective object previously created.
	<i>paramName</i>	Parameter that you want to remove from the objective.

Examples Remove Inlineparameters from the objective.

```
removeInheritedParam(obj, 'InlineParams');
```

See Also `get_param`

How To

- “Creating Custom Objectives”
- “Parameter Command-Line Information Summary”

rtIOStreamClose

Purpose Shut down communications channel with remote processor

Syntax

```
int rtIOStreamClose(  
    int streamID  
)
```

Arguments *streamID*
A handle to the stream that was returned by a previous call to `rtIOStreamOpen`.

Description

```
int rtIOStreamClose(  
    int streamID  
)
```

Call this function to shut down the communications channel and clean up any associated resources.

A return value of zero indicates success. `RTIOSTREAM_ERROR` indicates an error.

`RTIOSTREAM_ERROR` is defined in `rtiostream.h` as:

```
#define RTIOSTREAM_ERROR (-1)
```

See Also `rtIOStreamOpen`, `rtIOStreamSend`, `rtIOStreamRecv`, `rtiostream_wrapper`

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_rtiostream`

`rtwdemo_custom_pil`

Purpose	Initialize communications channel with remote processor
Syntax	<pre>int rtIOStreamOpen(int argc, void * argv[])</pre>
Arguments	<p><i>argc</i> Integer argument count, i.e., the number of parameters in argv[]</p> <p><i>argv[]</i> An array of pointers to parameters; typically these are null-terminated string parameters, however, this is allowed to be implementation dependent.</p>
Description	<pre>int rtIOStreamOpen(int argc, void * argv[])</pre> <p>This function initializes a communication stream to allow exchange of data between host and target.</p> <p>The input parameters allows driver-specific parameters to be passed to the communications driver.</p> <p>If successful, the function returns a nonnegative integer greater than zero, representing a stream handle. A return value of RTIOSTREAM_ERROR indicates an error.</p> <p>RTIOSTREAM_ERROR is defined in rtiostream.h as:</p> <pre>#define RTIOSTREAM_ERROR (-1)</pre>
See Also	rtIOStreamSend, rtIOStreamRecv, rtIOStreamClose, rtiostream_wrapper

rtIOStreamOpen

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_rtiostream`

`rtwdemo_custom_pil`

Purpose

Receive data from remote processor

Syntax

```
int rtIOStreamRecv(  
    int      streamID,  
    void    * dst,  
    size_t   size,  
    size_t  * sizeRecvd  
)
```

Arguments

streamID

A handle to the stream that was returned by a previous call to `rtIOStreamOpen`.

size

Size of data to copy into the buffer. For byte-addressable architectures, size is measured in bytes. Some DSP architectures are not byte-addressable. In these cases, size is measured in number of WORDs, where `sizeof(WORD) == 1`.

dst

A pointer to the start of the buffer where received data must be copied.

sizeRecvd

The number of units of data received and copied into the buffer *dst* (zero if no data was copied).

Description

```
int rtIOStreamRecv(  
    int      streamID,  
    void    * dst,  
    size_t   size,  
    size_t  * sizeRecvd  
)
```

This function receives data over a communication channel with a remote processor.

A return value of zero indicates success. `RTIOSTREAM_ERROR` indicates an error.

RTIOSTREAM_ERROR is defined in rtiostream.h as:

```
#define RTIOSTREAM_ERROR (-1)
```

See also rtiostreamSend for implementation and performance considerations.

See Also

rtIOStreamSend, rtIOStreamOpen, rtIOStreamClose,
rtIOStream_wrapper

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

rtwdemo_rtiostream

rtwdemo_custom_pil

Purpose Send data to remote processor

Syntax

```
int rtIOStreamSend(  
    int          streamID,  
    const void * src,  
    size_t       size,  
    size_t       * sizeSent  
)
```

Arguments

streamID

A handle to the stream that was returned by a previous call to `rtIOStreamOpen`.

src

A pointer to the start of the buffer containing an array of data to transmit

size

Size of data to transmit. For byte-addressable architectures, *size* is measured in bytes. Some DSP architectures are not byte-addressable. In these cases, *size* is measured in number of WORDs, where `sizeof(WORD) == 1`.

sizeSent

Size of data actually transmitted (always less than or equal to *size*), or zero if no data was transmitted

Description

```
int rtIOStreamSend(  
    int          streamID,  
    const void * src,  
    size_t       size,  
    size_t       * sizeSent  
)
```

This function sends data over a communication stream with a remote processor.

A return value of zero indicates success. `RTIOSTREAM_ERROR` indicates an error.

`RTIOSTREAM_ERROR` is defined in `rtiostream.h` as:

```
#define RTIOSTREAM_ERROR (-1)
```

Implementation and Performance Considerations

The API for `rtIOStream` functions is designed to be independent of the physical layer across which the data is sent. Possible physical layers include RS232, Ethernet, or Controller Area Network (CAN). The choice of physical layer affects the achievable data rates for the host-target communication.

For a processor-in-the-loop (PIL) application there is no minimum data rate requirement. However, the higher the data rate, the faster the simulation will run.

In general, a communications device driver will require additional hardware-specific or channel-specific configuration parameters. For example:

- A CAN channel may require specification of which available CAN Node should be used.
- A TCP/IP channel may require a port or static IP address to be configured.
- A CAN channel may require the CAN message ID and priority to be specified.

It is the responsibility of the user who implements the `rtIOStream` driver functions to provide this configuration data, for example by hard-coding it, or by supplying arguments to `rtIOStreamOpen`.

See Also

`rtIOStreamOpen`, `rtIOStreamClose`, `rtIOStreamRecv`,
`rtiostream_wrapper`

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_rtiostream`

`rtwdemo_custom_pil`

rtiostream_wrapper

Purpose Test rtiostream shared library methods

Syntax

```
STATION_ID = rtiostream_wrapper(SHARED_LIB, 'open')
STATION_ID = rtiostream_wrapper(SHARED_LIB, 'open', 'p1', v1,
    'p2', v2, ...)
[RES, SIZE_SENT] = rtiostream_wrapper(SHARED_LIB, 'send', ID,
    DATA, SIZE)
[RES, SIZE_RECVD] = rtiostream_wrapper(SHARED_LIB, 'recv', ID,
    SIZE)
RES = rtiostream_wrapper(SHARED_LIB, 'close', ID)
rtiostream_wrapper(SHARED_LIB, 'unloadlibrary')
```

Description rtiostream_wrapper enables you to access the methods of an rtiostream shared library from M-code, for testing purposes.

STATION_ID = rtiostream_wrapper(SHARED_LIB, 'open') opens an rtIOStream communication channel via a shared library.

If successful, STATION_ID is a handle to the channel. A value of -1 for STATION_ID indicates that the attempt to open a channel was unsuccessful. SHARED_LIB is the name of a shared library that implements the required rtIOStream functions rtIOStreamOpen, rtIOStreamSend, rtIOStreamRecv and rtIOStreamClose. The shared library must be on the system path.

STATION_ID =
rtiostream_wrapper(SHARED_LIB, 'open', 'p1', v1, 'p2', v2, ...)
opens an rtIOStream communication channel via a shared library, where p1, v1 are additional parameter value pairs. These arguments are implementation dependent, i.e., they are specific to the shared library being called.

[RES, SIZE_SENT] = rtiostream_wrapper(SHARED_LIB, 'send', ID, DATA, SIZE) sends DATA into the communication channel with handle ID, and attempts to send SIZE bytes. A value of RES==-1 indicates that an error occurred, and RES==0 indicates no error. SIZE_SENT is the number of bytes accepted by the communication channel. SIZE_SENT may be less than SIZE, i.e., the requested number of bytes to send.

[RES,SIZE_RECVD] = rtiostream_wrapper(SHARED_LIB,'recv',ID,SIZE) receives up to SIZE bytes of DATA from the communication channel with handle ID. A value of RES==-1 indicates that an error occurred, and RES==0 indicates no error. SIZE_RECVD is the number of bytes actually received from the channel. SIZE_RECVD may be less than SIZE, i.e., the requested number of bytes to send.

RES = rtiostream_wrapper(SHARED_LIB,'close',ID) closes the communication channel with handle ID.

rtiostream_wrapper(SHARED_LIB,'unloadlibrary') unloads the SHARED_LIB, clearing any persistent data.

See Also

rtIOStreamOpen, rtIOStreamSend, rtIOStreamRecv, rtIOStreamClose

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

rtwdemo_rtiostream

rtwdemo_custom_pil

RTW.AutosarInterface class

Purpose	Control and validate AUTOSAR configuration	
Description	You can use methods of the <code>RTW.AutosarInterface</code> class to configure AUTOSAR code generation and XML import and export options.	
Construction	<code>RTW.AutosarInterface</code>	Construct <code>RTW.AutosarInterface</code> object
Methods	<code>addIOConf</code>	Add AUTOSAR I/O configuration to model
	<code>attachToModel</code>	Attach <code>RTW.AutosarInterface</code> object to model
	<code>getComponentName</code>	Get XML component name
	<code>getDataTypePackageName</code>	Get XML data type package name
	<code>getDefaultConf</code>	Get default configuration
	<code>getExecutionPeriod</code>	Get runnable execution period
	<code>getImplementationName</code>	Get XML implementation name
	<code>getInitEventName</code>	Get initial event name
	<code>getInitRunnableName</code>	Get initial runnable name
	<code>getInterfacePackageName</code>	Get XML interface package name
	<code>getInternalBehaviorName</code>	Get XML internal behavior name
	<code>getIOAutosarPortName</code>	Get I/O AUTOSAR port name
	<code>getIODataAccessMode</code>	Get I/O data access mode
	<code>getIODataElement</code>	Get I/O data element name
	<code>getIOErrorStatusReceiver</code>	Get receiver port name
	<code>getIOInterfaceName</code>	Get I/O interface name

getIOPortNumber	Get I/O AUTOSAR port number
getIOServiceInterface	Get port I/O service interface
getIOServiceName	Get port I/O service name
getIOServiceOperation	Get port I/O service operation
getIsServerOperation	Determine whether server is specified
getPeriodicEventName	Get periodic event name
getPeriodicRunnableName	Get periodic runnable name
getPortDefaultConf	Get port default configuration
getServerInterfaceName	Get name of server interface
getServerOperationPrototype	Get server operation prototype
getServerPortName	Get server port name
getServerType	Determine server type
runValidation	Validate RTW.AutosarInterface object against model
setComponentName	Set XML component name
setInitEventName	Set initial event name
setInitRunnableName	Set initial runnable name
setIOAutosarPortName	Set AUTOSAR port name
setIODataAccessMode	Set I/O data access mode
setIODataElement	Set I/O data element
setIOInterfaceName	Set I/O interface name
setIsServerOperation	Indicate that server is specified
setPeriodicEventName	Set periodic event name
setPeriodicRunnableName	Set periodic runnable name
setServerInterfaceName	Set name of server interface

RTW.AutosarInterface class

setServerOperationPrototype	Specify operation prototype
setServerPortName	Set server port name
setServerType	Specify server type
syncWithModel	Synchronize configuration with model

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

See Also

“Using the Configure AUTOSAR Interface Dialog Box”, “Configuring Ports for Basic Software and Error Status Receivers”, and “Modifying and Validating an Existing AUTOSAR Interface” in the Real-Time Workshop Embedded Coder documentation

Purpose

Construct RTW.AutosarInterface object

Syntax

```
autosarInterfaceObject = AutosarInterface()  
autosarInterfaceObject = AutosarInterface(model_handle)  
autosarInterfaceObject = AutosarInterface(object_name,  
                                           model_handle)
```

Description

autosarInterfaceObject = AutosarInterface() creates an RTW.AutosarInterface object without specifying a model, and returns a handle to this object.

autosarInterfaceObject = AutosarInterface(*model_handle*) creates an RTW.AutosarInterface object with a model specified, and returns a handle to this object. The software sets the name of the RTW.AutosarInterface object to 'AutosarInterface'.

autosarInterfaceObject = AutosarInterface(*object_name*, *model_handle*) creates an RTW.AutosarInterface object with a model specified, and returns a handle to this object. The software sets the name of the RTW.AutosarInterface object to *object_name*.

Inputs

<i>model_handle</i>	Handle to Simulink model
<i>object_name</i>	Name of RTW.AutosarInterface object

Outputs

<i>autosarInterfaceObject</i>	Handle to newly created RTW.AutosarInterface object.
-------------------------------	--

See Also

“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.attachToModel

rtw.codegenObjectives.Objective Class

Purpose	Customize code generation objectives
Description	An <code>rtw.codegenObjectives.Objective</code> object creates a code generation objective.
Construction	<code>rtw.codegenObjectives.Objective</code> Create custom code generation objectives

Methods	<code>addCheck</code>	Add checks
	<code>addParam</code>	Add parameters
	<code>excludeCheck</code>	Exclude checks
	<code>modifyInheritedParam</code>	Modify inherited parameter values
	<code>register</code>	Register objective
	<code>removeInheritedCheck</code>	Remove inherited checks
	<code>removeInheritedParam</code>	Remove inherited parameters
	<code>setObjectiveName</code>	Specify objective name

Copy Semantics Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples Create a custom objective named `Reduce RAM Example`. The following code is the contents of the `sl_customization.m` file that you create.

```
function sl_customization(cm)
%SL_CUSTOMIZATION objective customization callback

objCustomizer = cm.ObjectiveCustomizer;
index = objCustomizer.addCallbackObjFcn(@addObjectives);
objCustomizer.callbackFcn{index}();
```



```
end

function addObjectives

% Create the custom objective
obj = rtw.codegenObjectives.Objective('ex_ram_1');
setObjectiveName(obj, 'Reduce RAM Example');

% Add parameters to the objective
addParam(obj, 'InlineParams', 'on');
addParam(obj, 'BooleanDataType', 'on');
addParam(obj, 'OptimizeBlockIOStorage', 'on');
addParam(obj, 'EnhancedBackFolding', 'on');
addParam(obj, 'BooleansAsBitfields', 'on');

% Add additional checks to the objective
% The Code Generation Advisor automatically includes 'Check model
% configuration settings against code generation objectives' in every
% objective.
addCheck(obj, 'Identify unconnected lines, input ports, and output ports');
addCheck(obj, 'Check model and local libraries for updates');

%Register the objective
register(obj);

end
```

See Also

DASstudio.CustomizationManager.ObjectiveCustomizer

How To

- “Creating Custom Objectives”

rtw.codegenObjectives.Objective

Purpose Create custom code generation objectives

Syntax

```
obj = rtw.codegenObjectives.Objective('objID')  
obj = rtw.codegenObjectives.Objective('objID',  
    'base_objID')
```

Description `obj = rtw.codegenObjectives.Objective('objID')` creates an objective object, `obj`.

`obj = rtw.codegenObjectives.Objective('objID', 'base_objID')` creates an object, `obj`, for a new objective that is identical to an existing objective. You can then modify the new objective to meet your requirements.

Inputs

<code>objID</code>	A permanent, unique identifier for the objective. <ul style="list-style-type: none">You must have<ul style="list-style-type: none"><code>objID</code>.The value of <code>objID</code> must remain constant.When you refresh your customizations, if <code>objID</code> is not unique, Simulink generates an error.
<code>base_objID</code>	The identifier of the objective that you want to base the new objective on.

Examples Create a new objective:

```
obj = rtw.codegenObjectives.Objective('ex_ram_1');
```

Create a new objective based on the existing Efficiency objective:

```
obj = rtw.codegenObjectives.Objective('ex_my_efficiency_1', 'Efficiency');
```

How To

- “Creating Custom Objectives”

Purpose	Configure C function prototype or C++ encapsulation interface for right-click build of specified subsystem		
Syntax	<code>RTW.configSubsystemBuild(<i>block</i>)</code>		
Description	<p><code>RTW.configSubsystemBuild(<i>block</i>)</code> opens a graphical user interface where you can configure either C function prototype information or C++ encapsulation interface information for right-click builds of a specified nonvirtual subsystem. The appropriate dialog box opens based on the Language value selected for your model on the Real-Time Workshop pane of the Configuration Parameters dialog box.</p> <p>To configure and generate C++ encapsulation interfaces for a nonvirtual subsystem, you must</p> <ul style="list-style-type: none">• Select the system target file <code>ert.tlc</code> for the model.• Select the Language parameter value C++ (Encapsulated) for the model.• Make sure that the subsystem is convertible to a Model block using the function <code>Simulink.SubSystem.convertToModelReference</code>. For referenced model conversion requirements, see the Simulink reference page <code>Simulink.SubSystem.convertToModelReference</code>.		
Inputs	<table><tr><td><i>block</i></td><td>String specifying the name of a nonvirtual subsystem block in an ERT-based Simulink model.</td></tr></table>	<i>block</i>	String specifying the name of a nonvirtual subsystem block in an ERT-based Simulink model.
<i>block</i>	String specifying the name of a nonvirtual subsystem block in an ERT-based Simulink model.		
See Also	<p>“Configuring Function Prototypes for Nonvirtual Subsystems” in the Real-Time Workshop Embedded Coder documentation</p> <p>“Controlling Generation of Function Prototypes” in the Real-Time Workshop Embedded Coder documentation</p> <p>“Configuring C++ Encapsulation Interfaces for Nonvirtual Subsystems” in the Real-Time Workshop Embedded Coder documentation</p>		

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

Purpose

Provide parameters to each target connectivity component

Syntax

```
componentArgs = rtw.connectivity.ComponentArgs (componentPath,  
        componentCodePath, componentCodeName, applicationCodePath)
```

Description

Syntax of constructor ComponentArgs:

```
componentArgs = rtw.connectivity.ComponentArgs  
(componentPath, componentCodePath, componentCodeName,  
applicationCodePath)
```

You can use the methods of this class to get information about the source component (e.g., the referenced model under test) and the target application (e.g., the PIL application).

For methods, see the following table.

Method	Syntax and Description
getComponentPath	<pre>componentPath = obj.getComponentPath</pre>
	Returns the Simulink system path of the source component (e.g., the path of the referenced model that is under test).
getComponentCodePath	<pre>componentCodePath = obj.getComponentCodePath</pre>
	Returns the Real-Time Workshop Embedded Coder code generation directory path associated with the source component (e.g., the code generation directory of the referenced model that is under test).

rtw.connectivity.ComponentArgs

Method	Syntax and Description
getComponentCodeName	<code>componentCodeName = obj.getComponentCodeName</code>
	Returns the <i>modelName.c</i> name used by Real-Time Workshop Embedded Coder during code generation of the source component.
getApplicationCodePath	<code>applicationCodePath = obj.getApplicationCodePath</code>
	Returns the directory path associated with the target application (e.g., the path associated with the PIL application).

See `rtw.connectivity.Config` for more information.

See Also

`rtw.connectivity.Config`

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

Purpose Define connectivity implementation, comprising builder, launcher, and communicator components

Syntax Config(componentArgs, builder, launcher, communicator)

Description

Constructor	Description
ComponentArgs	Wrapper for the connectivity component classes builder, launcher and communicator.

Constructor Arguments	
componentArgs	rtw.connectivity.ComponentArgs object.
builder	rtw.connectivity.Builder (e.g. rtw.connectivity.MakefileBuilder) object.
launcher	rtw.connectivity.Launcher object.
communicator	rtw.connectivity.Communicator (e.g. rtw.connectivity.-RtIOStreamHostCommunicator) object.

Constructor syntax:

Config(componentArgs, builder, launcher, communicator)

To define a connectivity implementation:

- 1 You must create a subclass of rtw.connectivity.Config that creates instances of your connectivity component classes:
 - rtw.connectivity.MakefileBuilder
 - rtw.connectivity.Launcher

- `rtw.connectivity.RtIOStreamHostCommunicator`

You can see an example `ConnectivityConfig.m`, used in the demo `rtwdemo_custom_pil`.

- 2 Define the constructor for your subclass as follows:

```
function this = MyConfig(componentArgs)
```

When Simulink creates an instance of your subclass of `rtw.connectivity.Config`, it provides an instance of the `rtw.connectivity.ComponentArgs` class as the only constructor argument. If you want to test your subclass of `rtw.connectivity.Config` manually, you may want to create an `rtw.connectivity.ComponentArgs` object to pass as a constructor argument.

- 3 After instantiating the builder, launcher and communicator objects in your subclass, call the constructor of the superclass `rtw.connectivity.Config` to define your complete target connectivity configuration, as shown in this example.

```
% call super class constructor to register components  
this@rtw.connectivity.Config(componentArgs,...  
builder, launcher, communicator);
```

You will register your subclass name (e.g. “`MyPIL.ConnectivityConfig`”) to Simulink by using the class `rtw.connectivity.ConfigRegistry`. This uses the `sl_customization.m` mechanism to register your connectivity configuration.

The PIL infrastructure instantiates your subclass as required. The `sl_customization.m` mechanism helps to ensure that a connectivity configuration is suitable for use with a particular PIL component (and its configuration set). It is also possible for the subclass to do extra validation on construction. For example, you can use the `componentPath` returned by the `getComponentPath` method of the

`componentArgs` constructor argument to query and validate parameters associated with the PIL component under test.

For supported hardware implementation settings and other support information, see “PIL Simulation Support and Limitations” in the Real-Time Workshop Embedded Coder documentation.

See Also

`rtw.connectivity.MakefileBuilder`, `rtw.connectivity.Launcher`,
`rtw.connectivity.RtIOStreamHostCommunicator`,
`rtw.connectivity.ComponentArgs`

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_custom_pil`

rtw.connectivity.ConfigRegistry

Purpose Register connectivity configuration

Syntax
`config = rtw.connectivity.ConfigRegistry`
`config = rtw.connectivity.ConfigRegistry`

Description Use this class to register your connectivity configuration with Simulink by using the `sl_customization.m` mechanism. The connectivity configuration is registered by a call to `registerTargetInfo` inside a `sl_customization.m` file.

Create or add to your `sl_customization.m` file as shown in the “Example” on page 3-178 section, and place the file on the MATLAB path. Simulink software reads the `sl_customization.m` when it starts, and registers your connectivity configuration. This step also defines the set of Simulink models that the new connectivity configuration is compatible with.

A connectivity configuration must have a unique name and be associated with a connectivity implementation class (a subclass of `rtw.connectivity.Config`). The properties of the configuration (e.g. `SystemTargetFile`) define the set of Simulink models that the connectivity implementation class is compatible with. The properties are shown in the following table.

Properties of `rtw.connectivity.ConfigRegistry`

Property Name	Description
<code>ConfigName</code>	Unique string name for this configuration
<code>ConfigClass</code>	Full class name of the connectivity implementation (e.g. <code>rtw.pil.myConnectivityConfig</code>) to register.

Properties of rtw.connectivity.ConfigRegistry (Continued)

Property Name	Description
SystemTargetFile	<p>Cell array of strings listing System Target Files that support this ConfigRegistry.</p> <p>An empty cell array matches any System Target File.</p> <p>The model's SystemTargetFileConfiguration Parameter is validated against this cell array to determine if this ConfigRegistry is valid for use.</p>
TemplateMakefile	<p>Cell array of strings listing Template Makefiles that support this ConfigRegistry. An empty cell array matches any Template Makefile and nonmakefile based targets (GenerateMakefile: off).</p> <p>The model's TemplateMakefile Configuration Parameter is validated against this cell array to determine if this ConfigRegistry is valid for use.</p>
TargetHWDeviceType	<p>Cell array of strings listing Hardware Device Types that support this ConfigRegistry.</p> <p>An empty cell array matches any Hardware Device Type.</p> <p>The model's TargetHWDeviceTypeConfiguration Parameter is validated against this cell array to determine if this ConfigRegistry is valid for use.</p>

Example

The following code shows an example `sl_customization.m` registration. You must use the `sl_customization.m` file structure shown in the example following. You must call the `registerTargetInfo` function exactly as shown.

```
function sl_customization(cm)
% SL_CUSTOMIZATION for PIL connectivity config:...
% mypil.ConnectivityConfig

% Copyright 2008 The MathWorks, Inc.
% $Revision: 1.1.4.9 $

cm.registerTargetInfo(@loc_createConfig);

% local function
function config = loc_createConfig

config = rtw.connectivity.ConfigRegistry;
config.ConfigName = 'My PIL Example';
config.ConfigClass = 'mypil.ConnectivityConfig';

% match only ert.tlc
config.SystemTargetFile = {'ert.tlc'};
% match the standard ert TMF's
config.TemplateMakefile = {'ert_default_tmf' ...
                           'ert_unix.tmf', ...
                           'ert_vc.tmf', ...
                           'ert_vcx64.tmf', ...
                           'ert_lcc.tmf'};

% match regular 32-bit machines and Custom for e.g. ...
% 64-bit Linux
config.TargetHWDeviceType = {'Generic->32-bit x86 ...
                             compatible'
                             'Generic->Custom'};
```

You must configure the file to perform the following steps when Simulink software starts:

- 1 Create an instance of the `rtw.connectivity.ConfigRegistry` class. For example,

```
config = rtw.connectivity.ConfigRegistry;
```

- 2 Assign a connectivity configuration name to the `ConfigName` property of the object. For example,

```
config.ConfigName = 'My PIL Example';
```

- 3 Associate the connectivity configuration with the connectivity API implementation (created in step 1). For example,

```
config.ConfigClass = 'mypil.ConnectivityConfig';
```

- 4 Define compatible models for this target connectivity configuration, by setting the `SystemTargetFile`, `TemplateMakefile` and `TargetHWDeviceType` properties of the object. For example,

```
% match only ert.tlc
config.SystemTargetFile = {'ert.tlc'};
% match the standard ert TMF's
config.TemplateMakefile = {'ert_default_tmf' ...
                           'ert_unix.tmf', ...
                           'ert_vc.tmf', ...
                           'ert_vcx64.tmf', ...
                           'ert_lcc.tmf'};
% match regular 32-bit machines and Custom for e.g. ...
% 64-bit Linux
config.TargetHWDeviceType = {'Generic->32-bit x86 ...
                             compatible'
                             'Generic->Custom'};
```

See Also

`rtw.connectivity.Config`

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

rtw.connectivity.ConfigRegistry

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

```
rtwdemo_custom_pil
```

Purpose Control downloading, starting and resetting executable on target hardware

Syntax Launcher(componentArgs, builder)

Description

Constructor	Syntax
Launcher	Launcher(componentArgs, builder)

The Launcher component launches an application built by a Builder object. Launcher controls the download, start and reset of the application (e.g. PIL application) associated with a rtw.connectivity.Builder object. You must make a subclass and implement the startApplication and stopApplication methods.

If necessary, you can implement a destructor method that cleans up any resources (e.g., a handle to a 3rd party download tool) when this object is cleared from memory. There is significant flexibility in how the startApplication and stopApplication methods can be implemented.

See MyPIL.Launcher for an example.

For methods, see the following table.

Method	Syntax and Description
getBuilder	builder = obj.getBuilder Returns the rtw.connectivity.Builder object associated with this Launcher object.

Method	Syntax and Description
startApplication	<p data-bbox="865 317 1163 343"><code>obj.startApplication</code></p> <p data-bbox="865 362 1251 614">Abstract method that you must implement in a subclass. Called by Simulink to start execution of the target application, created by the <code>rtw.connectivity.Builder</code> object associated with this Launcher object.</p> <p data-bbox="865 619 1285 774">The <code>startApplication</code> method must always reset the application to its initial state by ensuring that external and static (global) variables are zero initialized.</p> <p data-bbox="865 779 1222 970">Use the <code>getApplicationExecutable</code> method of the associated <code>rtw.connectivity.Builder</code> object to determine the application to start, e.g.,</p> <pre data-bbox="902 1003 1311 1060">exe = this.getBuilder.get... ApplicationExecutable</pre>

Method	Syntax and Description
stopApplication	<p><code>obj.stopApplication</code></p> <p>Abstract method that you must implemented in a subclass. Called by Simulink to stop execution of the target application, created by the <code>rtw.connectivity.Builder</code> object associated with this Launcher object.</p> <p>Use the <code>getApplicationExecutable</code> method of the associated <code>rtw.connectivity.Builder</code> object to determine the application to stop, e.g.,</p> <pre>exe = this.getBuilder.get... ApplicationExecutable</pre>
getComponentArgs	<p><code>componentArgs = obj.getComponentArgs</code></p> <p>Returns the <code>rtw.connectivity.ComponentArgs</code> object associated with this Launcher object.</p>

See Also

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_custom_pil`

rtw.connectivity.MakefileBuilder

Purpose Configure makefile-based build process

Syntax `MakefileBuilder(componentArgs, targetApplicationFramework, exeExtension)`

Description

Constructor	Description
MakefileBuilder	Control makefile-based build process.

Constructor Arguments	
componentArgs	rtw.connectivity.ComponentArgs
TargetApplicationFramework	rtw.pil.RtIOStream-ApplicationFramework (e.g. MyPIL.TargetFramework)
exeExtension	Filename extension of an executable for the target system. The extension depends on the makefile and compiler that are called by the <code>MakefileBuilder</code> . These are defined by the template makefile specified by the source component (e.g., the referenced model under test). For an embedded target the extension may be <code>'.elf'</code> , <code>'.abs'</code> , <code>'.sre'</code> , <code>'.hex'</code> , or others. For a Windows® host-based target the extension is <code>'.exe'</code> . For a UNIX® host-based target the extension is empty, <code>''</code> .

Constructor syntax:

```
MakefileBuilder(componentArgs, targetApplicationFramework,
exeExtension)
```

MakefileBuilder controls the customizable makefile-based build process supporting the creation of custom applications (e.g. a PIL application) that interface with a Simulink component such as a referenced model (represented as a collection of binary libraries).

To build the PIL application, you must provide a template makefile that includes the target `MAKEFILEBUILDER_TGT`. You can use any of the standard TMF files, e.g., `ert_unix.tmf` or `ert_vc.tmf`.

See Also

`rtw.pil.RtIOStreamApplicationFramework`,
`rtw.connectivity.ComponentArgs`

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_custom_pil`

rtw.connectivity.RtIOStreamHostCommunicator

Purpose Configure host-side communications

Syntax `RtIOStreamHostCommunicator (componentArgs, launcher, rtiostreamLib)`

Description

Constructor	Description
<code>RtIOStreamHostCommunicator</code>	Configure host-side communications with the target by loading and initializing a shared library that implements the <code>rtiostream</code> functions.

Constructor Arguments	
<code>componentArgs</code>	A <code>rtw.connectivity.ComponentArgs</code> object.
<code>launcher</code>	A <code>rtw.connectivity.Launcher</code> object.
<code>rtiostreamLib</code>	An <code>rtiostream</code> shared library that implements the host side of host-target communications.

Constructor syntax:

```
RtIOStreamHostCommunicator (componentArgs, launcher, rtiostreamLib)
```

This class configures host-side communications with the target by loading and initializing a shared library that implements the `rtiostream` functions.

Real-Time Workshop Embedded Coder provides an implementation of this shared library to support TCP/IP communications between host and target (all platforms), as well as a Windows only version for serial

communications. With TCP/IP or serial, you need only supply the target-side drivers.

For other communications protocols (e.g. USB), you must supply an appropriate shared library for the host-side of the communications link as well as the target-side drivers.

To create your instance of `rtw.connectivity.RtIOStreamHostCommunicator`, you have two options:

- Instantiate `rtw.connectivity.RtIOStreamHostCommunicator` directly, providing custom arguments to supply to the `rtiostream` shared library. This is sufficient in most cases.
- Alternatively, create a subclass of `rtw.connectivity.RtIOStreamHostCommunicator`. This may be necessary when more complex configuration is required. For example, the demo subclass `rtw.connectivity.HostTCPIPCommunicator` includes additional code to determine the TCP/IP port number on which the executable application is serving, or you could use a subclass to specify a serial port number, or specify verbose or silent operation.

Methods	
<code>setTimeoutRecvSecs</code>	Sets the timeout value for reading data.
<code>hostCommunicator.setTimeoutRecvSecs(<i>timeout</i>)</code> configures data reading to time out if no new data is received for a period of greater than <code>timeout</code> seconds.	

rtw.connectivity.RtIOStreamHostCommunicator

Methods	
setInitCommsTimeout	Sets the timeout value for initial setup of the communications channel.
<code>hostCommunicator.setInitCommsTimeout(<i>timeout</i>)</code> For some targets you may need to set a timeout value for initial setup of the communications channel. For example, the target processor may take a few seconds before it is ready to open its side of the communications channel. If you set a nonzero timeout value then the communicator repeatedly tries to open the communications channel until the timeout value is reached.	

See Also

`rtw.connectivity.ComponentArgs`, `rtw.connectivity.Launcher`,
`rtiostream_wrapper`

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_custom_pil`

RTW.getEncapsulationInterfaceSpecification

Purpose	Get handle to model-specific C++ encapsulation interface control object	
Syntax	<code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code>	
Description	<code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> returns a handle to a model-specific C++ encapsulation interface control object.	
Inputs	<code>modelName</code>	String specifying the name of a loaded ERT-based Simulink model.
Outputs	<code>obj</code>	Handle to the C++ encapsulation interface control object associated with the specified model. If the model does not have any associated C++ encapsulation interface control object, the function returns [].
Alternatives	The Configure C++ Encapsulation Interface button on the Interface pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. Once you validate and apply your changes, you can generate code based on your C++ encapsulation interface modifications. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation “Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation	

RTW.getEncapsulationInterfaceSpecification

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

Purpose	Get handle to model-specific C prototype function control object	
Syntax	<code>obj = RTW.getFunctionSpecification(modelName)</code>	
Description	<code>obj = RTW.getFunctionSpecification(modelName)</code> returns a handle to the model-specific C function prototype control object.	
Inputs	<i>modelName</i>	String specifying the name of a loaded ERT-based Simulink model.
Outputs	<i>obj</i>	Handle to the model-specific C prototype function control object associated with the specified model. If the model does not have any associated function control object, the function returns [].
Alternatives	The Configure Model Functions button on the Interface pane of the Simulink Configuration Parameters dialog box launches the Model Interface dialog box, which provides you flexible control over the C function prototypes that are generated for your model. Once you validate and apply your changes, you can generate code based on your C function prototype modifications. See “Configuring Model Function Prototypes” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code	

RTW.ModelCPPArgsClass class

Superclasses ModelCPPClass

Purpose Control C++ encapsulation interfaces for models using I/O arguments style step method

Description The ModelCPPArgsClass class provides objects that describe C++ encapsulation interfaces for models using an I/O arguments style step method. Use the `attachToModel` method to attach a C++ encapsulation interface to a loaded ERT-based Simulink model.

Construction RTW.ModelCPPArgsClass Create C++ encapsulation interface object for configuring model class with I/O arguments style step method

Methods See the methods of the base class `RTW.ModelCPPClass`, plus the following methods.

<code>getArgCategory</code>	Get argument category for Simulink model port from model-specific C++ encapsulation interface
<code>getArgName</code>	Get argument name for Simulink model port from model-specific C++ encapsulation interface
<code>getArgPosition</code>	Get argument position for Simulink model port from model-specific C++ encapsulation interface
<code>getArgQualifier</code>	Get argument type qualifier for Simulink model port from model-specific C++ encapsulation interface

runValidation	Validate model-specific C++ encapsulation interface against Simulink model
setArgCategory	Set argument category for Simulink model port in model-specific C++ encapsulation interface
setArgName	Set argument name for Simulink model port in model-specific C++ encapsulation interface
setArgPosition	Set argument position for Simulink model port in model-specific C++ encapsulation interface
setArgQualifier	Set argument type qualifier for Simulink model port in model-specific C++ encapsulation interface

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Alternatives

The **Configure C++ Encapsulation Interface** button on the **Interface** pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. Once you validate and apply your changes, you can generate code based on your C++ encapsulation interface modifications. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPArgsClass class

“Sample M-Script for Configuring the Step Method for a Model Class”
in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

Purpose	Create C++ encapsulation interface object for configuring model class with I/O arguments style step method
Syntax	<code>obj = RTW.ModelCPPArgsClass</code>
Description	<code>obj = RTW.ModelCPPArgsClass</code> returns a handle, <i>obj</i> , to a newly created object of class <code>RTW.ModelCPPArgsClass</code> .
Outputs	<i>obj</i> Handle to a newly created C++ encapsulation interface object for configuring a model class with an I/O arguments style step method. The object has not yet been configured or attached to an ERT-based Simulink model.
Alternatives	The Configure C++ Encapsulation Interface button on the Interface pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.
See Also	“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation “Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation “Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPClass class

Purpose Control C++ encapsulation interfaces for models

Description The ModelCPPClass class is the base class for the classes RTW.ModelCPPArgsClass and RTW.ModelCPPVoidClass, which provide objects that describe C++ encapsulation interfaces for models using either an I/O arguments style step method or a void-void style step method. Use the attachToModel method to attach a C++ encapsulation interface to a loaded ERT-based Simulink model.

Construction To access the methods of this class, use the constructor for either RTW.ModelCPPArgsClass or RTW.ModelCPPVoidClass.

Methods		
	attachToModel	Attach model-specific C++ encapsulation interface to loaded ERT-based Simulink model
	getClassName	Get class name from model-specific C++ encapsulation interface
	getDefaultConf	Get default configuration information for model-specific C++ encapsulation interface from Simulink model
	getNumArgs	Get number of step method arguments from model-specific C++ encapsulation interface
	getStepMethodName	Get step method name from model-specific C++ encapsulation interface

setClassName	Set class name in model-specific C++ encapsulation interface
setStepMethodName	Set step method name in model-specific C++ encapsulation interface

Alternatives

The **Configure C++ Encapsulation Interface** button on the **Interface** pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. Once you validate and apply your changes, you can generate code based on your C++ encapsulation interface modifications. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPVoidClass class

Superclasses	ModelCPPClass
Purpose	Control C++ encapsulation interfaces for models using void-void style step method
Description	The ModelCPPVoidClass class provides objects that describe C++ encapsulation interfaces for models using a void-void style step method. Use the attachToModel method to attach a C++ encapsulation interface to a loaded ERT-based Simulink model.
Construction	RTW.ModelCPPVoidClass Create C++ encapsulation interface object for configuring model class with void-void style step method
Methods	See the methods of the base class RTW.ModelCPPClass, plus the following method. runValidation Validate model-specific C++ encapsulation interface against Simulink model
Copy Semantics	Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.
Alternatives	The Configure C++ Encapsulation Interface button on the Interface pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. Once you validate and apply your changes, you can generate code based on your C++ encapsulation interface modifications. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPVoidClass

Purpose Create C++ encapsulation interface object for configuring model class with void-void style step method

Syntax `obj = RTW.ModelCPPVoidClass`

Description `obj = RTW.ModelCPPVoidClass` returns a handle, `obj`, to a newly created object of class `RTW.ModelCPPVoidClass`.

Outputs `obj` Handle to a newly created C++ encapsulation interface object for configuring a model class with a void-void style step method. The object has not yet been configured or attached to an ERT-based Simulink model.

Alternatives The **Configure C++ Encapsulation Interface** button on the **Interface** pane of the Simulink Configuration Parameters dialog box launches the Configure C++ encapsulation interface dialog box, where you can flexibly control the C++ encapsulation interfaces that are generated for your model. See “Generating and Configuring C++ Encapsulation Interfaces to Model Code” in the Real-Time Workshop Embedded Coder documentation.

See Also “Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation
“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation
“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype class

Purpose	Describe signatures of functions for model	
Description	A <code>ModelSpecificCPrototype</code> object describes the signatures of the step and initialization functions for a model. You must use this in conjunction with the <code>attachToModel</code> method.	
Construction	<code>RTW.ModelSpecificCPrototype</code>	Create model-specific C prototype object
Methods	<code>addArgConf</code>	Add argument configuration information for Simulink model port to model-specific C function prototype
	<code>attachToModel</code>	Attach model-specific C function prototype to loaded ERT-based Simulink model
	<code>getArgCategory</code>	Get argument category for Simulink model port from model-specific C function prototype
	<code>getArgName</code>	Get argument name for Simulink model port from model-specific C function prototype
	<code>getArgPosition</code>	Get argument position for Simulink model port from model-specific C function prototype
	<code>getArgQualifier</code>	Get argument type qualifier for Simulink model port from model-specific C function prototype

RTW.ModelSpecificCPrototype class

getDefaultConf	Get default configuration information for model-specific C function prototype from Simulink model
getFunctionName	Get function name from model-specific C function prototype
getNumArgs	Get number of function arguments from model-specific C function prototype
getPreview	Get model-specific C function prototype code preview
runValidation	Validate model-specific C function prototype against Simulink model
setArgCategory	Set argument category for Simulink model port in model-specific C function prototype
setArgName	Set argument name for Simulink model port in model-specific C function prototype
setArgPosition	Set argument position for Simulink model port in model-specific C function prototype
setArgQualifier	Set argument type qualifier for Simulink model port in model-specific C function prototype
setFunctionName	Set function name in model-specific C function prototype

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples

The code below creates a function control object, `a`, and uses it to add argument configuration information to the model.

```
% Open the rtdemo_counter model and specify the System Target File
rtdemo_counter
set_param(gcs,'SystemTargetFile','ert.tlc')

%% Create a function control object
a=RTW.ModelSpecificCPrototype

%% Add argument configuration information for Input and Output ports
addArgConf(a,'Input','Pointer','inputArg','const *')
addArgConf(a,'Output','Pointer','outputArg','none')

%% Attach the function control object to the model
attachToModel(a,gcs)
```

Alternatives

You can create a function control object using the Model Interface dialog box.

See Also

`RTW.ModelSpecificCPrototype.addArgConf`
“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

RTW.ModelSpecificCPrototype

Purpose Create model-specific C prototype object

Syntax `obj = RTW.ModelSpecificCPrototype`

Description `obj = RTW.ModelSpecificCPrototype` creates a handle, `obj`, to an object of class `RTW.ModelSpecificCPrototype`.

Outputs `obj` Handle to model specific C prototype object.

Examples Create a function control object, `a`, and use it to add argument configuration information to the model:

```
% Open the rtwdemo_counter model and specify the System Target File
rtwdemo_counter
set_param(gcs,'SystemTargetFile','ert.tlc')

%% Create a function control object
a=RTW.ModelSpecificCPrototype

%% Add argument configuration information for Input and Output ports
addArgConf(a,'Input','Pointer','inputArg','const *')
addArgConf(a,'Output','Pointer','outputArg','none')

%% Attach the function control object to the model
attachToModel(a,gcs)
```

Alternatives The **Configure Model Functions** button on the **Interface** pane of the Simulink Configuration Parameters dialog box launches the Model Interface dialog box, which provides you flexible control over the C function prototypes that are generated for your model. See “Configuring Model Function Prototypes” in the Real-Time Workshop Embedded Coder documentation.

See Also `RTW.ModelSpecificCPrototype.addArgConf`

“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

rtw.pil.RtIOStreamApplicationFramework

Purpose Configure target-side communications

Syntax `applicationFramework = rtw.pil.RtIOStreamApplicationFramework(
componentArgs)`

Description

Constructor	Description
RtIOStreamApplicationFramework	Specify target-specific libraries and source files that are required to build the executable.

Constructor Argument	
componentArgs	A <code>rtw.connectivity.ComponentArgs</code> object.

Constructor syntax:

```
applicationFramework =  
rtw.pil.RtIOStreamApplicationFramework(componentArgs)
```

You must create a subclass of `rtw.pil.RtIOStreamApplicationFramework`. The purpose of this class is to specify target-specific libraries and source files that are required to build the executable for the PIL application. These libraries and source files must include the device drivers that implement the target-side of the `rtiostream` communications channel. See also `rtiostream_wrapper`.

The class provides an `RTW.BuildInfo` object containing PIL-specific files (including a PIL main) that will be combined with the PIL component libraries, by the `rtw.connectivity.MakefileBuilder`, to create the PIL application. You must make a subclass and add source files, libraries, include paths and preprocessor defines that are required

to implement the `rtiostream` target communications interface to the `RTW.BuildInfo` object (access via `getBuildInfo` method).

For methods, see the following table.

Method	Syntax and Description
<code>getComponentArgs</code>	<code>componentArgs = obj.getComponentArgs</code> Returns the <code>rtw.connectivity.ComponentArgs</code> object associated with this object.
<code>getBuildInfo</code>	<code>buildInfo = obj.getBuildInfo</code> Returns the <code>RTW.BuildInfo</code> object associated with this object.
<code>addPILMain</code>	<code>obj.addPILMain(type)</code> To build the PIL application you must specify a <code>main.c</code> file. Use the <code>addPILMain</code> method to add one of the two provided files to the application framework. Use the <code>type</code> argument to specify 'target' or 'host', depending on which one of the following example PIL <code>main.c</code> files you want to use. 1) To specify a <code>main.c</code> adapted for on-target PIL and suitable for most PIL implementations, enter: <pre>obj.addPILMain('target')</pre> 2) To specify a <code>main.c</code> adapted for host-based PIL, for example, as

rtw.pil.RtIOStreamApplicationFramework

Method	Syntax and Description
	used in the mypil host example, enter: <code>obj.addPILMain('host')</code>

See Also

`rtw.connectivity.ComponentArgs`, `rtiostream_wrapper`

“Verifying Compiled Object Code with Processor-in-the-Loop Simulation” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for a Target” in the Real-Time Workshop Embedded Coder documentation.

“Build Information Object” in the Real-Time Workshop documentation.

`rtwdemo_custom_pil`

Purpose Execute CGV object

Syntax `result = cgvObj.run()`

Description `result = cgvObj.run()` executes the model once for each input data that you added to the object. `result` is a boolean value that indicates whether the run completed without execution error. `cgvObj` is a handle to a `cgv.CGV` object.

After each execution of the model, the object captures and writes the following metadata to a file in the output folder:

`ErrorDetails` — If errors occur, the error information.

`status` — The execution status.

`ver` — Version information for MathWorks products.

`hostname` — Name of computer.

`dateTime` — Date and time of execution.

`warnings` — If warnings occur, the warning messages.

`username` — Name of user.

`runtime` — The amount of time that lapsed for the execution.

Note:

- Configure the object before calling `run`. The object ignores all calls to methods after you call `run`.
- Do at least one call to `addInputData`.
- Only call `run` once for each `cgv.CGV` object.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”

RTW.AutosarInterface.runValidation

Purpose Validate RTW.AutosarInterface object against model

Syntax `[Status, Message] = autosarInterfaceObj.runValidation`

Description `[Status, Message] = autosarInterfaceObj.runValidation` runs a validation check for `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object. This check is made against the model to which `autosarInterfaceObj` is attached.

Before calling `runValidation`, you must call `attachToModel`.

The method `runValidation` performs the checks described in the following tables. The first table describes validation checks for all AUTOSAR use cases, and the second table describes specific validation checks when exporting multiple runnable entities.

Validation Checks

Group	Check
Valid names and paths	Runnable names and event names must all be unique, and must be valid AUTOSAR short name identifiers (see definition 1 following).
	AUTOSAR port, interface, and data element names must be valid AUTOSAR short name identifiers (see definition 1 following).
	AUTOSAR XML options for the component name, internal behavior name, and implementation name must be valid AUTOSAR path and short name identifiers (see definition 2 following).
	AUTOSAR XML options for the interface package name and data type package name must be valid AUTOSAR path identifiers (see definition 3 following).

Validation Checks (Continued)

Group	Check
Valid names and paths for sender/receiver ports	<p data-bbox="698 387 1317 447">For sender/receiver ports (Implicit or explicit data access mode):</p> <ul data-bbox="698 482 1326 1168" style="list-style-type: none"><li data-bbox="698 482 1326 578">• Simulink ports may have duplicated AUTOSAR port names, however the AUTOSAR Interface name must also be the same.<li data-bbox="698 595 1326 656">• A Simulink inport and an outport cannot have the same AUTOSAR port name.<li data-bbox="698 673 1326 769">• For any duplicated AUTOSAR port name and AUTOSAR Interface name, the Data element names must be unique.<li data-bbox="698 786 1326 881">• Sender/receiver ports AUTOSAR port name cannot be the same as the ServiceName of a basic software port.<li data-bbox="698 899 1326 994">• Sender/receiver ports AUTOSAR port name and Interface cannot be the same as the port name or interface of a calibration object.<li data-bbox="698 1012 1326 1168">• Sender/receiver ports Interface plus XML Option Interface package (e.g., of the form AUTOSAR/Service/servicename) cannot be the same as the ServiceInterface of a basic software port.

RTW.AutosarInterface.runValidation

Validation Checks (Continued)

Group	Check
Valid names and paths for basic software ports	<p>For basic software ports:</p> <ul style="list-style-type: none"> • ServiceName and ServiceOperation must be valid AUTOSAR short name identifiers (see definition 1 following); and ServiceInterface must be a valid AUTOSAR path identifier (see definition 3 following). • Simulink ports may have duplicated ServiceName, however the ServiceInterface must also be the same. • For any duplicated ServiceName and ServiceInterface, the ServiceOperation must be unique. • For duplicated ServiceOperation and ServiceInterface, the ServiceName must be unique. • Basic software port ServiceName name and ServiceInterface cannot be the same as the port name or interface of a calibration object.
Unsupported features	Model must not contain custom code blocks.
	Model must not contain continuous time.
	Model must not contain noninlined S-functions.
	Model must not contain nonfinite numbers.
	Model must not contain complex numbers.
	Model must not contain multitasking
	Model must not contain asynchronous rates
	Storage class of root I/O ports must be auto.
I/O must be 1D or scalar.	

Validation Checks (Continued)

Group	Check
	The sample time of a runnable must be a positive real scalar. Sample times with offset, e.g. [2 1], cause an error message.
Error status validation	An error status inport cannot point to itself (i.e., cannot specify itself as the inport for which it permits access to error status).
	Error status inports can only be defined to correspond to other inports that have Data Access Mode set to ImplicitReceive or ExplicitReceive
	Each receiver port can have only one error status port designate it as its error status.

Multiple Runnable Validation Checks

Group	Check
Wrapper subsystem validation when exporting multiple runnables. The "wrapper subsystem" is the top diagram runnables are exported from.	"Top-level" function-call subsystems (that are in the top diagram of the wrapper subsystem) must not be reusable functions. Their 'RTW System code' option must be set to 'Auto', 'Function' or 'Inline'.
	Top-level function-call subsystems cannot emit function calls.
	The only subsystems allowed at the top diagram are function-call subsystems, and empty subsystems (e.g., subsystems that contain no executable blocks, which may be used to display text in the model, or to double-click for help callback.)

Multiple Runnable Validation Checks (Continued)

Group	Check
	Top-level function-call subsystems cannot have wide trigger ports.
	A signal connected to an output of the wrapper subsystem cannot have multiple destinations. The signal must have one destination that is uniquely a sender, service, or interrunable variable.
	A signal connected to an output of the wrapper subsystem cannot have an inport of that subsystem as its source.
	All data store memory blocks referenced from subsystems must be contained in the subsystems, to prevent data integrity issues.
	All lines must be contiguous. No line in the wrapper subsystem can be an output of a virtual Bus Creator or Mux block
	Constant blocks are not allowed in the wrapper subsystem.
	No Mux, or Demux blocks are allowed in the wrapper subsystem, because the signals being passed via the runnable I/O must be contiguous and have an address at the base of the array.

Multiple Runnable Validation Checks (Continued)

Group	Check
Wrapper level Merge block validation	<p>Merge blocks have some restrictions at wrapper level:</p> <ul style="list-style-type: none">• A merge block is only allowed in the wrapper subsystem when the merge block output is connected to a diagram output (not another Merge block).• The input to a Merge block in the wrapper subsystem must be connected to a function-call subsystem output.• The input to a Merge block in the wrapper subsystem does not need a label.• A merge block in the wrapper subsystem cannot merge signals of unequal widths.• You cannot connect a Merge block in the wrapper subsystem to more than one output of any given function-call subsystem.
Other multiple runnable validation checks	<p>All runnable names, event names, and interrunnable variable names must be unique. Lines representing interrunnable variables must be labelled with valid AUTOSAR short name identifiers. No goto-from pairs are allowed because then the signal label is not unique.</p> <p>Interrunnable variables cannot be structs. All interrunnable variables must be scalar, noncomplex types. This is required by the AUTOSAR specification.</p> <p>Signal lines that connect two top-level function-call subsystems represent interrunnable variables.</p>

Multiple Runnable Validation Checks (Continued)

Group	Check
	Function-call subsystem output cannot be connected to its own input. An output of a function-call subsystem inside the wrapper subsystem cannot be connected to an input of same subsystem.
	The blocks in the top diagram of the wrapper subsystem must not have unconnected ports.
	Any top-level input that is Explicit Receive, Error Status, or Basic Software Service cannot be connected to more than one inport of any given function-call subsystem.
	The sample time of the inport associated with an error status must be the same sample time as its corresponding data port.
	Each function call subsystem being exported as a runnable entity must specify an AUTOSAR interface.

Outputs

<i>Status</i>	Status flag indicating whether the configuration is valid. If valid, <i>Status</i> is true; otherwise, it is false.
<i>Message</i>	If <i>Status</i> is false, <i>Message</i> explains why the configuration is invalid.

Definitions

The following are requirements for identifiers:

- 1 *AUTOSAR short name identifiers* must be composed of at most 32 characters, must begin with a letter, and can contain only

letters, numbers, and underscore characters. For example, `this_is_valid123`.

- 2** *AUTOSAR path and short name identifiers* must contain at least two path delimiter “/” characters, e.g., `/path/shortname`. Strings in between the path delimiters must be composed of at most 32 characters, must begin with a letter, and can contain only letters, numbers, and underscore characters.
- 3** *AUTOSAR path identifiers* must contain at least one path delimiter “/” characters, e.g., `/path`. Strings in between the path delimiters must be composed of at most 32 characters, must begin with a letter and can contain only letters, numbers, and underscore characters.

See Also

“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPArgsClass.runValidation

Purpose Validate model-specific C++ encapsulation interface against Simulink model

Syntax `[status, msg] = runValidation(obj)`

Description `[status, msg] = runValidation(obj)` runs a validation check of the specified model-specific C++ encapsulation interface against the ERT-based Simulink model to which it is attached.

Before calling this function, you must call either `attachToModel`, to attach a function prototype to a loaded model, or `RTW.getEncapsulationInterfaceSpecification`, to get the handle to a function prototype previously attached to a loaded model.

Inputs

<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
------------	---

Outputs

<i>status</i>	Boolean value; true for a valid configuration, false otherwise.
<i>msg</i>	If <i>status</i> is false, <i>msg</i> contains a string of information describing why the configuration is invalid.

Alternatives To validate a C++ encapsulation interface in the Simulink Configuration Parameters graphical user interface, go to the **Interface** pane and click the **Configure C++ Encapsulation Interface** button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. Click the **Validate** button to validate your current model step

function configuration. The **Validation** pane displays success or failure status and an explanation of any failure. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelCPPVoidClass.runValidation

Purpose Validate model-specific C++ encapsulation interface against Simulink model

Syntax `[status, msg] = runValidation(obj)`

Description `[status, msg] = runValidation(obj)` runs a validation check of the specified model-specific C++ encapsulation interface against the ERT-based Simulink model to which it is attached.

Before calling this function, you must call either `attachToModel`, to attach a function prototype to a loaded model, or `RTW.getEncapsulationInterfaceSpecification`, to get the handle to a function prototype previously attached to a loaded model.

Inputs

<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
------------	---

Outputs

<i>status</i>	Boolean value; true for a valid configuration, false otherwise.
<i>msg</i>	If <i>status</i> is false, <i>msg</i> contains a string of information describing why the configuration is invalid.

Alternatives To validate a C++ encapsulation interface in the Simulink Configuration Parameters graphical user interface, go to the **Interface** pane and click the **Configure C++ Encapsulation Interface** button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. Click the **Validate** button to validate your current model step

function configuration. The **Validation** pane displays success or failure status and an explanation of any failure. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.runValidation

Purpose Validate model-specific C function prototype against Simulink model

Syntax `[status, msg] = runValidation(obj)`

Description `[status, msg] = runValidation(obj)` runs a validation check of the specified model-specific C function prototype against the ERT-based Simulink model to which it is attached.

Before calling this function, you must call either `attachToModel`, to attach a function prototype to a loaded model, or `RTW.getFunctionSpecification`, to get the handle to a function prototype previously attached to a loaded model.

Inputs

<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .
------------	--

Outputs

<i>status</i>	True for a valid configuration; false otherwise.
<i>msg</i>	If <i>status</i> is false, <i>msg</i> contains a string explaining why the configuration is invalid.

Alternatives Click the **Validate** button in the Model Interface dialog box to run a validation check of the specified model-specific C function prototype against the ERT-based Simulink model to which it is attached. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.

See Also “Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

RTW.ModelCPPArgsClass.setArgCategory

Purpose	Set argument category for Simulink model port in model-specific C++ encapsulation interface						
Syntax	<code>setArgCategory(obj, portName, category)</code>						
Description	<code>setArgCategory(obj, portName, category)</code> sets the category — 'Value', 'Pointer', or 'Reference' — of the argument corresponding to a specified Simulink model inport or output in a specified model-specific C++ encapsulation interface.						
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code>.</td></tr><tr><td><i>portName</i></td><td>String specifying the unqualified name of an inport or output in your Simulink model.</td></tr><tr><td><i>category</i></td><td>String specifying the argument category — 'Value', 'Pointer', or 'Reference' — to be set for the specified Simulink model port.</td></tr></table>	<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code> .	<i>portName</i>	String specifying the unqualified name of an inport or output in your Simulink model.	<i>category</i>	String specifying the argument category — 'Value', 'Pointer', or 'Reference' — to be set for the specified Simulink model port.
<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code> .						
<i>portName</i>	String specifying the unqualified name of an inport or output in your Simulink model.						
<i>category</i>	String specifying the argument category — 'Value', 'Pointer', or 'Reference' — to be set for the specified Simulink model port.						

Note If you change the argument category for an output from 'Pointer' to 'Value', the change causes the argument to move to the first argument position when `attachToModel` or `runValidation` is called.

Alternatives To set argument categories in the Simulink Configuration Parameters graphical user interface, go to the **Interface** pane and click the **Configure C++ Encapsulation Interface** button. This button launches the Configure C++ encapsulation interface dialog box, where

RTW.ModelCPPArgsClass.setArgCategory

you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the **Get Default Configuration** button to display step method argument categories that you can examine and modify. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.setArgCategory

Purpose Set argument category for Simulink model port in model-specific C function prototype

Syntax `setArgCategory(obj, portName, category)`

Description `setArgCategory(obj, portName, category)` sets the category, 'Value' or 'Pointer', of the argument corresponding to a specified Simulink model inport or outport in a specified model-specific C function prototype.

Inputs	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .
	<i>portName</i>	String specifying the unqualified name of an inport or outport in your Simulink model.
	<i>category</i>	String specifying the argument category, 'Value' or 'Pointer', that you set for the specified Simulink model port.

Note If you change the argument category for an outport from 'Pointer' to 'Value', it causes the argument to move to the first argument position when you call `RTW.ModelSpecificCPrototype.attachToModel` or `RTW.ModelSpecificCPrototype.runValidation`.

Alternatives Use the **Step function arguments** table in the Model Interface dialog box to specify argument categories. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.

RTW.ModelSpecificCPrototype.setArgCategory

See Also

“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

RTW.ModelCPPArgsClass.setArgName

Purpose	Set argument name for Simulink model port in model-specific C++ encapsulation interface						
Syntax	<code>setArgName(obj, portName, argName)</code>						
Description	<code>setArgName(obj, portName, argName)</code> sets the argument name that corresponds to a specified Simulink model inport or output in a specified model-specific C++ encapsulation interface.						
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code>.</td></tr><tr><td><code>portName</code></td><td>String specifying the name of an inport or output in your Simulink model.</td></tr><tr><td><code>argName</code></td><td>String specifying the argument name to set for the specified Simulink model port. The argument must be a valid C identifier.</td></tr></table>	<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .	<code>portName</code>	String specifying the name of an inport or output in your Simulink model.	<code>argName</code>	String specifying the argument name to set for the specified Simulink model port. The argument must be a valid C identifier.
<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .						
<code>portName</code>	String specifying the name of an inport or output in your Simulink model.						
<code>argName</code>	String specifying the argument name to set for the specified Simulink model port. The argument must be a valid C identifier.						
Alternatives	To set argument names in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display step method argument names that you can examine and modify. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.						
See Also	“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation						

RTW.ModelCPPArgsClass.setArgName

“Sample M-Script for Configuring the Step Method for a Model Class”
in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.setArgName

Purpose Set argument name for Simulink model port in model-specific C function prototype

Syntax `setArgName(obj, portName, argName)`

Description `setArgName(obj, portName, argName)` sets the argument name corresponding to a specified Simulink model inport or outport in a specified model-specific C function prototype.

Inputs

<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .
<i>portName</i>	String specifying the name of an inport or outport in your Simulink model.
<i>argName</i>	String specifying the argument name to set for the specified Simulink model port. The argument must be a valid C identifier.

Alternatives Use the **Step function arguments** table in the Model Interface dialog box to specify argument names. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.

See Also “Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

RTW.ModelCPPArgsClass.setArgPosition

Purpose	Set argument position for Simulink model port in model-specific C++ encapsulation interface	
Syntax	<code>setArgPosition(<i>obj</i>, <i>portName</i>, <i>position</i>)</code>	
Description	<code>setArgPosition(<i>obj</i>, <i>portName</i>, <i>position</i>)</code> sets the position — 1 for first, 2 for second, etc. — of the argument that corresponds to a specified Simulink model inport or outport in a specified model-specific C++ encapsulation interface. The specified argument is then moved to the specified position, and other arguments shifted by one position accordingly.	
Inputs	<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(<i>modelName</i>)</code> .
	<i>portName</i>	String specifying the name of an inport or outport in your Simulink model.
	<i>position</i>	Integer specifying the argument position — 1 for first, 2 for second, etc. — to be set for the specified Simulink model port. The value must be greater than or equal to 1 and less than or equal to the number of function arguments.
Alternatives	To set argument positions in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the	

Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the **Get Default Configuration** button to display step method argument positions that you can examine and modify. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.setArgPosition

Purpose Set argument position for Simulink model port in model-specific C function prototype

Syntax `setArgPosition(obj, portName, position)`

Description `setArgPosition(obj, portName, position)` sets the position — 1 for first, 2 for second, etc. — of the argument corresponding to a specified Simulink model inport or outport in a specified model-specific C function prototype. The specified argument moves to the specified position, and other arguments shift by one position accordingly.

Inputs

<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .
<i>portName</i>	String specifying the name of an inport or outport in your Simulink model.
<i>position</i>	Integer specifying the argument position — 1 for first, 2 for second, etc. — to be set for the specified Simulink model port. The value must be greater than or equal to 1 and less than or equal to the number of function arguments.

Alternatives Use the **Step function arguments** table in the Model Interface dialog box to specify argument position. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.

See Also “Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code

Purpose	Set argument type qualifier for Simulink model port in model-specific C++ encapsulation interface						
Syntax	<code>setArgQualifier(obj, portName, qualifier)</code>						
Description	<code>setArgQualifier(obj, portName, qualifier)</code> sets the type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — of the argument that corresponds to a specified Simulink model inport or outport in a specified model-specific C++ encapsulation interface.						
Inputs	<table><tr><td><code>obj</code></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code>.</td></tr><tr><td><code>portName</code></td><td>String specifying the name of an inport or outport in your Simulink model.</td></tr><tr><td><code>qualifier</code></td><td>String specifying the argument type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — to be set for the specified Simulink model port.</td></tr></table>	<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .	<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.	<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — to be set for the specified Simulink model port.
<code>obj</code>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <code>obj = RTW.ModelCPPArgsClass</code> or <code>obj = RTW.getEncapsulationInterfaceSpecification(modelName)</code> .						
<code>portName</code>	String specifying the name of an inport or outport in your Simulink model.						
<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — to be set for the specified Simulink model port.						
Alternatives	To set argument qualifiers in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display step method argument qualifiers that you can examine and modify. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.						

RTW.ModelCPPArgsClass.setArgQualifier

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.setArgQualifier

Purpose	Set argument type qualifier for Simulink model port in model-specific C function prototype						
Syntax	<code>setArgQualifier(obj, portName, qualifier)</code>						
Description	<code>setArgQualifier(obj, portName, qualifier)</code> sets the type qualifier — 'none', 'const', 'const *', or 'const * const'— of the argument corresponding to a specified Simulink model inport or outport in a specified model-specific C function prototype.						
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code>.</td></tr><tr><td><i>portName</i></td><td>String specifying the name of an inport or outport in your Simulink model.</td></tr><tr><td><i>qualifier</i></td><td>String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const'— to be set for the specified Simulink model port.</td></tr></table>	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .	<i>portName</i>	String specifying the name of an inport or outport in your Simulink model.	<i>qualifier</i>	String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const'— to be set for the specified Simulink model port.
<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .						
<i>portName</i>	String specifying the name of an inport or outport in your Simulink model.						
<i>qualifier</i>	String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const'— to be set for the specified Simulink model port.						
Alternatives	Use the Step function arguments table in the Model Interface dialog box to specify argument qualifiers. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.						
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code						

RTW.ModelCPPClass.setClassName

Purpose	Set class name in model-specific C++ encapsulation interface	
Syntax	<code>setClassName(obj, className)</code>	
Description	<code>setClassName(obj, className)</code> sets the class name in the specified model-specific C++ encapsulation interface.	
Inputs	<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> , <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> , or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code> .
	<i>className</i>	String specifying a new name for the class described by the specified model-specific C++ encapsulation interface. The argument must be a valid C/C++ identifier.
Alternatives	To set the model class name in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display the model class name, which you can examine and modify. In the void-void step method view, you can examine and modify the model class name without having to click a button. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.	
See Also	“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation	

“Sample M-Script for Configuring the Step Method for a Model Class”
in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the
Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setComponentName

Purpose Set XML component name

Syntax `autosarInterfaceObj.setComponentName(componentName)`

Description `autosarInterfaceObj.setComponentName(componentName)` sets the XML component name of `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.

Inputs

<code>componentName</code>	XML component name for <code>autosarInterfaceObj</code>
----------------------------	---

See Also

`RTW.AutosarInterface.getComponentName`

“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

Purpose Set XML file dependencies

Syntax `importerObj.setDependencies(dependencies)`

Description `importerObj.setDependencies(dependencies)` sets the XML file dependencies associated with the `arxml.importer` object, `importerObj`.

Inputs `dependencies` Can be:

- a cell array of strings (for a list of dependencies)
- a char array (for a single dependency)
- or the empty array `[]` (for removing any dependency)

Note All atomic software components described in the XML file dependencies are ignored.

See Also “Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

arxml.importer.setFile

Purpose Set XML file name for `arxml.importer` object

Syntax `importerObj.setFile(filename)`

Description `importerObj.setFile(filename)` sets the name of the XML file associated with the `arxml.importer` object, `importerObj`.

Inputs

<code>filename</code>	XML file name. Only atomic software components described in this file can be imported.
-----------------------	--

See Also “Importing an AUTOSAR Software Component” in the Real-Time Workshop Embedded Coder documentation

RTW.ModelSpecificCPrototype.setFunctionName

Purpose	Set function name in model-specific C function prototype						
Syntax	<code>setFunctionName(obj, fcnName, fcnType)</code>						
Description	<code>setFunctionName(obj, fcnName, fcnType)</code> sets the step or initialization function name in the specified function control object.						
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code>.</td></tr><tr><td><i>fcnName</i></td><td>String specifying a new name for the function described by the function control object. The argument must be a valid C identifier.</td></tr><tr><td><i>fcnType</i></td><td>Optional. String specifying which function to name. Valid strings are 'step' and 'init'. If <i>fcnType</i> is not specified, sets the step function name.</td></tr></table>	<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .	<i>fcnName</i>	String specifying a new name for the function described by the function control object. The argument must be a valid C identifier.	<i>fcnType</i>	Optional. String specifying which function to name. Valid strings are 'step' and 'init'. If <i>fcnType</i> is not specified, sets the step function name.
<i>obj</i>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.ModelSpecificCPrototype</code> or <code>obj = RTW.getFunctionSpecification(modelName)</code> .						
<i>fcnName</i>	String specifying a new name for the function described by the function control object. The argument must be a valid C identifier.						
<i>fcnType</i>	Optional. String specifying which function to name. Valid strings are 'step' and 'init'. If <i>fcnType</i> is not specified, sets the step function name.						
Alternatives	Use the Initialize function name and Step function name fields in the Model Interface dialog box to specify function names. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.						
See Also	“Controlling Generation of Function Prototypes” — Explains how to configure model function prototypes in generated code						

RTW.AutosarInterface.setInitEventName

Purpose Set initial event name

Syntax `autosarInterfaceObj.setInitEventName(initEventName)`

Description `autosarInterfaceObj.setInitEventName(initEventName)` sets the initial event name for `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.

Inputs `initEventName` Initial event name for `autosarInterfaceObj`

See Also RTW.AutosarInterface.getInitEventName
“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setInitRunnableName

Purpose Set initial runnable name

Syntax `autosarInterfaceObj.setInitRunnableName(initRunnableName)`

Description `autosarInterfaceObj.setInitRunnableName(initRunnableName)` sets the initial runnable name for `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.

Inputs

<code><i>initRunnableName</i></code>	Initial runnable name for <code>autosarInterfaceObj</code> .
--------------------------------------	--

See Also RTW.AutosarInterface.getInitRunnableName
“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setIOAutosarPortName

Purpose Set AUTOSAR port name

Syntax `autosarInterfaceObj.setIOAutosarPortName(portName, autosarPort)`

Description `autosarInterfaceObj.setIOAutosarPortName(portName, autosarPort)` updates the AUTOSAR port name in the configuration for the specified port.

`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.

By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.

Inputs	<code>portName</code>	Name of inport/outport (string)
	<code>autosarPort</code>	AUTOSAR port name for <code>portName</code> (string).

See Also “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setIODataAccessMode

Purpose Set I/O data access mode

Syntax `autosarInterfaceObj.setIODataAccessMode(portName, dataAccessMode)`

Description `autosarInterfaceObj.setIODataAccessMode(portName, dataAccessMode)` sets the data access mode in the configuration for the specified port. `autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.

Inputs

<code>portName</code>	Name of inport/outport (string).
<code>dataAccessMode</code>	Data access mode (string). Can be one of the following: <ul style="list-style-type: none">• 'ImplicitSend'• 'ImplicitReceive'• 'ExplicitSend'• 'ExplicitReceive'

See Also `RTW.AutosarInterface.getIODataAccessMode`
“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setIODataElement

Purpose Set I/O data element

Syntax `autosarInterfaceObj.setIODataElement(portName,dataElement)`

Description `autosarInterfaceObj.setIODataElement(portName,dataElement)` updates the name of the I/O data element in the configuration for the specified port.

`autosarInterfaceObj` is a model-specific RTW.AutosarInterface object.

By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.

Inputs	<code>portName</code>	Name of the inport/outport (string).
	<code>dataElement</code>	Name of the I/O data element for <code>portName</code> (string).

See Also “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setIOInterfaceName

Purpose

Set I/O interface name

Syntax

```
autosarInterfaceObj.setIOInterfaceName(portName,  
                                         interfaceName)
```

Description

autosarInterfaceObj.setIOInterfaceName(*portName*,*interfaceName*) updates the I/O interface name in the configuration for the specified port.

autosarInterfaceObj is a model-specific RTW.AutosarInterface object.

By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.

Inputs

<i>portName</i>	Name of inport/outport (string).
<i>interfaceName</i>	Name of I/O interface for <i>portName</i> (string).

See Also

“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setIsServerOperation

Purpose	Indicate that server is specified		
Syntax	<code>autosarInterfaceObj.setIsServerOperation(isServerOperation)</code>		
Description	<p><code>autosarInterfaceObj.setIsServerOperation(isServerOperation)</code> sets the value of the property 'isServerOperation' in <code>autosarInterfaceObj</code>.</p> <p><code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.</p>		
Inputs	<table><tr><td><code>isServerOperation</code></td><td>True or false (default). If true, indicates that a server is specified in <code>autosarInterfaceObj</code>.</td></tr></table>	<code>isServerOperation</code>	True or false (default). If true, indicates that a server is specified in <code>autosarInterfaceObj</code> .
<code>isServerOperation</code>	True or false (default). If true, indicates that a server is specified in <code>autosarInterfaceObj</code> .		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

rtw.codegenObjectives.Objective.setObjectiveName

Purpose Specify objective name

Syntax `setObjectiveName(obj, objName)`

Description `setObjectiveName(obj, objName)` specifies a name for the objective. The Configuration Set Objectives dialog box displays the name of the objective.

Inputs

<i>obj</i>	Handle to a code generation objective object previously created.
<i>objName</i>	Optional string that indicates the name of the objective. If you do not specify an objective name, the Configuration Set Objectives dialog box displays the objective ID for the objective name.

Examples Name the objective Reduce RAM Example:

```
setObjectiveName(obj, 'Reduce RAM Example');
```

How To

- “Creating Custom Objectives”

cgv.CGV.setOutputDir

Purpose Specify folder

Syntax `cgvObj.setOutputDir('path')`

Description `cgvObj.setOutputDir('path')` is an optional method that specifies a location where the object writes all output and metadata files for execution. `cgvObj` is a handle to a `cgv.CGV` object. `path` is the absolute or relative path to the folder. If the path does not exist, the object attempts to create the folder. If you do not call `setOutputDir`, the object uses the current working folder.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”

Purpose Specify output data file name

Syntax `cgvObj.setOutputFile(InputIndex,OutputFile)`

Description `cgvObj.setOutputFile(InputIndex,OutputFile)` is an optional method that changes the default file name for the output data. *cgvObj* is a handle to a `cgv.CGV` object. *InputIndex* is a unique numeric identifier that specifies which output data to write to the file. The *InputIndex* is associated with specific input data. *OutputFile* is the name of the file, with or without the `.mat` extension.

How To

- “Verifying Numerical Equivalence of Results with Code Generation Verification”

RTW.AutosarInterface.setPeriodicEventName

Purpose	Set periodic event name
Syntax	<code>autosarInterfaceObj.setPeriodicEventName(<i>periodicEventName</i>)</code>
Description	<code>autosarInterfaceObj.setPeriodicEventName(<i>periodicEventName</i>)</code> sets the name of the periodic event for <code>autosarInterfaceObj</code> , a model-specific RTW.AutosarInterface object.
Inputs	<code>periodicEventName</code> Name of the periodic event for <code>autosarInterfaceObj</code> .
See Also	RTW.AutosarInterface.getPeriodicEventName “Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setPeriodicRunnableName

Purpose Set periodic runnable name

Syntax `autosarInterfaceObj.setPeriodicRunnableName(periodicRunnableName)`

Description `autosarInterfaceObj.setPeriodicRunnableName(periodicRunnableName)` sets the name of the periodic runnable for `autosarInterfaceObj`, a model-specific RTW.AutosarInterface object.

Inputs

<code><i>periodicRunnableName</i></code>	Name of periodic runnable for <code>autosarInterfaceObj</code> .
--	--

See Also RTW.AutosarInterface.getPeriodicRunnableName
“Using the Configure AUTOSAR Interface Dialog Box” in the Real-Time Workshop Embedded Coder documentation

setReservedIdentifiers

Purpose Register specified reserved identifiers to be associated with TFL table

Syntax `void setReservedIdentifiers(hTable, ids)`

Arguments

hTable

Handle to a TFL table previously returned by *hTable* = RTW.TflTable.

ids

Structure specifying reserved keywords to be registered in the TFL table. The structure must contain the following:

- **LibraryName** element, a string that specifies a TFL name: 'ANSI', 'ISO', 'GNU', or a TFL name of your choice.
- **HeaderInfos** element, a structure or cell array of structures containing
 - **HeaderName** element, a string that specifies the header file in which the identifiers are declared
 - **ReservedIds** element, a cell array of strings that specifies the names of the identifiers to be registered as reserved keywords

For example,

```
d{1}.LibraryName = 'ANSI';  
d{1}.HeaderInfos{1}.HeaderName = 'math.h';  
d{1}.HeaderInfos{1}.ReservedIds = {'y0', 'y1'};
```

Description

In a TFL table, each function implementation name defined by a table entry will be registered as a reserved identifier. You can register additional reserved identifiers for the table on a per-header-file basis. Providing additional reserved identifiers can help prevent duplicate symbols and other identifier-related compile and link issues.

The `setReservedIdentifiers` function allows you to register up to four reserved identifier structures in a TFL table. One set of reserved

identifiers can be associated with an arbitrary TFL, while the other three (if present) must be associated with ANSI^{®1}, ISO^{®2}, or GNU^{®3} libraries.

Example

In the following example, `setReservedIdentifiers` is used to register four reserved identifier structures, for 'ANSI', 'ISO', 'GNU', and 'My Custom TFL', respectively.

```
hLib = RTW.TflTable;

% Create and register TFL entries here

.
.
.

% Create and register reserved identifiers
d{1}.LibraryName = 'ANSI';
d{1}.HeaderInfos{1}.HeaderName = 'math.h';
d{1}.HeaderInfos{1}.ReservedIds = {'a', 'b'};
d{1}.HeaderInfos{2}.HeaderName = 'foo.h';
d{1}.HeaderInfos{2}.ReservedIds = {'c', 'd'};

d{2}.LibraryName = 'ISO';
d{2}.HeaderInfos{1}.HeaderName = 'math.h';
d{2}.HeaderInfos{1}.ReservedIds = {'a', 'b'};
d{2}.HeaderInfos{2}.HeaderName = 'foo.h';
d{2}.HeaderInfos{2}.ReservedIds = {'c', 'd'};

d{3}.LibraryName = 'GNU';
d{3}.HeaderInfos{1}.HeaderName = 'math.h';
d{3}.HeaderInfos{1}.ReservedIds = {'a', 'b'};
d{3}.HeaderInfos{2}.HeaderName = 'foo.h';
```

1. ANSI[®] is a registered trademark of the American National Standards Institute, Inc.
2. ISO[®] is a registered trademark of the International Organization for Standardization.
3. GNU[®] is a registered trademark of the Free Software Foundation.

setReservedIdentifiers

```
d{3}.HeaderInfos{2}.ReservedIds = {'c', 'd'};

d{4}.LibraryName = 'My Custom TFL';
d{4}.HeaderInfos{1}.HeaderName = 'my_math_lib.h';
d{4}.HeaderInfos{1}.ReservedIds = {'y1', 'u1'};
d{4}.HeaderInfos{2}.HeaderName = 'my_oper_lib.h';
d{4}.HeaderInfos{2}.ReservedIds = {'foo', 'bar'};

setReservedIdentifiers(hLib, d);
```

See Also

“Adding Target Function Library Reserved Identifiers” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

RTW.AutosarInterface.setServerInterfaceName

Purpose	Set name of server interface		
Syntax	<code>autosarInterfaceObj.setServerInterfaceName(ServerInterfaceName)</code>		
Description	<code>autosarInterfaceObj.setServerInterfaceName(ServerInterfaceName)</code> sets the name of the server interface specified in <code>autosarInterfaceObj</code> . <code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.		
Inputs	<table><tr><td><code>ServerInterfaceName</code></td><td>Server interface name for <code>autosarInterfaceObj</code>.</td></tr></table>	<code>ServerInterfaceName</code>	Server interface name for <code>autosarInterfaceObj</code> .
<code>ServerInterfaceName</code>	Server interface name for <code>autosarInterfaceObj</code> .		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

RTW.AutosarInterface.setServerOperationPrototype

Purpose	Specify operation prototype		
Syntax	<code>autosarInterfaceObj.setServerOperationPrototype(operation_prototype)</code>		
Description	<code>autosarInterfaceObj.setServerOperationPrototype(operation_prototype)</code> defines the server operation prototype for <code>autosarInterfaceObj</code> . <code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.		
Inputs	<table><tr><td><code>operation_prototype</code></td><td>String with names of prototype and arguments: <code>operation_name(dir1 datatype1 arg1, dir2 datatype2 arg2, ..., dirN datatypeN argN, ...)</code><ul style="list-style-type: none">• <code>operation_name</code> — Name of operation• <code>dirN</code> — Either IN or OUT, which indicates whether data is passed in or out of the function.• <code>datatypeN</code> — Data type, which can be an AUTOSAR basic data type or record, Simulink data type, or array.• <code>argN</code> — Name of the argumentPrototype and argument names must be valid AUTOSAR short-name identifiers.</td></tr></table>	<code>operation_prototype</code>	String with names of prototype and arguments: <code>operation_name(dir1 datatype1 arg1, dir2 datatype2 arg2, ..., dirN datatypeN argN, ...)</code> <ul style="list-style-type: none">• <code>operation_name</code> — Name of operation• <code>dirN</code> — Either IN or OUT, which indicates whether data is passed in or out of the function.• <code>datatypeN</code> — Data type, which can be an AUTOSAR basic data type or record, Simulink data type, or array.• <code>argN</code> — Name of the argument Prototype and argument names must be valid AUTOSAR short-name identifiers.
<code>operation_prototype</code>	String with names of prototype and arguments: <code>operation_name(dir1 datatype1 arg1, dir2 datatype2 arg2, ..., dirN datatypeN argN, ...)</code> <ul style="list-style-type: none">• <code>operation_name</code> — Name of operation• <code>dirN</code> — Either IN or OUT, which indicates whether data is passed in or out of the function.• <code>datatypeN</code> — Data type, which can be an AUTOSAR basic data type or record, Simulink data type, or array.• <code>argN</code> — Name of the argument Prototype and argument names must be valid AUTOSAR short-name identifiers.		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

RTW.AutosarInterface.setServerPortName

Purpose	Set server port name		
Syntax	<code>autosarInterfaceObj.setServerPortName(serverPortName)</code>		
Description	<code>autosarInterfaceObj.setServerPortName(serverPortName)</code> sets the server port name for the model-specific RTW.AutosarInterface object defined by <code>autosarInterfaceObj</code> .		
Inputs	<table><tr><td><code>serverPortName</code></td><td>Name for server port of <code>autosarInterfaceObj</code></td></tr></table>	<code>serverPortName</code>	Name for server port of <code>autosarInterfaceObj</code>
<code>serverPortName</code>	Name for server port of <code>autosarInterfaceObj</code>		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

RTW.AutosarInterface.setServerType

Purpose	Specify server type		
Syntax	<code>autosarInterfaceObj.setServerType(serverType)</code>		
Description	<p><code>autosarInterfaceObj.setServerType(serverType)</code> specifies whether the server in <code>autosarInterfaceObj</code> is application software or AUTOSAR Basic Software.</p> <p><code>autosarInterfaceObj</code> is a model-specific RTW.AutosarInterface object.</p>		
Inputs	<table><tr><td><code>serverType</code></td><td>Either 'Application software' or 'Basic software'</td></tr></table>	<code>serverType</code>	Either 'Application software' or 'Basic software'
<code>serverType</code>	Either 'Application software' or 'Basic software'		
See Also	“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation		

RTW.ModelCPPClass.setStepMethodName

Purpose	Set step method name in model-specific C++ encapsulation interface				
Syntax	<code>setStepMethodName(<i>obj</i>, <i>fcnName</i>)</code>				
Description	<code>setStepMethodName(<i>obj</i>, <i>fcnName</i>)</code> sets the step method name in the specified model-specific C++ encapsulation interface.				
Inputs	<table><tr><td><i>obj</i></td><td>Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code>, <i>obj</i> = <code>RTW.ModelCPPVoidClass</code>, or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(<i>modelName</i>)</code>.</td></tr><tr><td><i>fcnName</i></td><td>String specifying a new name for the step method described by the specified model-specific C++ encapsulation interface. The argument must be a valid C/C++ identifier.</td></tr></table>	<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> , <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> , or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(<i>modelName</i>)</code> .	<i>fcnName</i>	String specifying a new name for the step method described by the specified model-specific C++ encapsulation interface. The argument must be a valid C/C++ identifier.
<i>obj</i>	Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by <i>obj</i> = <code>RTW.ModelCPPArgsClass</code> , <i>obj</i> = <code>RTW.ModelCPPVoidClass</code> , or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(<i>modelName</i>)</code> .				
<i>fcnName</i>	String specifying a new name for the step method described by the specified model-specific C++ encapsulation interface. The argument must be a valid C/C++ identifier.				
Alternatives	To set the step method name in the Simulink Configuration Parameters graphical user interface, go to the Interface pane and click the Configure C++ Encapsulation Interface button. This button launches the Configure C++ encapsulation interface dialog box, where you can display and configure the step method for your model class. In the I/O arguments step method view of this dialog box, click the Get Default Configuration button to display the step method name, which you can examine and modify. In the void-void step method view, you can examine and modify the step method name without having to click a button. For more information, see “Configuring the Step Method for Your Model Class” in the Real-Time Workshop Embedded Coder documentation.				

RTW.ModelCPPClass.setStepMethodName

See Also

“Configuring C++ Encapsulation Interfaces Programmatically” in the Real-Time Workshop Embedded Coder documentation

“Sample M-Script for Configuring the Step Method for a Model Class” in the Real-Time Workshop Embedded Coder documentation

“Controlling Generation of Encapsulated C++ Model Interfaces” in the Real-Time Workshop Embedded Coder documentation

setTf1CFunctionEntryParameters

- Purpose** Set specified parameters for function entry in TFL table
- Syntax** `void setTf1CFunctionEntryParameters(hEntry, varargin)`
- Arguments**
- hEntry*
Handle to a TFL table entry previously returned by *hEntry* = RTW.Tf1CFunctionEntry.
 - varargin*
Parameter/value pairs for the function entry. See varargin Parameters.

varargin Parameters The following function entry parameters can be specified to the setTf1CFunctionEntryParameters function using parameter/value argument pairs. For example,

```
setTf1CFunctionEntryParameters(..., 'Key', 'sqrt', ...);
```

Key

String specifying the name of the function to be replaced. The name must match one of the functions supported for replacement:

Floating-Point Math Functions			
abs	cos	log10	tan
acos	cosh	pow (Simulink)/power (Embedded MATLAB)	tanh
asin	exp	sin	
atan	floor	sinh	
ceil	log	sqrt	
Copy Utility Function			
memcpy			

setTflCFunctionEntryParameters

Nonfinite Support Utility Functions			
getInf	getMinusInf	getNaN	

GenCallback

String specifying '' or 'RTW.copyFileToBuildDir'. The default is ''. If you specify 'RTW.copyFileToBuildDir', and if this function entry is matched and used, the function RTW.copyFileToBuildDir will be called after code generation to copy additional header, source, or object files that you have specified for this function entry to the build directory. For more information, see “Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation.

Priority

Positive integer specifying the function entry’s search priority, 0-100, relative to other entries of the same function name and conceptual argument list within this table. Highest priority is 0, and lowest priority is 100. The default is 100. If the table provides two implementations for a function, the implementation with the higher priority will shadow the one with the lower priority.

ImplType

Specifies the type of entry: FCN_IMPL_FUNCT for function or FCN_IMPL_MACRO for macro. The default is FCN_IMPL_FUNCT.

ImplementationName

String specifying the name of the implementation function, for example, 'sqrt', which can match or differ from the Key name. The default is ''.

ImplementationHeaderFile

String specifying the name of the header file that declares the implementation function, for example, '<math.h>'. The default is ''.

ImplementationHeaderPath

String specifying the full path to the implementation header file. The default is ''.

ImplementationSourceFile

String specifying the name of the implementation source file. The default is ''.

ImplementationSourcePath

String specifying the full path to the implementation source file. The default is ''.

AcceptExprInput

Boolean value used to flag the code generator that the implementation function described by this entry should accept expression inputs. The default value is true if ImplType equals FCN_IMPL_FUNCT and false if ImplType equals FCN_IMPL_MACRO.

If the value is true, expression inputs are integrated into the generated code in a form similar to the following:

```
rtY.Out1 = mySin(rtU.In1 + rtU.In2);
```

If the value is false, a temporary variable is generated for the expression input, as follows:

```
real_T rtb_Sum;  
  
rtb_Sum = rtU.In1 + rtU.In2;  
rtY.Out1 = mySin(rtb_Sum);
```

SideEffects

Boolean value used to flag the code generator that the implementation function described by this entry should not be optimized away. This parameter applies to implementation functions that return void but should not be optimized away, such as a memcpy implementation or an implementation function that accesses global memory values. For those implementation functions only, you must include this parameter and specify the value true. The default is false.

Description

The setTflCFunctionEntryParameters function sets specified parameters for a function entry in a TFL table.

setTf1CFunctionEntryParameters

Example

In the following example, the `setTf1CFunctionEntryParameters` function is used to set specified parameters for a TFL function entry for `sqrt`.

```
fcn_entry = RTW.Tf1CFunctionEntry;
fcn_entry.setTf1CFunctionEntryParameters( ...
                                     'Key',           'sqrt', ...
                                     'Priority',       100, ...
                                     'ImplementationName', 'sqrt', ...
                                     'ImplementationHeaderFile', '<math.h>' );
```

See Also

“Example: Mapping Math Functions to Target-Specific Implementations” in the Real-Time Workshop Embedded Coder documentation

“Creating Function Replacement Tables” in the Real-Time Workshop Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

setTf1COperationEntryParameters

Purpose Set specified parameters for operator entry in TFL table

Syntax `void setTf1COperationEntryParameters(hEntry, varargin)`

Arguments *hEntry*
Handle to a TFL table entry previously returned by one of the following class instantiations:

`hEntry = RTW.Tf1COperationEntry;` Supports operator replacement, described in “Example: Mapping Scalar Operators to Target-Specific Implementations” and “Mapping Nonscalar Operators to Target-Specific Implementations”

`hEntry = RTW.Tf1COperationEntry-Generator;` Provides relative scaling factor (RSF) fixed-point parameters, described in “Mapping Fixed-Point Operators to Target-Specific Implementations”, that are not available in `RTW.Tf1COperationEntry`

`hEntry = RTW.Tf1COperationEntry-Generator_NetSlope;` Provides net slope parameters, described in “Mapping Fixed-Point Operators to Target-Specific Implementations”, that are not available in `RTW.Tf1COperationEntry`

`hEntry = RTW.Tf1BlasEntry-Generator;` Supports replacement of nonscalar operators with MathWorks BLAS functions, described in “Mapping Nonscalar Operators to Target-Specific Implementations”

`hEntry = RTW.Tf1CBlasEntry-Generator;` Supports replacement of nonscalar operators with ANSI/ISO C BLAS functions, described in “Mapping Nonscalar Operators to Target-Specific Implementations”

setTf1COperationEntryParameters

Note If you want to specify any of the parameters `SlopesMustBeTheSame`, `MustHaveZeroNetBias`, `RelativeScalingFactorF`, or `RelativeScalingFactorE` for your operator entry, instantiate your table entry using `hEntry = RTW.Tf1COperationEntryGenerator` rather than `hEntry = RTW.Tf1COperationEntry`. If you want to use `NetSlopeAdjustmentFactor` and `NetFixedExponent`, instantiate your table entry using `hEntry = RTW.Tf1COperationEntryGenerator_NetSlope`.

varargin

Parameter/value pairs for the operator entry. See `varargin` Parameters.

varargin Parameters

The following operator entry parameters can be specified to the `setTf1COperationEntryParameters` function using parameter/value argument pairs. For example,

```
setTf1COperationEntryParameters(..., 'Key', 'RTW_OP_ADD', ...);
```

Key

String specifying the operator to be replaced, among the operators supported for replacement:

- 'RTW_OP_ADD' for + (addition)
- 'RTW_OP_MINUS' for - (subtraction)
- 'RTW_OP_MUL' for * (multiplication)
- 'RTW_OP_DIV' for / (division)
- 'RTW_OP_TRANS' for matrix or array transposition
- 'RTW_OP_CAST' for data type conversion (cast).
- 'RTW_OP_SL' for << (shift left)

The default is 'RTW_OP_ADD'.

GenCallback

String specifying '' or 'RTW.copyFileToBuildDir'. The default is ''. If you specify 'RTW.copyFileToBuildDir', and if this operator entry is matched and used, the function RTW.copyFileToBuildDir will be called after code generation to copy additional header, source, or object files that you have specified for this operator entry to the build directory. For more information, see “Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation.

Priority

Positive integer specifying the operator entry’s search priority, 0-100, relative to other entries of the same operator name and conceptual argument list within this table. Highest priority is 0, and lowest priority is 100. The default is 100. If the table provides two implementations for an operator, the implementation with the higher priority will shadow the one with the lower priority.

RoundingMode

String specifying the rounding mode supported by the implementation function: 'RTW_ROUND_FLOOR', 'RTW_ROUND_CEILING', 'RTW_ROUND_ZERO', 'RTW_ROUND_NEAREST', 'RTW_ROUND_NEAREST_ML', 'RTW_ROUND_SIMPLEST', 'RTW_ROUND_CONV', or 'RTW_ROUND_UNSPECIFIED'. The default is 'RTW_ROUND_UNSPECIFIED'.

SaturationMode

String specifying the saturation mode supported by the implementation function: 'RTW_SATURATE_ON_OVERFLOW', 'RTW_WRAP_ON_OVERFLOW', or 'RTW_SATURATE_UNSPECIFIED'. The default is 'RTW_SATURATE_UNSPECIFIED'.

SlopesMustBeTheSame

Boolean flag that, when set to true, indicates that TFL replacement request processing must check that the slopes on all arguments (input and output) are equal. The default is false.

setTflCOperationEntryParameters

This parameter and `MustHaveZeroNetBias` can be used for fixed-point addition and subtraction replacement. Set both parameters to `true` to disregard specific slope and bias values and map relative slope and bias values to a replacement function.

To use this parameter, you must instantiate your table entry using `hEntry = RTW.TflCOperationEntryGenerator` rather than `hEntry = RTW.TflCOperationEntry`.

`MustHaveZeroNetBias`

Boolean flag that, when set to `true`, indicates that TFL replacement request processing must check that the net bias on all arguments is zero. The default is `false`.

This parameter and `SlopesMustBeTheSame` can be used for fixed-point addition and subtraction replacement. Set both parameters to `true` to disregard specific slope and bias values and map relative slope and bias values to a replacement function.

To use this parameter, you must instantiate your table entry using `hEntry = RTW.TflCOperationEntryGenerator` rather than `hEntry = RTW.TflCOperationEntry`.

`RelativeScalingFactorF`

Floating-point value specifying the slope adjustment factor (F) part of the relative scaling factor, $F2^E$, for relative scaling TFL entries. The default is `1.0`.

This parameter and `RelativeScalingFactorE` can be used for fixed-point multiplication and division replacement. Specify both parameters to map a range of slope and bias values to a replacement function.

To use this parameter, you must instantiate your table entry using `hEntry = RTW.TflCOperationEntryGenerator` rather than `hEntry = RTW.TflCOperationEntry`.

RelativeScalingFactorE

Floating-point value specifying the fixed exponent (E) part of the relative scaling factor, $F2^E$, for relative scaling TFL entries. For example, -3.0. The default is 0.

This parameter and `RelativeScalingFactorF` can be used for fixed-point multiplication and division replacement. Specify both parameters to map a range of slope and bias values to a replacement function.

To use this parameter, you must instantiate your table entry using `hEntry = RTW.Tf1COperationEntryGenerator` rather than `hEntry = RTW.Tf1COperationEntry`.

isRSF

Boolean value specifying that the operator entry is a relative scaling factor (RSF) entry. Specify `true` if the values of `RelativeScalingFactorF` and `RelativeScalingFactorE` equal their defaults, 1.0 and 0, but the entry nonetheless should be interpreted by the code generation process as an RSF entry.

NetSlopeAdjustmentFactor

Floating-point value specifying the slope adjustment factor (F) part of the net slope, $F2^E$, for net slope TFL entries. The default is 1.0.

This parameter and `NetFixedExponent` can be used for fixed-point multiplication and division replacement. Specify both parameters to map a range of slope and bias values to a replacement function.

To use this parameter, you must instantiate your table entry using `hEntry = RTW.Tf1COperationEntryGenerator_NetSlope` rather than `hEntry = RTW.Tf1COperationEntry`.

NetFixedExponent

Floating-point value specifying the fixed exponent (E) part of the net slope, $F2^E$, for net slope TFL entries. For example, -3.0. The default is 0.

setTf1COperationEntryParameters

This parameter and `NetSlopeAdjustmentFactor` can be used for fixed-point multiplication and division replacement. Specify both parameters to map a range of slope and bias values to a replacement function.

To use this parameter, you must instantiate your table entry using `hEntry = RTW.Tf1COperationEntryGenerator_NetSlope` rather than `hEntry = RTW.Tf1COperationEntry`.

ImplementationName

String specifying the name of the implementation function, for example, `'s8_add_s8_s8'`. The default is `''`.

ImplementationHeaderFile

String specifying the name of the header file that declares the implementation function, for example, `'s8_add_s8_s8.h'`. The default is `''`.

ImplementationHeaderPath

String specifying the full path to the implementation header file. The default is `''`.

ImplementationSourceFile

String specifying the name of the implementation source file, for example, `'s8_add_s8_s8.c'`. The default is `''`.

ImplementationSourcePath

String specifying the full path to the implementation source file. The default is `''`.

AcceptExprInput

Boolean value used to flag the code generator that the implementation function described by this entry should accept expression inputs. If the value is `true` (the default), expression inputs are integrated into the generated code in a form similar to the following:

```
rtY.Out1 = u8_add_u8_u8(u8_add_u8_u8(rtU.In1, rtU.In2), rtU.In3);
```

setTf1COperationEntryParameters

If the value is `false`, a temporary variable is generated for the expression input, as follows:

```
uint8_T tempVar;  
  
tempVar = u8_add_u8_u8(rtU.In1, rtU.In2);  
rtY.Out1 = u8_add_u8_u8(tempVar, rtU.In3);
```

SideEffects

Boolean value used to flag the code generator that the implementation function described by this entry should not be optimized away. This parameter applies to implementation functions that return `void` but should not be optimized away, such as an implementation function that accesses global memory values. For those implementation functions only, you must include this parameter and specify the value `true`. The default is `false`.

Description

The `setTf1COperationEntryParameters` function sets specified parameters for an operator entry in a TFL table.

Example

In the following example, the `setTf1COperationEntryParameters` function is used to set parameters for a TFL operator entry for `uint8` addition.

```
op_entry = RTW.Tf1COperationEntry;  
op_entry.setTf1COperationEntryParameters( ...  
    'Key', 'RTW_OP_ADD', ...  
    'Priority', 90, ...  
    'SaturationMode', 'RTW_SATURATE_UNSPECIFIED', ...  
    'RoundingMode', 'RTW_ROUND_UNSPECIFIED', ...  
    'ImplementationName', 'u8_add_u8_u8', ...  
    'ImplementationHeaderFile', 'u8_add_u8_u8.h', ...  
    'ImplementationSourceFile', 'u8_add_u8_u8.c' );
```

In the following example, the `setTf1COperationEntryParameters` function is used to set parameters for a TFL operator entry for

setTf1COperationEntryParameters

fixed-point int16 division. The table entry specifies a relative scaling between the operator inputs and output in order to map a range of slope and bias values to a replacement function.

```
op_entry = RTW.Tf1COperationEntryGenerator;
op_entry.setTf1COperationEntryParameters( ...
    'Key',                'RTW_OP_DIV', ...
    'Priority',           90, ...
    'SaturationMode',    'RTW_WRAP_ON_OVERFLOW', ...
    'RoundingMode',      'RTW_ROUND_CEILING', ...
    'RelativeScalingFactorF', 1.0, ...
    'RelativeScalingFactorE', -3.0, ...
    'ImplementationName', 's16_div_s16_s16_rsfOp125', ...
    'ImplementationHeaderFile', 's16_div_s16_s16_rsfOp125.h', ...
    'ImplementationSourceFile', 's16_div_s16_s16_rsfOp125.c' );
```

In the following example, the `setTf1COperationEntryParameters` function is used to set parameters for a TFL operator entry for fixed-point uint16 addition. The table entry specifies equal slope and zero net bias across operator inputs and output in order to map relative slope and bias values (rather than a specific slope and bias combination) to a replacement function.

```
op_entry = RTW.Tf1COperationEntryGenerator;
op_entry.setTf1COperationEntryParameters( ...
    'Key',                'RTW_OP_ADD', ...
    'Priority',           90, ...
    'SaturationMode',    'RTW_WRAP_ON_OVERFLOW', ...
    'RoundingMode',      'RTW_ROUND_UNSPECIFIED', ...
    'SlopesMustBeTheSame', true, ...
    'MustHaveZeroNetBias', true, ...
    'ImplementationName', 'u16_add_SameSlopeZeroBias', ...
    'ImplementationHeaderFile', 'u16_add_SameSlopeZeroBias.h', ...
    'ImplementationSourceFile', 'u16_add_SameSlopeZeroBias.c' );
```

See Also

“Example: Mapping Scalar Operators to Target-Specific Implementations” in the Real-Time Workshop Embedded Coder documentation

“Mapping Fixed-Point Operators to Target-Specific Implementations”
in the Real-Time Workshop Embedded Coder documentation

“Creating Function Replacement Tables” in the Real-Time Workshop
Embedded Coder documentation

“Replacing Math Functions and Operators Using Target Function
Libraries” in the Real-Time Workshop Embedded Coder documentation

slConfigUIGetVal

Purpose	Return current value for custom target configuration option
Syntax	<code>value = slConfigUIGetVal(hDlg, hSrc, 'OptionName')</code>
Arguments	<p><code>hDlg</code> Handle created in the context of a <code>SelectCallback</code> function and used by the System Target File Callback Interface functions. Pass this variable but do not set it or use it for any other purpose.</p> <p><code>hSrc</code> Handle created in the context of a <code>SelectCallback</code> function and used by the System Target File Callback Interface functions. Pass this variable but do not set it or use it for any other purpose.</p> <p><code>'OptionName'</code> Quoted name of the TLC variable defined for a custom target configuration option.</p>
Returns	Current value of the specified option. The data type of the return value depends on the data type of the option.
Description	The <code>slConfigUIGetVal</code> function is used in the context of a user-written <code>SelectCallback</code> function, which is triggered when the custom target is selected in the System Target File Browser in the Configuration Parameters dialog box. You use <code>slConfigUIGetVal</code> to read the current value of a specified target option.
Example	In the following example, the <code>slConfigUIGetVal</code> function returns the value of the Terminate function required option on the Real-Time Workshop/Interface pane of the Configuration Parameters dialog box.

```
function usertarget_selectcallback(hDlg, hSrc)

    disp(['*** Select callback triggered:', sprintf('\n'), ...
        ' Uncheck and disable "Terminate function required."]);

    disp(['Value of IncludeMdlTerminateFcn was ', ...
```

```
slConfigUIGetVal(hDlg, hSrc, 'IncludeMdlTerminateFcn']));  
  
slConfigUISetVal(hDlg, hSrc, 'IncludeMdlTerminateFcn', 'off');  
slConfigUISetEnabled(hDlg, hSrc, 'IncludeMdlTerminateFcn', false);
```

See Also

slConfigUISetEnabled, slConfigUISetVal

“Defining and Displaying Custom Target Options” in the Real-Time Workshop Embedded Coder documentation

“Parameter Command-Line Information Summary” in the Real-Time Workshop documentation

sIConfigUISetEnabled

Purpose	Enable or disable custom target configuration option
Syntax	<pre>sIConfigUISetEnabled(hDlg, hSrc, 'OptionName', true) sIConfigUISetEnabled(hDlg, hSrc, 'OptionName', false)</pre>
Arguments	<p>hDlg Handle created in the context of a <code>SelectCallback</code> function and used by the System Target File Callback Interface functions. Pass this variable but do not set it or use it for any other purpose.</p> <p>hSrc Handle created in the context of a <code>SelectCallback</code> function and used by the System Target File Callback Interface functions. Pass this variable but do not set it or use it for any other purpose.</p> <p>'OptionName' Quoted name of the TLC variable defined for a custom target configuration option.</p> <p>true Specifies that the option should be enabled.</p> <p>false Specifies that the option should be disabled.</p>

Description The `sIConfigUISetEnabled` function is used in the context of a user-written `SelectCallback` function, which is triggered when the custom target is selected in the System Target File Browser in the Configuration Parameters dialog box. You use `sIConfigUISetEnabled` to enable or disable a specified target option.

Example In the following example, the `sIConfigUISetEnabled` function disables the **Terminate function required** option on the **Real-Time Workshop/Interface** pane of the Configuration Parameters dialog box.

```
function usertarget_selectcallback(hDlg, hSrc)

    disp(['*** Select callback triggered:', sprintf('\n'), ...
```



```
        ' Uncheck and disable "Terminate function required".');  
  
disp(['Value of IncludeMdlTerminateFcn was ', ...  
     slConfigUIGetVal(hDlg, hSrc, 'IncludeMdlTerminateFcn')]);  
  
slConfigUISetVal(hDlg, hSrc, 'IncludeMdlTerminateFcn', 'off');  
slConfigUISetEnabled(hDlg, hSrc, 'IncludeMdlTerminateFcn', false);
```

See Also

slConfigUIGetVal, slConfigUISetVal

“Defining and Displaying Custom Target Options” in the Real-Time Workshop Embedded Coder documentation

“Parameter Command-Line Information Summary” in the Real-Time Workshop documentation

slConfigUISetVal

Purpose	Set value for custom target configuration option
Syntax	<code>slConfigUISetVal(hDlg, hSrc, 'OptionName', OptionValue)</code>
Arguments	<p>hDlg Handle created in the context of a <code>SelectCallback</code> function and used by the System Target File Callback Interface functions. Pass this variable but do not set it or use it for any other purpose.</p> <p>hSrc Handle created in the context of a <code>SelectCallback</code> function and used by the System Target File Callback Interface functions. Pass this variable but do not set it or use it for any other purpose.</p> <p>'OptionName' Quoted name of the TLC variable defined for a custom target configuration option.</p> <p>OptionValue Value to be set for the specified option.</p>

Description The `slConfigUISetVal` function is used in the context of a user-written `SelectCallback` function, which is triggered when the custom target is selected in the System Target File Browser in the Configuration Parameters dialog box. You use `slConfigUISetVal` to set the value of a specified target option.

Example In the following example, the `slConfigUISetVal` function sets the value 'off' for the **Terminate function required** option on the **Real-Time Workshop/Interface** pane of the Configuration Parameters dialog box.

```
function usertarget_selectcallback(hDlg, hSrc)

    disp(['*** Select callback triggered:', sprintf('\n'), ...
        '  Uncheck and disable "Terminate function required".']);

    disp(['Value of IncludeMdlTerminateFcn was ', ...
        slConfigUIGetVal(hDlg, hSrc, 'IncludeMdlTerminateFcn')]);
```

```
slConfigUISetVal(hDlg, hSrc, 'IncludeMdlTerminateFcn', 'off');  
slConfigUISetEnabled(hDlg, hSrc, 'IncludeMdlTerminateFcn', false);
```

See Also

slConfigUIGetVal, slConfigUISetEnabled

“Defining and Displaying Custom Target Options” in the Real-Time Workshop Embedded Coder documentation

“Parameter Command-Line Information Summary” in the Real-Time Workshop documentation

RTW.AutosarInterface.syncWithModel

Purpose	Synchronize configuration with model
Syntax	<code>autosarInterfaceObj.syncWithModel</code>
Description	<p><code>autosarInterfaceObj.syncWithModel</code> synchronizes the configuration with the model for the <code>RTW.AutosarInterface</code> class.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p>
See Also	“Generating Code That Complies with AUTOSAR Standards” in the Real-Time Workshop Embedded Coder documentation

Block Reference

AUTOSAR Client-Server
Communication (p. 4-2)

Configuration Wizards (p. 4-3)

Module Packaging (p. 4-4)

Invoke AUTOSAR server operation

Automatically update configuration
of parent Simulink model

Create potential Simulink data
objects

AUTOSAR Client-Server Communication

Invoke AUTOSAR Server Operation	Configure AUTOSAR client port to access Basic Software or application software components
Mode Switch for Invoke AUTOSAR Server Operation	Toggle AUTOSAR client-server operation subsystem blocks between simulation and code generation mode

Configuration Wizards

Custom M-file	Automatically update active configuration parameters of parent model using custom M-file
ERT (optimized for fixed-point)	Automatically update active configuration parameters of parent model for ERT fixed-point code generation
ERT (optimized for floating-point)	Automatically update active configuration parameters of parent model for ERT floating-point code generation
GRT (debug for fixed/floating-point)	Automatically update active configuration parameters of parent model for GRT fixed- or floating-point code generation with debugging enabled
GRT (optimized for fixed/floating-point)	Automatically update active configuration parameters of parent model for GRT fixed- or floating-point code generation

Module Packaging

Data Object Wizard

Simulink data object wizard for creating potential Simulink data objects

Blocks — Alphabetical List

Custom M-file

Purpose

Automatically update active configuration parameters of parent model using custom M-file

Library

Configuration Wizards

Description



When you add a Custom M-file block to your Simulink model and double-click it, a custom M-file script executes and automatically configures model parameters that are relevant to code generation. You can also set a block option to invoke the build process after configuring the model.

After double-clicking the block, you can verify that the model parameter values have changed by opening the Configuration Parameters dialog box and examining the settings.

The MathWorks provides an example M-file script, `matlabroot/toolbox/rtw/rtw/rtwsampleconfig.m`, that you can use with the Custom M-file block and adapt to your model requirements. The block and the script provide a starting point for customization. For more information, see “Creating a Custom Configuration Wizard Block” in the Real-Time Workshop Embedded Coder documentation.

Note You can include more than one Configuration Wizard block in your model. This provides a quick way to switch between configurations.

Parameters

Configure the model for

Value selected from

- ERT (optimized for fixed-point)
- ERT (optimized for floating-point)
- GRT (optimized for fixed/floating-point)
- GRT (debug for fixed/floating-point)
- Custom

For this block, Custom is selected by default.

Configuration function

Name of the predefined or custom M-file script to be used to update the active configuration parameters of the parent Simulink model. The default value is `rtwsampleconfig`, which refers to the example M-file script `rtwsampleconfig.m`.

Invoke build process after configuration

If selected, the script initiates the code generation and build process after updating the model's configuration parameters. If not selected (the default), the build process is not initiated.

See Also

ERT (optimized for fixed-point), ERT (optimized for floating-point), GRT (debug for fixed/floating-point), GRT (optimized for fixed/floating-point)

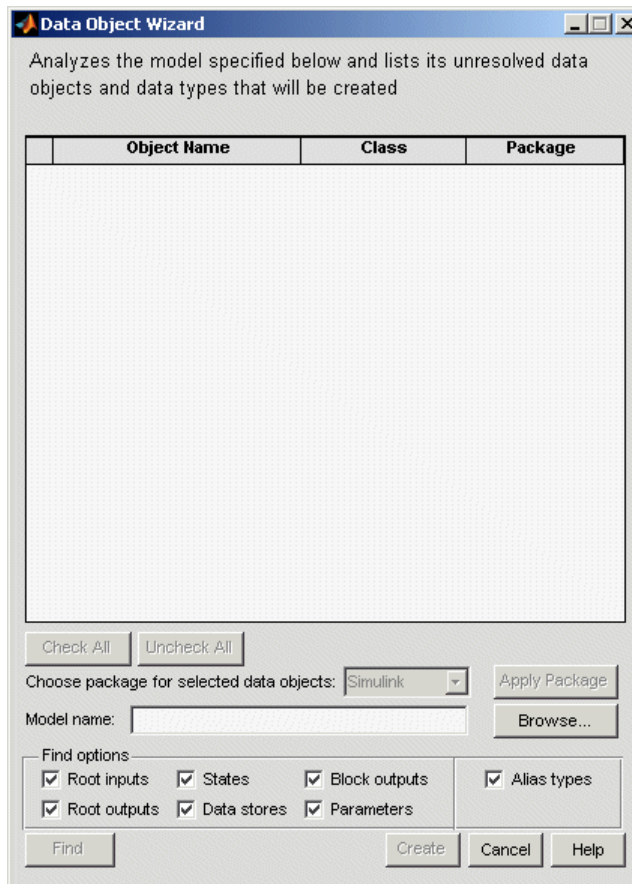
“Optimizing Your Model with Configuration Wizard Blocks and Scripts” in the Real-Time Workshop Embedded Coder documentation

Data Object Wizard

Purpose Simulink data object wizard for creating potential Simulink data objects

Library Module Packaging

Description When you add a Data Object Wizard block to your Simulink model and double-click it, the Data Object Wizard is launched:



The Data Object Wizard allows you to determine quickly which model data is not associated with Simulink data objects and to create and associate data objects with the data.

For detailed information about using the Data Object Wizard, see “Data Object Wizard” in the Simulink documentation and “Creating Simulink Data Objects with Data Object Wizard” in the Real-Time Workshop Embedded Coder documentation.

You can also launch the Data Object Wizard by entering `dataobjectwizard` at the MATLAB command line or by selecting **Data Object Wizard** from the **Tools** menu of your model.

Example

For an example of a model that incorporates the Data Object Wizard block, see `rtwdemo_mpf`.

See Also

“Data Object Wizard” in the Simulink documentation

“Creating Simulink Data Objects with Data Object Wizard” in the Real-Time Workshop Embedded Coder documentation

“Customizing Data Object Wizard User Packages” in the Real-Time Workshop Embedded Coder documentation

ERT (optimized for fixed-point)

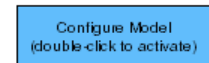
Purpose

Automatically update active configuration parameters of parent model for ERT fixed-point code generation

Library

Configuration Wizards

Description



ERT (optimized for fixed-point)

When you add an ERT (optimized for fixed-point) block to your Simulink model and double-click it, a predefined M-file script executes and automatically configures the model parameters optimally for fixed-point code generation with the ERT target. You can also set a block option to invoke the build process after configuring the model.

After double-clicking the block, you can verify that the model parameter values have changed by opening the Configuration Parameters dialog box and examining the settings.

Note You can include more than one Configuration Wizard block in your model. This provides a quick way to switch between configurations.

Parameters

Configure the model for

Value selected from

- ERT (optimized for fixed-point)
- ERT (optimized for floating-point)
- GRT (optimized for fixed/floating-point)
- GRT (debug for fixed/floating-point)
- Custom

For this block, ERT (optimized for fixed-point) is selected by default.

Configuration function

Grayed out unless **Configure the model for** is set to Custom. This parameter is used with the Custom M-file block.

Invoke build process after configuration

If selected, the script initiates the code generation and build process after updating the model's configuration parameters. If not selected (the default), the build process is not initiated.

See Also

Custom M-file, ERT (optimized for floating-point), GRT (debug for fixed/floating-point), GRT (optimized for fixed/floating-point)

“Optimizing Your Model with Configuration Wizard Blocks and Scripts” in the Real-Time Workshop Embedded Coder documentation

ERT (optimized for floating-point)

Purpose Automatically update active configuration parameters of parent model for ERT floating-point code generation

Library Configuration Wizards

Description When you add an ERT (optimized for floating-point) block to your Simulink model and double-click it, a predefined M-file script executes and automatically configures the model parameters optimally for floating-point code generation with the ERT target. You can also set a block option to invoke the build process after configuring the model.

After double-clicking the block, you can verify that the model parameter values have changed by opening the Configuration Parameters dialog box and examining the settings.

Note You can include more than one Configuration Wizard block in your model. This provides a quick way to switch between configurations.

Parameters **Configure the model for**
Value selected from

- ERT (optimized for fixed-point)
- ERT (optimized for floating-point)
- GRT (optimized for fixed/floating-point)
- GRT (debug for fixed/floating-point)
- Custom

For this block, ERT (optimized for floating-point) is selected by default.

Configuration function
Grayed out unless **Configure the model for** is set to Custom.
This parameter is used with the Custom M-file block.

Invoke build process after configuration

If selected, the script initiates the code generation and build process after updating the model's configuration parameters. If not selected (the default), the build process is not initiated.

See Also

Custom M-file, ERT (optimized for fixed-point), GRT (debug for fixed/floating-point), GRT (optimized for fixed/floating-point)

“Optimizing Your Model with Configuration Wizard Blocks and Scripts” in the Real-Time Workshop Embedded Coder documentation

GRT (debug for fixed/floating-point)

Purpose Automatically update active configuration parameters of parent model for GRT fixed- or floating-point code generation with debugging enabled

Library Configuration Wizards

Description When you add a GRT (debug for fixed/floating-point) block to your Simulink model and double-click it, a predefined M-file script executes and automatically configures the model parameters optimally for fixed/floating-point code generation, with TLC debugging options enabled, with the GRT target. You can also set a block option to invoke the build process after configuring the model.

After double-clicking the block, you can verify that the model parameter values have changed by opening the Configuration Parameters dialog box and examining the settings.

Note You can include more than one Configuration Wizard block in your model. This provides a quick way to switch between configurations.

Parameters **Configure the model for**
Value selected from

- ERT (optimized for fixed-point)
- ERT (optimized for floating-point)
- GRT (optimized for fixed/floating-point)
- GRT (debug for fixed/floating-point)
- Custom

For this block, GRT (debug for fixed/floating-point) is selected by default.

Configuration function
Grayed out unless **Configure the model for** is set to Custom. This parameter is used with the Custom M-file block.

Invoke build process after configuration

If selected, the script initiates the code generation and build process after updating the model's configuration parameters. If not selected (the default), the build process is not initiated.

See Also

Custom M-file, ERT (optimized for fixed-point), ERT (optimized for floating-point), GRT (optimized for fixed/floating-point)

“Optimizing Your Model with Configuration Wizard Blocks and Scripts” in the Real-Time Workshop Embedded Coder documentation

GRT (optimized for fixed/floating-point)

Purpose Automatically update active configuration parameters of parent model for GRT fixed- or floating-point code generation

Library Configuration Wizards

Description When you add a GRT (optimized for fixed/floating-point) block to your Simulink model and double-click it, a predefined M-file script executes and automatically configures the model parameters optimally for fixed/floating-point code generation with the GRT target. You can also set a block option to invoke the build process after configuring the model.

After double-clicking the block, you can verify that the model parameter values have changed by opening the Configuration Parameters dialog box and examining the settings.

Note You can include more than one Configuration Wizard block in your model. This provides a quick way to switch between configurations.

Parameters

Configure the model for

Value selected from

- ERT (optimized for fixed-point)
- ERT (optimized for floating-point)
- GRT (optimized for fixed/floating-point)
- GRT (debug for fixed/floating-point)
- Custom

For this block, GRT (optimized for fixed/floating-point) is selected by default.

Configuration function

Grayed out unless **Configure the model for** is set to Custom. This parameter is used with the Custom M-file block.

GRT (optimized for fixed/floating-point)

Invoke build process after configuration

If selected, the script initiates the code generation and build process after updating the model's configuration parameters. If not selected (the default), the build process is not initiated.

See Also

Custom M-file, ERT (optimized for fixed-point), ERT (optimized for floating-point), GRT (debug for fixed/floating-point)

“Optimizing Your Model with Configuration Wizard Blocks and Scripts” in the Real-Time Workshop Embedded Coder documentation

Invoke AUTOSAR Server Operation

Purpose Configure AUTOSAR client port to access Basic Software or application software components

Library Real-Time Workshop/ AUTOSAR

Description Use this block to configure an AUTOSAR client port for your Simulink model, which provides access to Basic Software or application software components:

- 1 Copy or drag this block from the AUTOSAR library into your model.
- 2 Double-click the block to open the Invoke AUTOSAR Server Operation dialog box.
- 3 Specify the parameters and click **OK**. This action updates the number of inports and outports to match the operation prototype.
- 4 Connect this block to other blocks in your model as required.
- 5 Save and build the model to generate AUTOSAR-compliant code and XML files.

Parameters

Client port name

Must be a valid AUTOSAR short-name identifier.

Operation prototype

Controls the type and number of inports and outports of the block, and must be of the form:

```
operation(prt1 datatype1 arg1, prt2 datatype2 arg2, ...  
prtN datatypeN argN, ...)
```

- *operation* — Name of the operation
- *prtN*. Either IN or OUT, which indicates whether data passes into or out of the function.

Invoke AUTOSAR Server Operation

- *datatypeN* — A string indicating data type, which can be an AUTOSAR basic data type or record, Simulink data type, or array.
- *argN* — Name of the argument

Interface path

The reference path for the client-server interface.

Server type

You select the value from:

- **Application software** — For communication with an application software component.
- **Basic software** — For communication with AUTOSAR Basic Software.

For this block, **Application software** is the default.

Show error status

If you select this, client port receives error status of client-server communication.

Sample time (-1 for inherited)

To inherit the sample time, set this parameter to -1.

See Also

Mode Switch for Invoke AUTOSAR Server Operation

“Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation

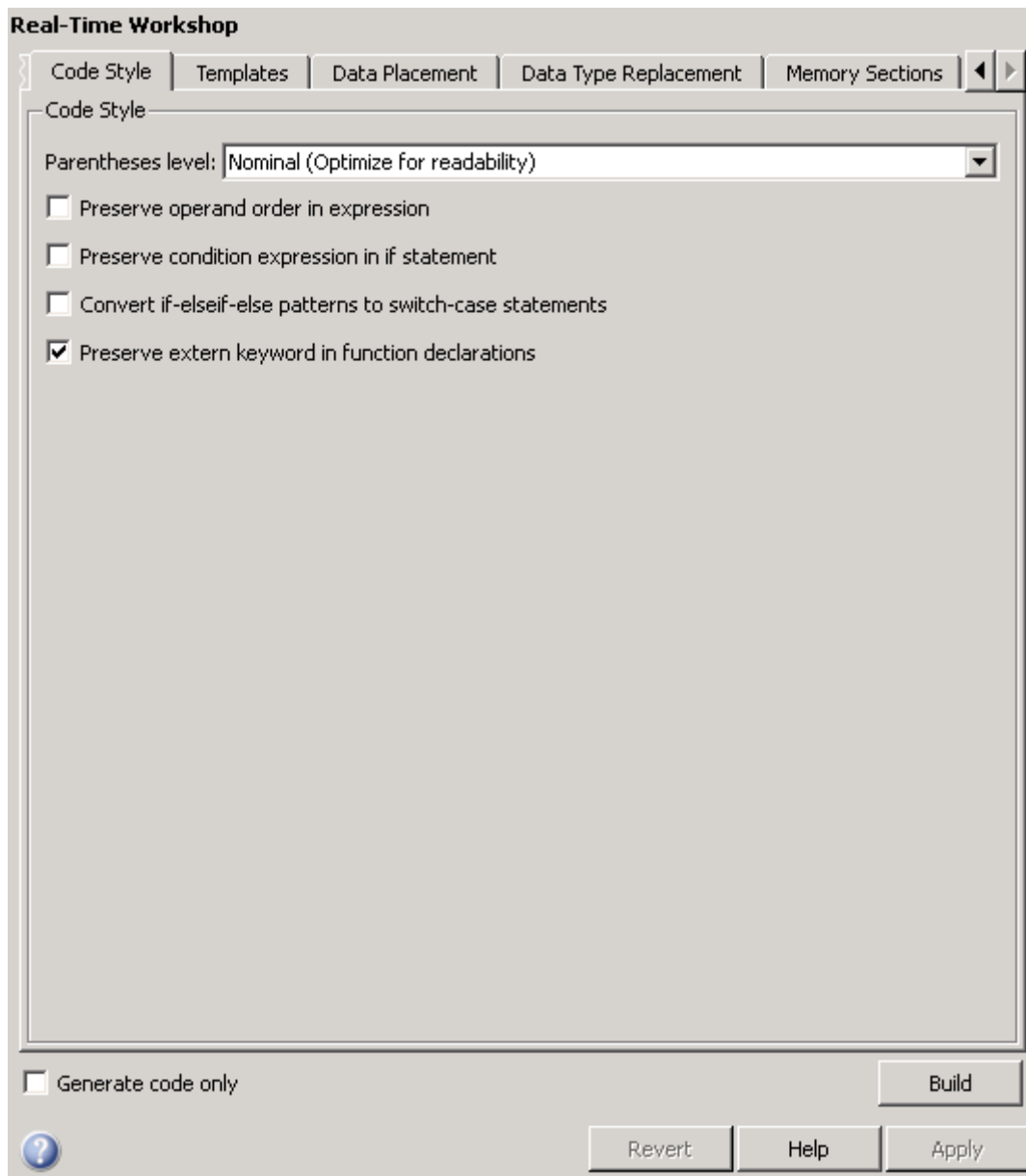
Mode Switch for Invoke AUTOSAR Server Operation

Purpose	Toggle AUTOSAR client-server operation subsystem blocks between simulation and code generation mode
Library	Real-Time Workshop/ AUTOSAR
Description	You can add this switch block to your Simulink model that contains client-server subsystem blocks. Double-click the switch block to toggle client-server blocks between simulation and code-generation mode.
Parameters	<p>Configure the model for Value selected from</p> <ul style="list-style-type: none">• code generation• simulation <p>For this block, code generation is selected by default.</p>
See Also	Invoke AUTOSAR Server Operation “Configuring Client-Server Communication” in the Real-Time Workshop Embedded Coder documentation

Configuration Parameters

- “Real-Time Workshop Pane: Code Style” on page 6-2
- “Real-Time Workshop Pane: Templates” on page 6-14
- “Real-Time Workshop Pane: Data Placement” on page 6-26
- “Real-Time Workshop Pane: Data Type Replacement” on page 6-45
- “Real-Time Workshop Pane: Memory Sections” on page 6-74
- “Real-Time Workshop Pane: AUTOSAR Code Generation Options” on page 6-91
- “Parameter Reference” on page 6-95

Real-Time Workshop Pane: Code Style



In this section...

“Real-Time Workshop: Code Style Tab Overview” on page 6-4

“Parentheses level” on page 6-5

“Preserve operand order in expression” on page 6-7

“Preserve condition expression in if statement” on page 6-8

“Convert if-elseif-else patterns to switch-case statements” on page 6-10

“Preserve extern keyword in function declarations” on page 6-12

Real-Time Workshop: Code Style Tab Overview

Control optimizations for readability in generated code.

Configuration

This tab appears only if you specify an ERT based system target file.

See Also

- “Controlling Code Style”
- “Real-Time Workshop Pane: Code Style” on page 6-2

Parentheses level

Specify parenthesization style for generated code.

Settings

Default: Nominal (Optimize for readability)

Minimum (Rely on C/C++ operators for precedence)

Inserts parentheses only where required by ANSI⁴ C or C++, or needed to override default precedence. For example:

```
isZero = var == 0;
if (isZero == 1 && (value < 3.7 || value > 9.27)) {
    /* code */
}
```

Nominal (Optimize for readability)

Inserts parentheses in a way that compromises between readability and visual complexity. The exact definition can change between releases.

Maximum (Specify precedence with parentheses)

Includes parentheses everywhere needed to specify meaning without relying on operator precedence. Code generated with this setting conforms to MISRA^{®5} requirements. For example:

```
isZero = (var == 0);
if ((isZero == 1) && ((value < 3.7) || (value > 9.27))) {
    /* code */
}
```

Command-Line Information

Parameter: ParenthesesLevel

Type: string

Value: 'Minimum' | 'Nominal' | 'Maximum'

Default: 'Nominal'

4. ANSI[®] is a registered trademark of the American National Standards Institute, Inc.

5. MISRA[®] is a registered trademarks of MIRA Ltd, held on behalf of the MISRA[®] Consortium.

Recommended Settings

Application	Setting
Debugging	Nominal (Optimized for readability)
Traceability	Nominal (Optimized for readability)
Efficiency	Minimum (Rely on C/C++ operators for precedence)
Safety precaution	Maximum (Specify precedence with parentheses)

See Also

Controlling Parenthesization

Preserve operand order in expression

Specify whether to preserve order of operands in expressions.

Settings

Default: off



On

Preserves the expression order specified in the model. Select this option to increase readability of the code or for code traceability purposes.

$$A * (B + C)$$


Off

Optimizes efficiency of code for nonoptimized compilers by reordering commutable operands to make expressions left-recursive. For example:

$$(B + C) * A$$

Command-Line Information

Parameter: PreserveExpressionOrder

Type: string

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	On
Traceability	On
Efficiency	Off
Safety precaution	On

Preserve condition expression in if statement

Specify whether to preserve empty primary condition expressions in `if` statements.

Settings

Default: off



On

Preserves empty primary condition expressions in `if` statements, such as the following, to increase the readability of the code or for code traceability purposes.

```
if expression1
else
    statements2;
end
```



Off

Optimizes empty primary condition expressions in `if` statements by negating them. For example, consider the following `if` statement:

```
if expression1
else
    statements2;
end
```

By default, the code generator negates this statement as follows:

```
if ~expression1
    statements2;
end
```

Command-Line Information

Parameter: `PreserveIfCondition`

Type: string

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	On
Traceability	On
Efficiency	Off
Safety precaution	Off

Convert if-elseif-else patterns to switch-case statements

Specify whether to generate code for if-elseif-else decision logic as switch-case statements.

This readability optimization works on a per-model basis and applies only to:

- Flow graphs in Stateflow[®] charts
- Embedded MATLAB functions in Stateflow charts
- Embedded MATLAB Function blocks in that model

Settings

Default: off



On

Generate code for if-elseif-else decision logic as switch-case statements.

For example, assume that you have the following logic pattern:

```
if (x == 1) {  
    y = 1;  
} else if (x == 2) {  
    y = 2;  
} else if (x == 3) {  
    y = 3;  
} else {  
    y = 4;  
}
```

Selecting this check box converts the if-elseif-else pattern to the following switch-case statements:

```
switch (x) {  
    case 1:  
        y = 1; break;  
    case 2:  
        y = 2; break;  
}
```

```

        case 3:
            y = 3; break;
        default:
            y = 4; break;
    }

```



Off

Preserve if-elseif-else decision logic in generated code.

Command-Line Information

Parameter: ConvertIfToSwitch

Type: string

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Off
Efficiency	On
Safety precaution	No impact

See Also

- “Enhancing Readability of Generated Code for Flow Graphs”
- “Enhancing Readability of Generated Code for Embedded MATLAB Function Blocks”
- “Controlling Code Style”

Preserve extern keyword in function declarations

Specify whether to include the `extern` keyword in function declarations in the generated code.

Note The `extern` keyword is optional for functions with external linkage. It is considered good programming practice to include the `extern` keyword in function declarations for code readability.

Settings

Default: on



On

Include the `extern` keyword in function declarations in the generated code. For example, the generated code for the demo model `rtwdemo_hyperlinks` contains the following function declarations in `rtwdemo_hyperlinks.h`:

```
/* Model entry point functions */
extern void rtwdemo_hyperlinks_initialize(void);
extern void rtwdemo_hyperlinks_step(void);
```

The `extern` keyword explicitly indicates that the function has external linkage. The function definitions in this example are in the generated file `rtwdemo_hyperlinks.c`.



Off

Remove the `extern` keyword from function declarations in the generated code.

Command-Line Information

Parameter: `PreserveExternInFcnDecls`

Type: string

Value: `'on'` | `'off'`

Default: `'on'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

For more information on code style options, see “Real-Time Workshop Pane: Code Style” on page 6-2

Real-Time Workshop Pane: Templates

Real-Time Workshop

Code Style | **Templates** | Data Placement | Data Type Replacement | Memory Sections

Code templates

Source file (*.c) template: Browse... Edit...

Header file (*.h) template: Browse... Edit...

Data templates

Source file (*.c) template: Browse... Edit...

Header file (*.h) template: Browse... Edit...

Custom templates

File customization template: Browse... Edit...

Generate an example main program

Target operating system:

Generate code only Build

? Revert Help Apply

In this section...

“Real-Time Workshop: Templates Tab Overview” on page 6-16

“Code templates: Source file (*.c) template” on page 6-17

“Code templates: Header file (*.h) template” on page 6-18

“Data templates: Source file (*.c) template” on page 6-19

“Data templates: Header file (*.h) template” on page 6-20

“File customization template” on page 6-21

“Generate an example main program” on page 6-22

“Target operating system” on page 6-24

Real-Time Workshop: Templates Tab Overview

Customize the organization of your generated code.

Configuration

This tab appears only if you specify an ERT based system target file.

See Also

“Real-Time Workshop Pane: Templates” on page 6-14

Code templates: Source file (*.c) template

Specify the code generation template (CGT) file to use when generating a source code file.

Settings

Default: ert_code_template.cgt

You can use a CGT file to define the top-level organization and formatting of generated source code files (.c or .cpp).

Note The CGT file must be located on the MATLAB path.

Command-Line Information

Parameter: ERTSrcFileBannerTemplate

Type: string

Value: any valid CGT file

Default: 'ert_code_template.cgt'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- Selecting and Defining Templates
- Custom File Processing

Code templates: Header file (*.h) template

Specify the code generation template (CGT) file to use when generating a code header file.

Settings

Default: ert_code_template.cgt

You can use a CGT file to define the top-level organization and formatting of generated header files (.h).

Note The CGT file must be located on the MATLAB path.

Command-Line Information

Parameter: ERTHdrFileBannerTemplate

Type: string

Value: any valid CGT file

Default: 'ert_code_template.cgt'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- Selecting and Defining Templates
- Custom File Processing

Data templates: Source file (*.c) template

Specify the code generation template (CGT) file to use when generating a data source file.

Settings

Default: ert_code_template.cgt

You can use a CGT file to define the top-level organization and formatting of generated data source files (.c or .cpp) that contain definitions of variables of global scope.

Note The CGT file must be located on the MATLAB path.

Command-Line Information

Parameter: ERTDataSrcFileTemplate
Type: string
Value: any valid CGT file
Default: 'ert_code_template.cgt'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- Selecting and Defining Templates
- Custom File Processing

Data templates: Header file (*.h) template

Specify the code generation template (CGT) file to use when generating a data header file.

Settings

Default: ert_code_template.cgt

You can use a CGT file to define the top-level organization and formatting of generated data header files (.h) that contain declarations of variables of global scope.

Note The CGT file must be located on the MATLAB path.

Command-Line Information

Parameter: ERTDataHdrFileTemplate

Type: string

Value: any valid CGT file

Default: 'ert_code_template.cgt'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- Selecting and Defining Templates
- Custom File Processing

File customization template

Specify the custom file processing (CFP) template file to use when generating code.

Settings

Default: ert_code_template.tlc

You can use a CFP template file to customize generated code. A CFP template file is a TLC file that organizes types of code (for example, includes, typedefs, and functions) into sections. The primary purpose of a CFP template is to assemble code to be generated into buffers, and to call a code template API to emit the buffered code into specified sections of generated source and header files. The CFP template file must be located on the MATLAB path.

Command-Line Information

Parameter: ERTCustomFileTemplate

Type: string

Value: any valid TLC file

Default: 'example_file_process.tlc'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- Selecting and Defining Templates
- Custom File Processing

Generate an example main program

Control whether to generate an example main program for a model.

Settings

Default: on



On

Generates an example main program, `ert_main.c` (or `.cpp`). The file includes:

- The `main()` function for the generated program
- Task scheduling code that determines how and when block computations execute on each time step of the model

The operation of the main program and the scheduling algorithm employed depend primarily on whether your model is single-rate or multirate, and also on your model's solver mode (`SingleTasking` or `MultiTasking`).



Off

Provides a static version of the file `ert_main.c` as a basis for custom modifications (*matlabroot/rtw/c/ert/ert_main.c*). You can use this file as a template for developing embedded applications.

Tips

- After you generate and customize the main program, disable this option to prevent regenerating the main module and overwriting your customized version.
- You can use a custom file processing (CFP) template file to override normal main program generation, and generate a main program module customized for your target environment.
- If you disable this option, the coder generates slightly different rate grouping code to maintain compatibility with an older static `ert_main.c` module.

Dependencies

- This parameter enables **Target operating system**.
- You must enable this parameter and select VxWorksExample for **Target operating system** if you use VxWorks^{®6} library blocks.

Command-Line Information

Parameter: GenerateSampleERTMain

Type: string

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- “Generating a Standalone Program”
- Static Main Program Module
- Custom File Processing

6. VxWorks[®] is a registered trademark of Wind River[®] Systems, Inc.

Target operating system

Specify a target operating system to use when generating model-specific example main program module.

Settings

Default: BareBoardExample

BareBoardExample

Generates a bareboard main program designed to run under control of a real-time clock, without a real-time operating system.

VxWorksExample

Generates a fully commented example showing how to deploy the code under the VxWorks real-time operating system.

Dependencies

- This parameter is enabled by **Generate an example main program**.
- This parameter must be the same for top-level and referenced models.

Command-Line Information

Parameter: TargetOS

Type: string

Value: 'BareBoardExample' | 'VxWorksExample'

Default: 'BareBoardExample'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- “Generating a Standalone Program”
- Static Main Program Module
- Custom File Processing

Real-Time Workshop Pane: Data Placement

Real-Time Workshop

Code Style | Templates | **Data Placement** | Data Type Replacement | Memory Sections

Global data placement (custom storage classes only)

Data definition: Auto

Data declaration: Auto

#include file delimiter: Auto

Global data placement (MPT data objects only)

Module naming: Not specified

Signal display level: 10 Parameter tune level: 10

Generate code only Build

Revert Help Apply

In this section...

“Real-Time Workshop: Data Placement Tab Overview” on page 6-28

“Data definition” on page 6-29

“Data definition filename” on page 6-31

“Data declaration” on page 6-33

“Data declaration filename” on page 6-35

“#include file delimiter” on page 6-36

“Module naming” on page 6-37

“Module name” on page 6-39

“Signal display level” on page 6-41

“Parameter tune level” on page 6-43

Real-Time Workshop: Data Placement Tab Overview

Specify the data placement in the generated code.

Configuration

This tab appears only if you specify an ERT based system target file.

See Also

- “Defining Data Representation and Storage for Code Generation”
- “Real-Time Workshop Pane: Data Placement” on page 6-26

Data definition

Specify where to place definitions of global variables.

Settings

Default: Auto

Auto

Lets the code generator determine where the definitions should be located.

Data defined in source file

Places definitions in `.c` source files where functions are located. The code generator places the definitions in one or more function `.c` files, depending on the number of function source files and the file partitioning previously selected in the Simulink model.

Data defined in a single separate source file

Places definitions in the source file specified in the **Data definition filename** field. The code generator organizes and formats the definitions based on the data source template specified by the **Source file (*.c) template** parameter in the data section of the **Templates** pane.

Dependencies

- This parameter applies to data with custom storage classes only.
- This parameter enables **Data definition filename**.

Command-Line Information

Parameter: GlobalDataDefinition

Type: string

Value: 'Auto' | 'InSourceFile' | 'InSeparateSourceFile'

Default: 'Auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid value
Efficiency	No impact
Safety precaution	No impact

See Also

- “Overview of Data Placement”
- “Managing Placement of Data Definitions and Declarations”
- “Data Placement Rules and Effects”

Data definition filename

Specify the name of the file that is to contain data definitions.

Settings

Default: `global.c` or `global.cpp`

The code generator organizes and formats the data definitions in the specified file based on the data source template specified by the **Source file (*.c) template** parameter in the data section of the **Real-Time Workshop** pane: **Templates** tab.

If you specify C++ as the target language, omit the `.cpp` extension. The code generator will generate the correct file and add the extension `.cpp`.

Dependency

This parameter is enabled by **Data definition**.

Command-Line Information

Parameter: `DataDefinitionFile`

Type: `string`

Value: any valid file

Default: `'global.c'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid file
Efficiency	No impact
Safety precaution	No impact

See Also

- [Selecting and Defining Templates](#)
- [Custom File Processing](#)

Data declaration

Specify where extern, typedef, and #define statements are to be declared.

Settings

Default: Auto

Auto

Lets the code generator determine where the declarations should be located.

Data declared in source file

Places declarations in .c source files where functions are located. The data header template file is not used. The code generator places the declarations in one or more function .c files, depending on the number of function source files and the file partitioning previously selected in the Simulink model.

Data defined in a single separate source file

Places declarations in the data header file specified in the **Data declaration filename** field. The code generator organizes and formats the declarations based on the data header template specified by the **header file (*.h) template** parameter in the data section of the **Real-Time Workshop** pane: **Templates** tab.

Dependencies

- This parameter applies to data with custom storage classes only.
- This parameter enables **Data declaration filename**.

Command-Line Information

Parameter: GlobalDataReference

Type: string

Value: 'Auto' | 'InSourceFile' | 'InSeparateHeaderFile'

Default: 'Auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid value
Efficiency	No impact
Safety precaution	No impact

See Also

- “Overview of Data Placement”
- “Managing Placement of Data Definitions and Declarations”
- “Data Placement Rules and Effects”

Data declaration filename

Specify the name of the file that is to contain data declarations.

Settings

Default: global.h

The code generator organizes and formats the data declarations in the specified file based on the data header template specified by the **Header file (*.h) template** parameter in the data section of the **Real-Time Workshop** pane: **Templates** tab.

Dependency

This parameter is enabled by **Data declaration**.

Command-Line Information

Parameter: DataReferenceFile

Type: string

Value: any valid file

Default: 'global.h'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid file
Efficiency	No impact
Safety precaution	No impact

See Also

- Selecting and Defining Templates
- Custom File Processing

#include file delimiter

Specify the type of #include file delimiter to use in generated code.

Settings

Default: Auto

Auto

Lets the code generator choose the #include file delimiter

#include header.h

Uses double quote (" ") characters to delimit file names in #include statements.

#include <header.h>

Uses angle brackets (< >) to delimit file names in #include statements.

Dependency

The delimiter format that you use when specifying parameter and signal object property values overrides what you set for this parameter.

Command-Line Information

Parameter: IncludeFileDelimiter

Type: string

Value: 'Auto' | 'UseQuote' | 'UseBracket'

Default: 'Auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid value
Efficiency	No impact
Safety precaution	No impact

Module naming

Specify whether to name the module that owns the model.

Settings

Default: Not specified

Not specified

Lets the code generator determine the module name.

Same as model

Uses the name of the model for the module name.

User specified

Uses the module name specified for **Module name** parameter for the module name.

Command-Line Information

Parameter: ModuleNamingRule

Type: string

Value: 'Unspecified' | 'SameAsModel' | 'UserSpecified'

Default: 'Unspecified'

Dependency

- Selecting **User specified** enables **Module name**.
- Use this parameter with the data object property **Owner** to specify module ownership.
- This parameter must be the same for top-level and referenced models.

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid value

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

- “Overview of Data Placement”
- Ownership Settings

Module name

Specify the name of module that is to own the model.

Settings

Default: ''

Specify a module name according to ANSI⁷ C/C++ conventions for naming identifiers.

Dependency

- This parameter is enabled by User specified.
- This parameter must be the same for top-level and referenced models.

Command-Line Information

Parameter: ModuleName
Type: string
Value: any valid name
Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid name
Efficiency	No impact
Safety precaution	No impact

See Also

- “Overview of Data Placement”

7. ANSI[®] is a registered trademark of the American National Standards Institute, Inc.

- Ownership Settings

Signal display level

Specify the persistence level for all MPT signal data objects.

Settings

Default: 10

Specify an integer value indicating the persistence level for all MPT signal data objects. This value indicates the level at which to declare signal data objects as global data in the generated code. The persistence level allows you to make intermediate variables global during initial development so you can remove them during later stages of development to gain efficiency.

This parameter is related to the **Persistence level** value that you can specify for a specific MPT signal data object in the Model Explorer signal properties dialog.

Dependency

This parameter must be the same for top-level and referenced models.

Command-Line Information

Parameter: SignalDisplayLevel

Type: integer

Value: any valid integer

Default: 10

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid integer
Efficiency	No impact
Safety precaution	No impact

See Also

Selecting Persistence Level for Signals and Parameters

Parameter tune level

Specify the persistence level for all MPT parameter data objects.

Settings

Default: 10

Specify an integer value indicating the persistence level for all MPT parameter data objects. This value indicates the level at which to declare parameter data objects as tunable global data in the generated code. The persistence level allows you to make intermediate variables global and tunable during initial development so you can remove them during later stages of development to gain efficiency.

This parameter is related to the **Persistence level** value you that can specify for a specific MPT parameter data object in the Model Explorer parameter properties dialog.

Dependency

This parameter must be the same for top-level and referenced models.

Command-Line Information

Parameter: ParamTuneLevel

Type: integer

Value: any valid integer

Default: 10

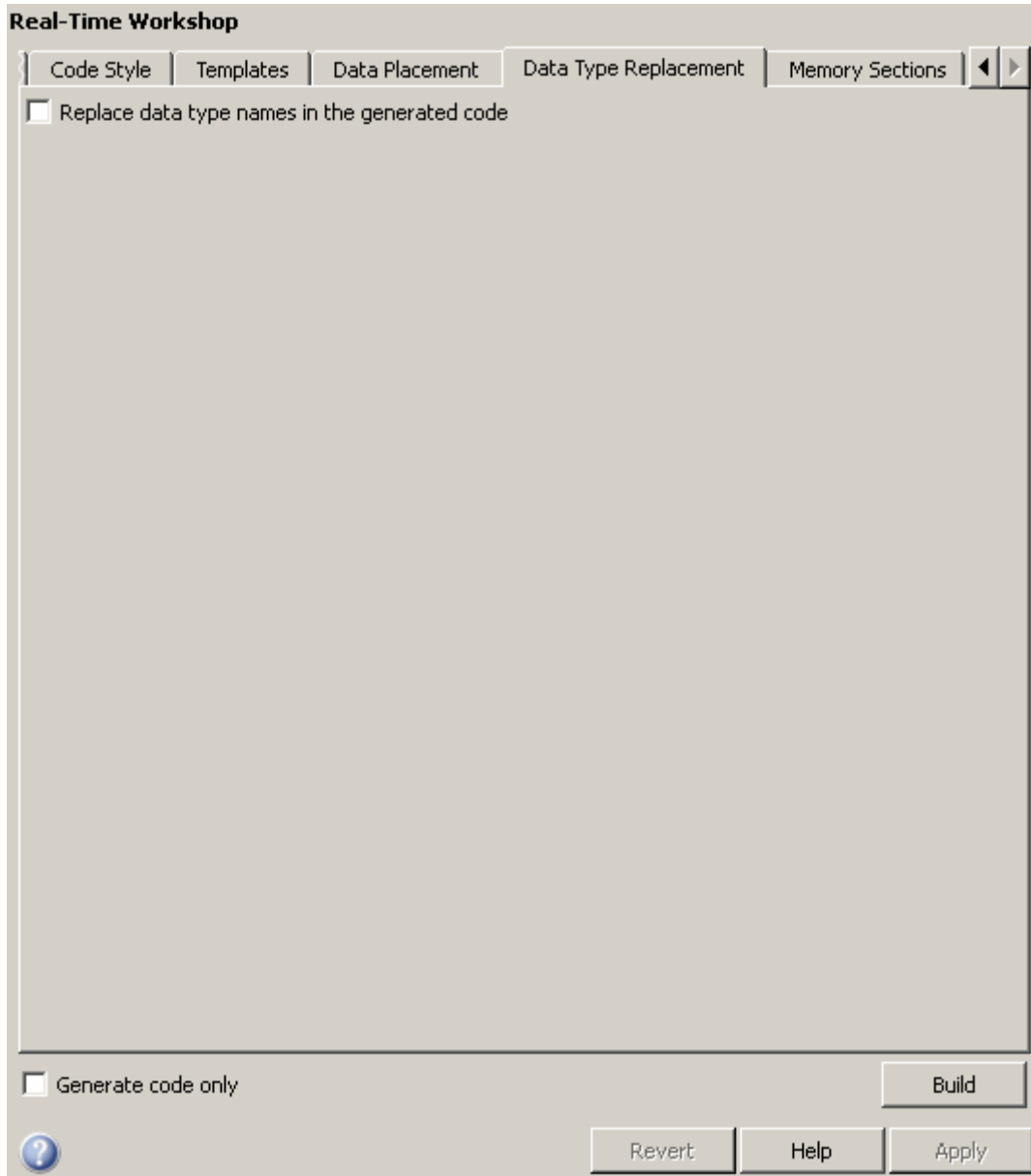
Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid integer
Efficiency	No impact
Safety precaution	No impact

See Also

Selecting Persistence Level for Signals and Parameters

Real-Time Workshop Pane: Data Type Replacement



In this section...

“Real-Time Workshop: Data Type Replacement Tab Overview” on page 6-47

“Replace data type names in the generated code” on page 6-48

“Replacement Name: double” on page 6-50

“Replacement Name: single” on page 6-52

“Replacement Name: int32” on page 6-54

“Replacement Name: int16” on page 6-56

“Replacement Name: int8” on page 6-58

“Replacement Name: uint32” on page 6-60

“Replacement Name: uint16” on page 6-62

“Replacement Name: uint8” on page 6-64

“Replacement Name: boolean” on page 6-66

“Replacement Name: int” on page 6-68

“Replacement Name: uint” on page 6-70

“Replacement Name: char” on page 6-72

Real-Time Workshop: Data Type Replacement Tab Overview

Replace built-in data type names with user-defined replacement data type names in the generated code for your model.

Configuration

This tab appears only if you specify an ERT based system target file.

If your application requires you to replace built-in data type names with user-defined replacement data type names in the generated code:

- 1 Select **Replace data type names in the generated code**.
- 2 Selectively specify replacement data type names to use for built-in Simulink data types in the **Replacement Name** fields.

See Also

- “Replacing Built-In Data Type Names in Generated Code”
- “Real-Time Workshop Pane: Data Type Replacement” on page 6-45

Replace data type names in the generated code

Specify whether to replace built-in data type names with user-defined data type names in generated code.

Settings

Default: off



On

Displays the **Data type names** table. The table provides a way for you to replace the names of built-in data types used in generated code. This mechanism can be particularly useful for generating code that adheres to application or site data type naming standards.

You can choose to specify new data type names for some or all Simulink built-in data types listed in the table. For each replacement data type name that you specify:

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- For `double`, `single`, `int32`, `int16`, `int8`, `uint32`, `uint16`, `uint8`, and `boolean`, the `BaseType` of the replacement data type must match the built-in data type.
- For `int`, `uint`, and `char`, the size of the replacement data type must match the size displayed for `int` or `char` on the **Hardware Implementation** pane of the Configuration Parameters dialog box.

An error occurs if a replacement data type specification is inconsistent.



Off

Uses Real-Time Workshop names for built-in Simulink data types in generated code.

Dependencies

This parameter enables:

double Replacement Name

single Replacement Name
int32 Replacement Name
int16 Replacement Name
int8 Replacement Name
uint32 Replacement Name
uint16 Replacement Name
uint8 Replacement Name
boolean Replacement Name
int Replacement Name
uint Replacement Name
char Replacement Name

Command-Line Information

Parameter: EnableUserReplacementTypes
Type: string
Value: 'on' | 'off'
Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	On
Efficiency	No impact
Safety precaution	Off

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: double

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.double`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: single

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.single`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: int32

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.int32`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: int16

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types .

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.int16`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: int8

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.int8`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: uint32

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.uint32`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: uint16

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.uint16`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: uint8

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.uint8`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: boolean

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The `BaseType` of the replacement data type must match the built-in data type.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.boolean`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and `BaseType` of the replacement data type must match the built-in data type

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: int

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The size of the replacement data type must match the size displayed on the **Hardware Implementation** pane of the Configuration Parameters dialog box.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.int`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and the size of the replacement data type must match the size displayed on the **Hardware Implementation** pane of the Configuration Parameters dialog box

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid value
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: uint

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The size of the replacement data type must match the size displayed on the **Hardware Implementation** pane of the Configuration Parameters dialog box.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.uint`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and the size of the replacement data type must match the size displayed on the **Hardware Implementation** pane of the Configuration Parameters dialog box

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Replacement Name: char

Specify names to use for built-in Simulink data types in generated code.

Settings

Default: ''

Specify strings that the code generator is to use as names for built-in Simulink data types.

- The name must match the name of a `Simulink.AliasType` object that exists in the base workspace.
- The `BaseType` property of the associated `Simulink.AliasType` object must be consistent with the built-in data type it replaces.
- The size of the replacement data type must match the size displayed for on the **Hardware Implementation** pane of the Configuration Parameters dialog box.

An error occurs if a replacement data type specification is inconsistent.

Dependency

This parameter is enabled by **Replace data type names in the generated code**.

Command-Line Information

Parameter: `ReplacementTypes`, `replacementName.char`

Type: string

Value: name of a `Simulink.AliasType` object that exists in the base workspace; `BaseType` property of object must be consistent with the built-in data type it replaces and the size of the replacement data type must match the size displayed on the **Hardware Implementation** pane of the Configuration Parameters dialog box

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string
Efficiency	No impact
Safety precaution	' '

See Also

“Replacing Built-In Data Type Names in Generated Code”

Real-Time Workshop Pane: Memory Sections

Real-Time Workshop

Code Style | Templates | Data Placement | Data Type Replacement | Memory Sections ◀ ▶

Package containing memory sections for model data and functions

Package: --- None --- Refresh package list

Memory sections for model functions and subsystem defaults

Initialize/Terminate: Default

Execution: Default

Memory sections for model data and subsystem defaults

Constants: Default

Inputs/Outputs: Default

Internal data: Default

Parameters: Default

Validation results

Package and memory sections found.

Generate code only Build

Revert Help Apply

In this section...

“Real-Time Workshop: Memory Sections Tab Overview” on page 6-76

“Package” on page 6-77

“Refresh package list” on page 6-79

“Initialize/Terminate” on page 6-80

“Execution” on page 6-81

“Constants” on page 6-82

“Inputs/Outputs” on page 6-84

“Internal data” on page 6-86

“Parameters” on page 6-88

“Validation results” on page 6-90

Real-Time Workshop: Memory Sections Tab Overview

Insert comments and pragmas into the generated code for data and functions.

Configuration

This tab appears only if you specify an ERT based system target file.

See Also

- Memory Sections
- “Real-Time Workshop Pane: Memory Sections” on page 6-74

Package

Specify a package that contains memory sections you want to apply to model-level functions and internal data.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: ---None---

---None---

Suppresses memory sections.

Simulink

Applies the built-in Simulink package.

mpt

Applies the built-in mpt package.

Tip

If you have defined any packages of your own, click **Refresh package list**. This action adds all user-defined packages on your search path to the package list.

Command-Line Information

Parameter: MemSecPackage

Type: string

Value: '--- None ---' | 'Simulink' | 'mpt'

Default: '--- None ---'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Refresh package list

Add user-defined packages that are on the search path to list of packages displayed by **Packages**.

Tip

If you have defined any packages of your own, click **Refresh package list**. This action adds all user-defined packages on your search path to the package list.

See Also

Memory Sections

Initialize/Terminate

Specify whether to apply a memory section to Initialize/Start and Terminate functions.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: Default

Default

Suppresses the use of a memory section for Initialize, Start and Terminate functions.

memory-section-name

Applies a memory section to Initialize, Start and Terminate functions.

Command-Line Information

Parameter: MemSecFuncInitTerm

Type: string

Value: 'Default' | 'MemConst' | 'MemVolatile' | 'MemConstVolatile'

Default: 'Default'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Execution

Specify whether to apply a memory section to execution functions.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: Default

Default

Suppresses the use of a memory section for Step, Run-time initialization, Derivative, Enable, and Disable functions.

memory-section-name

Applies a memory section to Step, Run-time initialization, Derivative, Enable, and Disable functions.

Command-Line Information

Parameter: MemSecFuncExecute

Type: string

Value: 'Default' | 'MemConst' | 'MemVolatile' | 'MemConstVolatile'

Default: 'Default'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Constants

Specify whether to apply a memory section to constants.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: Default

Default

Suppresses the use of a memory section for constants.

memory-section-name

Applies a memory section to constants.

This parameter applies to:

Data Definition	Data Purpose
<i>model_CP</i>	Constant parameters
<i>model_CB</i>	Constant block I/O
<i>model_Z</i>	Zero representation

Command-Line Information

Parameter: MemSecDataConstants

Type: string

Value: 'Default' | 'MemConst' | 'MemVolatile' | 'MemConstVolatile'

Default: 'Default'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Inputs/Outputs

Specify whether to apply a memory section to root input and output.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: Default

Default

Suppresses the use of a memory section for root-level input and output.

memory-section-name

Applies a memory section for root-level input and output.

This parameter applies to:

Data Definition	Data Purpose
<i>model_U</i>	Root-level input
<i>model_Y</i>	Root-level output

Command-Line Information

Parameter: MemSecDataIO

Type: string

Value: 'Default' | 'MemConst' | 'MemVolatile' | 'MemConstVolatile'

Default: 'Default'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Internal data

Specify whether to apply a memory section to internal data.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: Default

Default

Suppresses the use of a memory section for internal data.

memory-section-name

Applies a memory section for internal data.

This parameter applies to:

Data Definition	Data Purpose
<i>model_B</i>	Block I/O
<i>model_D</i>	DWork vectors
<i>model_M</i>	Run-time model
<i>model_Zero</i>	Zero-crossings

Command-Line Information

Parameter: MemSecDataInternal

Type: string

Value: 'Default' | 'MemConst' | 'MemVolatile' | 'MemConstVolatile'

Default: 'Default'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Parameters

Specify whether to apply a memory section to parameters.

Settings

Memory section specifications for model-level functions and internal data apply to the top level of the model and to all subsystems except atomic subsystems that contain overriding memory section specifications.

Default: Default

Default

Suppress the use of a memory section for parameters.

memory-section-name

Apply memory section for parameters.

This parameter applies to:

Data Definition	Data Purpose
<i>model_P</i>	Parameters

Command-Line Information

Parameter: MemSecDataParameters

Type: string

Value: 'Default' | 'MemConst' | 'MemVolatile' | 'MemConstVolatile'

Default: 'Default'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Memory Sections

Validation results

Display the results of memory section validation.

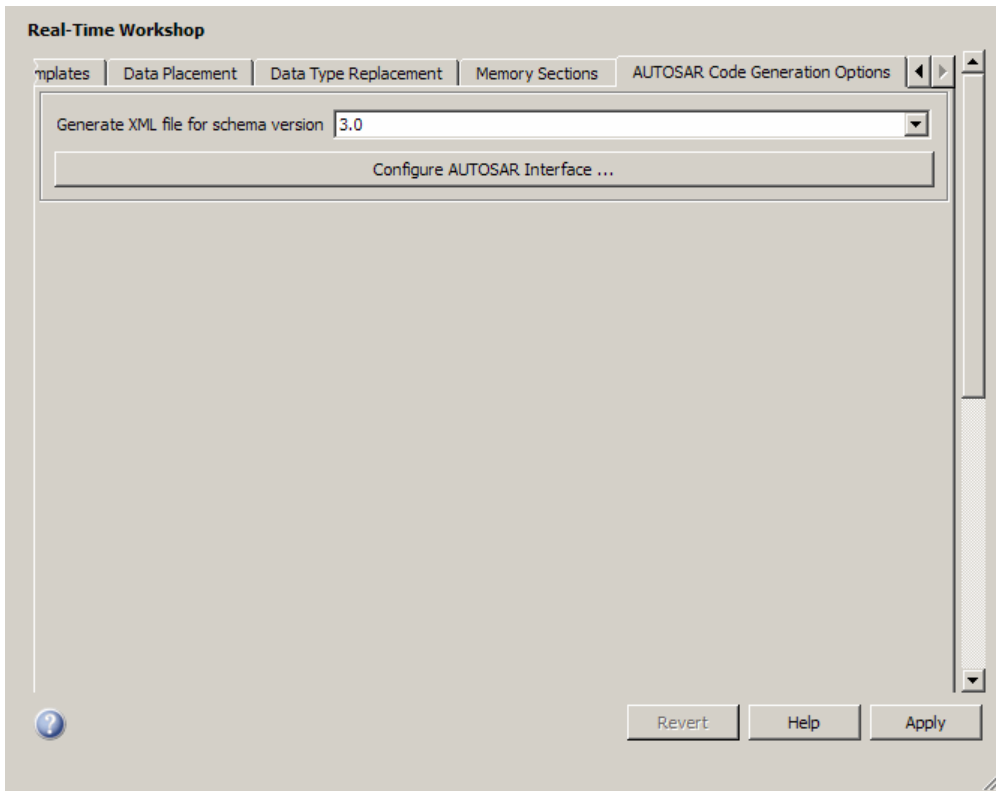
Settings

The Real-Time Workshop software checks and reports whether the currently chosen package is on the MATLAB path and that the selected memory sections exist inside the package.

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

Real-Time Workshop Pane: AUTOSAR Code Generation Options



In this section...

“Real-Time Workshop: AUTOSAR Code Generation Options Tab Overview”
on page 6-92

“Generate XML file from schema version” on page 6-93

“Configure AUTOSAR Interface” on page 6-94

Real-Time Workshop: AUTOSAR Code Generation Options Tab Overview

Parameters for controlling AUTOSAR code generation options.

Configuration

This pane appears only if you specify the `autosar.tlc` system target file.

Tip

Click the **Configure AUTOSAR Interface** button to open a dialog box where you can configure all other AUTOSAR options.

See Also

- “Generating Code That Complies with AUTOSAR Standards”
- “AUTOSAR Configuration” on page 1-3
- “AUTOSAR” on page 1-2
- “Real-Time Workshop Pane: AUTOSAR Code Generation Options” on page 6-91

Generate XML file from schema version

Select the AUTOSAR schema version to use when generating XML files.

Settings

Default: 3.0

3.0

Use schema version 3.0 for XML file generation.

2.1

Use schema version 2.1 for XML file generation.

2.0

Use schema version 2.0 for XML file generation.

Tip

Click the **Configure AUTOSAR Interface** button to open a dialog box where you can configure all other AUTOSAR options.

Command-Line Information

Parameter: AutosarSchemaVersion

Type: string

Value: '3.0' | '2.1' | '2.0'

Default: '3.0'

See Also

“Generating Code That Complies with AUTOSAR Standards”

Configure AUTOSAR Interface

Opens the Model Interface dialog box where you can configure all other AUTOSAR options.

Dependencies

This parameter is disabled if you are using Configuration Set Reference.

Command-Line Information

Parameter: `autosar_gui_launch`

Type: String

Value: *subsystemName*

Default: No default

See Also

- “Using the Configure AUTOSAR Interface Dialog Box”
- “Generating Code That Complies with AUTOSAR Standards”

Parameter Reference

In this section...

“Recommended Settings Summary” on page 6-95

“Parameter Command-Line Information Summary” on page 6-107

Recommended Settings Summary

The following table summarizes the impact of each Real-Time Workshop Embedded Coder configuration parameter on debugging, traceability, efficiency, and safety considerations, and indicates the factory default configuration settings for the ERT target. The Real-Time Workshop configuration parameters are documented in “Recommended Settings Summary” in the Real-Time Workshop documentation. For additional details, click the links in the Configuration Parameter column.

Mapping of Application Requirements to the Optimization Pane

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Application lifespan (days)	No impact	No impact	Optimal finite value	inf	1 for ERT targets
Parameter structure	No impact	Hierarchical	Non-Hierarchical	No impact	Hierarchical
Pack Boolean data into bitfields	No impact	No Impact	On	No impact	Off
Simplify array indexing	No impact	No impact	On	No impact	Off
Pass reusable subsystem outputs as	No impact	No impact	Individual arguments	No impact	Structure reference
Remove root level I/O zero initialization	No impact	No impact	On	Off	Off

Mapping of Application Requirements to the Optimization Pane (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Remove internal data zero initialization	No impact	No impact	On	Off	Off
Optimize initialization code for model reference	No impact	No impact	On	No impact	On
Remove code that protects against division arithmetic exceptions	No impact	No impact	On	Off	Off

Mapping of Application Requirements to the Real-Time Workshop Pane

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Ignore custom storage classes	No impact	No impact	No impact	No impact	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Report Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Code-to-model	On	On	No impact	On	Off
Model-to-code	On	On	No impact	On	Off
Eliminated / virtual blocks	On	On	No impact	On	Off
Traceable Simulink blocks	On	On	No impact	On	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Report Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Traceable Stateflow objects	On	On	No impact	On	Off
Traceable Embedded MATLAB functions	On	On	No impact	On	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Comments Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Simulink block descriptions	On	On	No impact	No impact	Off
Simulink data object descriptions	On	On	No impact	No impact	Off
Custom comments (MPT objects only)	On	On	No impact	No impact	Off
Custom comments function	Any valid file name	Any valid file name	No impact	No impact	' '

Mapping of Application Requirements to the Real-Time Workshop Pane: Comments Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Stateflow object descriptions	On	On	No impact	No impact	Off
Requirements in block comments	On	On	No impact	On	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Symbols Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Global variables	No impact	Any valid combination of tokens	No impact	\$R\$N\$M	\$R\$N\$M
Global types	No impact	Any valid combination of tokens	No impact	\$N\$R\$M	&N\$R\$M
Field name of global types	No impact	Any valid combination of tokens	No impact	\$N\$M	\$N\$M
Subsystem methods	No impact	Any valid combination of tokens	No impact	\$R\$N\$M\$F	\$R\$N\$M\$F
Subsystem method arguments	No impact	Any valid combination of tokens	No impact	rtu_ \$N\$M or rty_ \$N\$M	rtu_ \$N\$M or rty_ \$N\$M
Local temporary variables	No impact	Any valid combination of tokens	No impact	\$N\$M	\$N\$M

Mapping of Application Requirements to the Real-Time Workshop Pane: Symbols Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Local block output variables	No impact	Any valid combination of tokens	No impact	rtb_\$\$M	rtb_\$\$M
Constant macros	No impact	Any valid combination of tokens	No impact	\$\$M	\$\$M
Minimum mangle length	No impact	1	No impact	4	1
Generate scalar inlined parameters as	No impact	Macros	Literals	No impact	Literals
#define naming	No impact	Force uppercase	No impact	No impact	None
Parameter naming	No impact	Force uppercase	No impact	No impact	None
Signal naming	No impact	Force uppercase	No impact	No impact	None
M-function	No impact	No impact	No impact	No impact	' '

Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Support: floating-point numbers	No impact	No impact	Off (GUI), 'on' (command-line) for integer only	No impact	On (GUI), 'off' (command-line)

Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Support complex numbers	No impact	No impact	Off for real only	No impact	On
Support non-finite numbers	No impact	No impact	Off	Off	On
Support absolute time	No impact	No impact	Off	Off	On
Support continuous time	No impact	No impact	Off	Off	Off
Support non-inlined S-functions	No impact	No impact	Off	Off	Off
Support variable-size signals	No impact	No impact	Off	Off	Off
Multiword type definitions	No impact	No impact	Specifying User defined and a low value for Maximum word length reduces the size of the generated file <code>rtwtypes.h</code>	Use default	System defined

Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Maximum word length	No impact	No impact	Smaller values reduce the size of the generated file <code>rtwtypes.h</code>	Use default	256
GRT compatible call interface	No impact	Off	Off	Off	Off
Single output/update function	On	On	On	On	On
Terminate function required	No impact	No impact	Off	Off	On
Generate reusable code	No impact	No impact	Set for single instance	No impact	Off
Reusable code error diagnostic	Warning or Error	No impact	None	No impact	Error
Pass root-level I/O as	No impact	No impact	No impact	No impact	Individual arguments
Block parameter visibility	No impact	No impact	No impact	protected	private
Internal data visibility	No impact	No impact	No impact	protected	private

Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Block parameter access	Inlined method	Inlined method	Inlined method	None	None
Internal data access	Inlined method	Inlined method	Inlined method	None	None
External I/O access	Inlined method	Inlined method	Inlined method	None	None
Generate destructor	No impact	No impact	No impact	Off	On
Use operator new for referenced model object registration	No impact	No impact	On	Off	Off
Generate preprocessor conditionals	No impact	No impact	No impact	No impact	Use local settings
Suppress error status in real-time model data structure	Off	No impact	On	On	Off
Create Simulink (S-Function) block	On	No impact	No impact	No impact	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Enable portable word sizes	On	On	Off	No impact	Off
MAT-file logging	On	No impact	Off	Off	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Code Style Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Parentheses level	Nominal (Optimize for readability)	Nominal (Optimize for readability)	Minimum (Rely on C/C++ operators for precedence)	Maximum (Specify precedence with parentheses)	Nominal (Optimize for readability)
Preserve operand order in expression	On	On	Off	On	Off
Preserve condition expression in if statement	On	On	Off	Off	Off

Mapping of Application Requirements to the Real-Time Workshop Pane: Code Style Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Convert if-elseif-else patterns to switch-case statements	No impact	Off	On	No impact	Off
Preserve extern keyword in function declarations	No impact	No impact	No impact	No impact	On

Mapping of Application Requirements to the Real-Time Workshop Pane: Templates Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Code templates: Source file (*.c) template	No impact	No impact	No impact	No impact	ert_code_template.cgt
Code templates: Header file (*.h) template	No impact	No impact	No impact	No impact	ert_code_template.cgt
Data templates: Source file (*.c) template	No impact	No impact	No impact	No impact	ert_code_template.cgt
Data templates: Header file (*.h) template	No impact	No impact	No impact	No impact	ert_code_template.cgt

Mapping of Application Requirements to the Real-Time Workshop Pane: Templates Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
File customization template	No impact	No impact	No impact	No impact	example_file_ - process.tlc
Generate an example main program	No impact	No impact	No impact	No impact	On
Target operating system	No impact	No impact	No impact	No impact	BareBoard-Example

Mapping of Application Requirements to the Real-Time Workshop Pane: Data Placement Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Data definition	No impact	Any valid value	No impact	No impact	Auto
Data definition filename	No impact	Any valid value	No impact	No impact	global.c
Data declaration	No impact	Any valid value	No impact	No impact	Auto
Data declaration filename	No impact	Any valid value	No impact	No impact	global.h
#include file delimiter	No impact	Any valid value	No impact	No impact	Auto
Module naming	No impact	Any valid value	No impact	No impact	Not specified

Mapping of Application Requirements to the Real-Time Workshop Pane: Data Placement Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Module name	No impact	Any valid value	No impact	No impact	' '
Signal display level	No impact	Any valid integer	No impact	No impact	10
Parameter tune level	No impact	Any valid integer	No impact	No impact	10

Mapping of Application Requirements to the Real-Time Workshop Pane: Data Type Replacement Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Replace data type names in the generated code	No impact	On	No impact	Off	Off
Replacement Name	No impact	Any valid string	No impact	' '	' '

Mapping of Application Requirements to the Real-Time Workshop Pane: Memory Sections Tab

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Package	No impact	No impact	No impact	No impact	---None---
Initialize/-Terminate	No impact	No impact	No impact	No impact	Default
Execution	No impact	No impact	No impact	No impact	Default
Constants	No impact	No impact	No impact	No impact	Default

Mapping of Application Requirements to the Real-Time Workshop Pane: Memory Sections Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Inputs/Outputs	No impact	No impact	No impact	No impact	Default
Internal data	No impact	No impact	No impact	No impact	Default
Parameters	No impact	No impact	No impact	No impact	Default
Validation results	No impact	No impact	No impact	No impact	Package and memory sections found.

Parameter Command-Line Information Summary

The following tables list Real-Time Workshop Embedded Coder parameters that you can use to tune model and target configurations. The table provides brief descriptions, valid values (bold type highlights defaults), and a mapping to Configuration Parameter dialog box equivalents. For descriptions of the panes and options in that dialog box, see Configuration Parameters in the Real-Time Workshop Embedded Coder documentation.

Use the `get_param` and `set_param` commands to retrieve and set the values of the parameters on the MATLAB command line or programmatically in scripts. The Real-Time Workshop Embedded Coder Configuration Wizard also provides buttons and scripts for customizing code generation.

For information about Simulink parameters, see “Configuration Parameters Dialog Box” in the Simulink documentation. For information about Real-Time Workshop parameters, see “Configuration Parameters for Simulink Models” in the Real-Time Workshop documentation. For information on using `get_param` and `set_param` to tune the parameters for various model configurations, see “Parameter Tuning by Using MATLAB Commands”. See “Using Configuration Wizard Blocks” in the Real-Time Workshop Embedded Coder documentation for information on using Configuration Wizard features.

Note Parameters that are specific to the ERT target or targets based on the ERT target, the Stateflow product, or the Simulink® Fixed Point™ product are marked with (ERT), (Stateflow), and (Simulink Fixed Point), respectively. To set the values of parameters marked with (ERT), you must specify an ERT or ERT-based target for your configuration set. Also, note that the default setting for a parameter might vary for different targets. Parameters marked with (ERT) are listed with ERT target defaults.

Command-Line Information: Optimization Pane

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
DataBitsets (Stateflow) off , on	Optimization > Use bitsets for storing Boolean data	Use bit sets for storing Boolean data.
InlinedParameterPlacement (ERT) Hierarchical , NonHierarchical	Optimization > Parameter structure	Specify how generated code stores global (tunable) parameters. Specify NonHierarchical to trade off modularity for efficiency.
BooleansAsBitfields (ERT) off , on	Optimization > Pack Boolean data into bitfields	Specify how generated code stores Boolean signals. If selected, Boolean signals are stored into one-bit bitfields in global block I/O structures or DWork vectors.
StrengthReduction (ERT) off , on	Optimization > Simplify array indexing	Suppress generation of code that replaces multiply operations when accessing arrays in a loop.
PassReuseOutputArgsAs (ERT) Structure reference , Individual arguments	Optimization > Pass reusable subsystem output as	Specify how a reusable subsystem passes outputs. Specify Individual arguments for efficiency.
NoFixptDivByZeroProtection (ERT) (Simulink Fixed Point) off , on	Optimization > Remove code that protects against division arithmetic exceptions	Suppress generation of code that guards against division by zero for fixed-point data.

Command-Line Information: Optimization Pane (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
OptimizeModelRefInitCode (ERT) off, on	Optimization > Optimize initialization code for model reference	Suppress generation of initialization code to accommodate the case where this model is referred to by a subsystem that resets its states when enabled. Select this option if the model will never be referred to by such a subsystem. The Simulink engine reports an error if this constraint is violated, in which case you can disable this optimization.
StateBitsets (Stateflow) off, on	Optimization > Use bitsets for storing state configuration	Use bit sets for storing state configuration.
ZeroExternalMemoryAtStartup (ERT) off, on	Optimization > Remove root level I/O zero initialization	Suppress code that initializes root-level I/O data structures to zero.
ZeroInternalMemoryAtStartup (ERT) off, on	Optimization > Remove internal data zero initialization	Suppress code that initializes global data structures (for example, block I/O data structures) to zero.

Command-Line Information: Real-Time Workshop Pane: General Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
IgnoreCustomStorageClasses (ERT) <i>string</i> - off, on	Real-Time Workshop > General > Ignore custom storage classes	Treat custom storage classes as 'Auto'.

Command-Line Information: Real-Time Workshop Pane: Report Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
GenerateTraceInfo (ERT) <i>string</i> - off , on	Real-Time Workshop > Report > Model-to-code	Includes model-to-code traceability support in the generated HTML report.
IncludeHyperlinkInReport (ERT) <i>string</i> - off , on	Real-Time Workshop > Report > Code-to-model	Link code segments to the corresponding object in the model. This option increases code generation time for large models.
GenerateTraceReport (ERT) <i>string</i> - off , on	Real-Time Workshop > Report > Eliminated / virtual blocks	Include summary of eliminated and virtual blocks in Code Generation report.
GenerateTraceReportSl (ERT) <i>string</i> - off , on	Real-Time Workshop > Report > Traceable Simulink blocks	Include summary of Simulink blocks in Code Generation report.
GenerateTraceReportSf (ERT) <i>string</i> - off , on	Real-Time Workshop > Report > Traceable Stateflow objects	Include summary of Stateflow objects in Code Generation report.
GenerateTraceReportEm1 (ERT) <i>string</i> - off , on	Real-Time Workshop > Report > Traceable Embedded MATLAB functions	Include summary of Embedded MATLAB functions in Code Generation report.

Command-Line Information: Real-Time Workshop Pane: Comments Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CustomCommentsFcn (ERT) <i>string</i> -	Real-Time Workshop > Comments > Custom comments function	Specify the filename of the M-function or TLC function that adds the custom comment.

Command-Line Information: Real-Time Workshop Pane: Comments Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
EnableCustomComments (ERT) <i>string</i> - off , on	Real-Time Workshop > Comments > Custom comments (MPT objects only)	Add a comment above a signal's or parameter's identifier in the generated file.
InsertBlockDesc (ERT) <i>string</i> - off , on	Real-Time Workshop > Comments > Simulink block descriptions	Insert the contents of the Description field from the Block Parameters dialog box into the generated code as a comment.
ReqsInCode (ERT) <i>string</i> - off , on	Real-Time Workshop > Comments > Requirements in block comments	Include specified requirements in the generated code as a comment.
SFDataObjDesc (ERT) <i>string</i> - off , on	Real-Time Workshop > Comments > Stateflow object descriptions	Insert Stateflow object descriptions into the generated code as a comment.
SimulinkDataObjDesc (ERT) <i>string</i> - off , on	Real-Time Workshop > Comments > Simulink data object descriptions	Insert Simulink data object descriptions into the generated code as comments.

Command-Line Information: Real-Time Workshop Pane: Symbols Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CustomSymbolStrBlkIO (ERT) <i>string</i> - rtb_ <i>\$N</i> \$M	Real-Time Workshop > Symbols > Local block output variables	Specify a symbol format rule for local block output variables. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$N - Name of object \$A - Data type acronym
CustomSymbolStrFcn (ERT) <i>string</i> - \$R <i>\$N</i> \$M \$F	Real-Time Workshop > Symbols > Subsystem methods	Specify a symbol format rule for subsystem methods. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$R - Root model name \$N - Name of object \$H - System hierarchy number \$F - Subsystem method name
CustomSymbolStrFcnArg(ERT) <i>string</i> - rtu_ <i>\$N</i> \$M or rty_ <i>\$N</i> \$M	Real-Time Workshop > Symbols > Subsystem method arguments	Specify a symbol format rule for subsystem method arguments. The rule can contain valid C identifier characters and the following macros: \$I — u if the argument is an input or y if the argument is an output \$M - Mangle \$N - Name of object

Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CustomSymbolStrField (ERT) <i>string</i> - \$N\$M	Real-Time Workshop > Symbols > Field name of global types	Specify a symbol format rule for field name of global types. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$N - Name of object \$H - System hierarchy number \$A - Data type acronym
CustomSymbolStrGlobalVar (ERT) <i>string</i> - \$R\$N\$M	Real-Time Workshop > Symbols > Global variables	Specify a symbol format rule for global variables. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$R - Root model name \$N - Name of object
CustomSymbolStrMacro (ERT) <i>string</i> - \$R\$N\$M	Real-Time Workshop > Symbols > Constant macros	Specify a symbol format rule for constant macros. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$R - Root model name \$N - Name of object

Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CustomSymbolStrTmpVar (ERT) string - \$N\$M	Real-Time Workshop > Symbols > Local temporary variables	Specify a symbol format rule for local temporary variables. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$R - Root model name \$N - Name of object
CustomSymbolStrType (ERT) string - \$N\$R\$M	Real-Time Workshop > Symbols > Global types	Specify a symbol format rule for global types. The rule can contain valid C identifier characters and the following macros: \$M - Mangle \$R - Root model name \$N - Name of object
DefineNamingFcn (ERT) string -	Real-Time Workshop > Symbols > #define naming > Custom M-function	Specify a custom M-function to control the naming of symbols with #define statements. You can set this parameter only if DefineNamingRule is set to Custom.
DefineNamingRule (ERT) string - None , UpperCase, LowerCase, Custom	Real-Time Workshop > Symbols > #define naming	Specify the rule that changes the spelling of all #define names.

Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
IncDataTypeInIds (ERT) off , on	Real-Time Workshop > Symbol > Include data type acronym in identifiers	Include acronyms that express data types in signal and work vector identifiers. For example, 'rtB.i32_signame' identifies a 32-bit integer block output signal named 'signame'.
IncHierarchyInIds (ERT) off , on	Real-Time Workshop > Symbols > Include system hierarchy number in identifiers	Include the system hierarchy number in variable identifiers. For example, 's3_' is the system hierarchy number in rtB.s3_signame for a block output signal named 'signame'. Including the system hierarchy number in identifiers improves the traceability of generated code. To locate the subsystem in which the identifier resides, type <code>hilite_system('<S3>')</code> at the MATLAB prompt. The argument specified with <code>hilite_system</code> requires an uppercase S.
InlinedPrmAccess (ERT) string - Literals , Macros	Real-Time Workshop > Symbols > Generate scalar inlined parameters as	Specify whether inlined parameters are coded as numeric constants or macros. Specify Macros for more efficient code.

Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
MangleLength (ERT) int - 1	Real-Time Workshop > Symbols > Minimum mangle length	Specify the minimum number of characters to be used for name mangling strings generated and applied to symbols to avoid name collisions. A larger value reduces the chance of identifier disturbance when you modify the model.
ParamNamingRule (ERT) string - None , UpperCase, LowerCase, Custom	Real-Time Workshop > Symbols > Parameter naming	Select a rule that changes spelling of all parameter names.
PrefixModelToSubsysFcnNames (ERT) off, on	Real-Time Workshop > Symbols > Prefix model name to global identifiers	Add the model name as a prefix to subsystem function names for all code formats. When appropriate for the code format, also add the model name as a prefix to top-level functions and data structures. This prevents compiler errors due to name clashes when combining multiple models.
SignalNamingRule (ERT) string - None , UpperCase, LowerCase, Custom	Real-Time Workshop > Symbols > Signal naming	Specify a rule the code generator is to use that changes spelling of all signal names.

Command-Line Information: Real-Time Workshop Pane: Interface Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CombineOutputUpdateFcns (ERT) string - off, on	Real-Time Workshop > Interface > Single output/update function	Generate a model's output and update routines into a single-step function.
ERTMaxMultiwordLength (ERT) int - 256	Real-Time Workshop > Interface > Maximum word length	Specify a maximum word length, in bits, for which the code generation process will generate system-defined multiword types into the file <code>rtwtypes.h</code> . Specifying 0 provides you complete control over type definitions for multiword data types in generated code.
ERTMultiwordTypeDef (ERT) string - System defined , User defined	Real-Time Workshop > Interface > Multiword type definitions	Specify whether to use system-defined or user-defined type definitions for multiword data types in generated code.
GenerateDestructor (ERT) string - off, on	Real-Time Workshop > Interface > Generate destructor	Generate a destructor for the model class in C++ (Encapsulated) model code.
GenerateErtSFunction (ERT) string - off , on	Real-Time Workshop > Interface > Create Simulink (S-Function) block	Wrap the generated code inside an S-Function block. This allows you to validate the generated code in a Simulink model.

Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
GenerateExternalIOAccess- Methods (ERT) string - None , Method, Inlined method	Real-Time Workshop > Interface > External I/O access	Specify whether to generate access methods for root-level I/O signals for the C++ (Encapsulated) model class.
GenerateInternalMember- AccessMethods (ERT) string - None , Method, Inlined method	Real-Time Workshop > Interface > Internal data access	Specify whether to generate access methods for internal data structures such as Block I/O, DWork vectors, Run-time model, Zero-crossings, and continuous states for the C++ (Encapsulated) model class.
GenerateParameterAccess- Methods (ERT) string - None , Method, Inlined method	Real-Time Workshop > Interface > Block parameter access	Specify whether to generate access methods for block parameters for the C++ (Encapsulated) model class.
GeneratePreprocessor- Conditionals (ERT) string - Use local settings , Enable all, Disable all	Real-Time Workshop > Interface > Generate preprocessor conditionals	Specify whether to generate preprocessor conditionals locally for each Model block containing variants or globally for all Model blocks in a model.
GRTInterface (ERT) string - off , on	Real-Time Workshop > Interface > GRT compatible call interface	Include a code interface (wrapper) that is compatible with the GRT target.

Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
IncludeMdlTerminateFcn (ERT) string - off, on	Real-Time Workshop > Interface > Terminate function required	Generate a terminate function for the model.
InternalMemberVisibility (ERT) string - public, private , protected	Real-Time Workshop > Interface > Internal data visibility	Specify whether to generate internal data structures such as Block I/O, DWork vectors, Run-time model, Zero-crossings, and continuous states as public, private, or protected data members of the C++ (Encapsulated) model class.
MatFileLogging (ERT) string - off , on	Real-Time Workshop > Interface > MAT-file logging	Generate code that logs data to a MAT-file.
MultiInstanceErrorCode (ERT) string - None, Warning, Error	Real-Time Workshop > Interface > Reusable code error diagnostic	Specify the error diagnostic behavior for cases when data defined in the model violates the requirements for generation of reusable code.
MultiInstanceERTCode (ERT) string - off , on	Real-Time Workshop > Interface > Reusable code error diagnostic	Specify the error diagnostic behavior for cases when data defined in the model violates the requirements for generation of reusable code.

Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
ParameterMemberVisibility (ERT) string - public, private , protected	Real-Time Workshop > Interface > Block parameter visibility	Specify whether to generate the block parameter structure as a public, private, or protected data member of the C++ (Encapsulated) model class.
PortableWordSizes (ERT) string - off , on	Real-Time Workshop > Interface > Enable portable word sizes	Specify that model code should be generated with conditional processing macros that allow the same generated source code files to be used both for software-in-the-loop (SIL) testing on the host platform and for production deployment on the target platform.
PurelyIntegerCode (ERT) string - off , on	Real-Time Workshop > Interface > floating-point numbers	Support floating-point data types in the generated code. This option is forced on when SupportNonInlinedSFcns is on.
RootIOFormat (ERT) string - Individual arguments , Structure reference	Real-Time Workshop > Interface > Pass root-level I/O as	Specify how the code generator is to pass root-level I/O data into a reusable function.

Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
SupportAbsoluteTime (ERT) string - off , on	Real-Time Workshop > Interface > absolute time	Support absolute time in the generated code. Blocks such as the Discrete Integrator might require absolute time.
SupportComplex (ERT) string - off , on	Real-Time Workshop > Interface > complex numbers	Support complex data types in the generated code.
SupportContinuousTime (ERT) string - off, on	Real-Time Workshop > Interface > continuous time	Support continuous time in the generated code. This allows blocks to be configured with a continuous sample time. Not available if SuppressErrorStatus is on.
SupportNonFinite (ERT) string - off , on	Real-Time Workshop > Interface > nonfinite numbers	Support nonfinite values (inf, nan, -inf) in the generated code. This option is forced on when SupportNonInlinedSFcns is on.
SupportVariableSizeSignals (ERT) string - off , on	Real-Time Workshop >Interface > variable-size signals	Specify whether to generate code for models that use variable-size signals.

Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
SuppressErrorStatus (ERT) string - off , on	Real-Time Workshop > Interface > Suppress error status in real-time model data structure	Remove the error status field of the real-time model data structure to preserve memory. When selected, SupportContinuousTime is cleared.
UseOperatorNewForModelRef-Registration (ERT) string - off , on	Real-Time Workshop > Interface > Use operator new for referenced model object registration	For a model containing Model blocks, specify whether generated code should use the operator new, during model object registration, to instantiate objects for referenced models configured with a C++ encapsulation interface.

Command-Line Information: Real-Time Workshop Pane: Code Style Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
ConvertIfToSwitch (ERT) string - off , on	Real-Time Workshop > Code Style > Convert if-elseif-else patterns to switch-case statements	Control whether if-elseif-else decision logic appears in generated code as switch-case statements.
ParenthesesLevel (ERT) string - Minimum, Nominal , Maximum	Real-Time Workshop > Code Style > Parentheses Level	Control existence of optional parentheses in generated code.
PreserveExpressionOrder (ERT) string - off , on	Real-Time Workshop > Code Style > Preserve operand order in expression	Control reordering of commutable expressions.

Command-Line Information: Real-Time Workshop Pane: Code Style Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
PreserveExternInFcnDecls (ERT) string - off, on	Real-Time Workshop > Code Style > Preserve extern keyword in function declarations	Control whether extern keyword appears in function declarations with external linkage in the generated code.
PreserveIfCondition (ERT) string - off, on	Real-Time Workshop > Code Style > Preserve condition expression in if statement	Control preservation of if statement conditions.

Command-Line Information: Real-Time Workshop Pane: Templates Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
ERTCustomFileTemplate (ERT) string - example_file_process.tlc	Real-Time Workshop > Templates > File customization template	Specify a TLC callback script for customizing the generated code.
ERTDataHdrFileTemplate (ERT) string - ert_code_template.cgt	Real-Time Workshop > Templates > Header file (*.h) template	Specify a template that organizes the generated data .h header files.
ERTDataSrcFileTemplate (ERT) string - ert_code_template.cgt	Real-Time Workshop > Templates > Source file (*.c or *.cpp) template	Specify a template that organizes the generated data .c source files.
ERTHdrFileBannerTemplate (ERT) string - ert_code_template.cgt	Real-Time Workshop > Templates > Header file (*.h) template	Specify a template that organizes the generated code .h header files.
ERTSrcFileBannerTemplate (ERT) string - ert_code_template.cgt	Real-Time Workshop > Templates > Source file (*.c or *.cpp) template	Specify a template that organizes the generated code .c or .cpp source files.

Command-Line Information: Real-Time Workshop Pane: Templates Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
GenerateSampleERTMain (ERT) string - off , on	Real-Time Workshop > Templates > Generate an example main program	Generate an example main program that demonstrates how to deploy the generated code. The program is written to the file <code>ert_main.c</code> or <code>ert_main.cpp</code> .
TargetOS (ERT) string - BareBoardExample , VxWorksExample	Real-Time Workshop > Templates > Target operating system	Specify the target operating system for the example main <code>ert_main.c</code> or <code>ert_main.cpp</code> . BareBoardExample is a generic example that assumes no operating system. VxWorksExample is tailored to the VxWorks ⁸ real-time operating system.

Command-Line Information: Real-Time Workshop Pane: Data Placement Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
DataDefinitionFile (ERT) string - global.c	Real-Time Workshop > Data Placement > Data definition filename	Specify the name of a single separate <code>.c</code> or <code>.cpp</code> file that contains global data definitions.
DataReferenceFile (ERT) string - global.h	Real-Time Workshop > Data Placement > Data declaration filename	Specify the name of a single separate <code>.c</code> or <code>.cpp</code> file that contains global data references.

8. VxWorks[®] is a registered trademark of Wind River[®] Systems, Inc.

Command-Line Information: Real-Time Workshop Pane: Data Placement Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
GlobalDataDefinition (ERT) string - Auto , InSourceFile, InSeparateSourceFile	Real-Time Workshop > Data Placement > Data definition	Select the .c or .cpp file where variables of global scope are defined.
GlobalDataReference (ERT) string - Auto , InSourceFile, InSeparateHeaderFile	Real-Time Workshop > Data Placement > Data declaration	Select the .h file where variables of global scope are declared (for example, <code>extern real_T globalvar;</code>).
IncludeFileDelimiter (ERT) string - Auto , UseQuote, UseBracket	Real-Time Workshop > Data Placement > #include file delimiter	Specify the delimiter to be used for all data objects that do not have a delimiter specified in the IncludeFile property.
ModuleName (ERT) string -	Real-Time Workshop > Data Placement > Module name	Specify the name of the module that owns this model.
ModuleNamingRule (ERT) string - Unspecified , SameAsModel, UserSpecified	Real-Time Workshop > Data Placement > Module naming	Specify the rule to be used for naming the module.
ParamTuneLevel (ERT) int - 10	Real-Time Workshop > Data Placement > Parameter tune level	Specify whether the code generator is to declare a parameter data object as tunable global data in the generated code.
SignalDisplayLevel (ERT) int - 10	Real-Time Workshop > Data Placement > Signal display level	Specify whether the code generator is to declare a signal data object as global data in the generated code.

Command-Line Information: Real-Time Workshop Pane: Data Type Replacement Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
EnableUserReplacementTypes (ERT) string - off , on	Real-Time Workshop > Data Type Replacement	Specify whether to replace built-in data type names with user-defined data type names in generated code.
ReplacementTypes (ERT) string -	Real-Time Workshop > Data Type Replacement > Data type names	Specify names to use for built-in data types in generated code.

Command-Line Information: Real-Time Workshop Pane: Memory Sections Tab

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
MemSecPackage (ERT) string - --- None ---, Simulink, mpt	Real-Time Workshop > Memory Sections > Package	Specify the package that contains the memory sections that you want to apply.
MemSecFuncInitTerm (ERT) string - Default , MemConst, MemVolatile, MemConstVolatile	Real-Time Workshop > Memory Sections > Initialize/Terminate	Apply memory sections to: <ul style="list-style-type: none"> • Initialize/Start functions • Terminate functions
MemSecFuncExecute (ERT) string - Default , MemConst, MemVolatile, MemConstVolatile	Real-Time Workshop > Memory Sections > Execution	Apply memory sections to: <ul style="list-style-type: none"> • Step functions • Run-time initialization functions • Derivative functions • Enable functions • Disable functions

Command-Line Information: Real-Time Workshop Pane: Memory Sections Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
MemSecDataConstants (ERT) string - Default , MemConst, MemVolatile, MemConstVolatile	Real-Time Workshop > Memory Sections > Constants	Apply memory sections to: <ul style="list-style-type: none"> • Constant parameters • Constant block I/O • Zero representation
MemSecDataIO (ERT) string - Default , MemConst, MemVolatile, MemConstVolatile	Real-Time Workshop > Memory Sections > Inputs/Outputs	Apply memory sections to: <ul style="list-style-type: none"> • Root inputs • Root outputs
MemSecDataInternal (ERT) string - Default , MemConst, MemVolatile, MemConstVolatile	Real-Time Workshop > Memory Sections > Internal data	Apply memory sections to: <ul style="list-style-type: none"> • Block I/O • DWork vectors • Run-time model • Zero-crossings
MemSecDataParameters (ERT) string - Default , MemConst, MemVolatile, MemConstVolatile	Real-Time Workshop > Memory Sections > Parameters	Apply memory sections to: <ul style="list-style-type: none"> • Parameters

Command-Line Information: Not in GUI

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CPPClassGenCompliant (ERT) string - off, on	Not available	Set in <code>SelectCallback</code> for a target to indicate whether the target supports the ability to generate and configure C++ encapsulation interfaces to model code. Default is <code>off</code> for custom and non-ERT targets and <code>on</code> for ERT (<code>ert.tlc</code>) targets. (For more information, see “Supporting C++ Encapsulation Interface Control” in the Real-Time Workshop Embedded Coder documentation.)
ERTFirstTimeCompliant (ERT) string - off, on	Not available	Set in <code>SelectCallback</code> for a target to indicate whether the target supports the ability to control inclusion of the <code>firstTime</code> argument in the <code>model_initialize</code> function generated for a Simulink model. Default is <code>off</code> for custom and non-ERT targets and <code>on</code> for ERT targets. (For more information, see “Supporting firstTime Argument Control” in the Real-Time Workshop Embedded Coder documentation.)

Command-Line Information: Not in GUI (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
<p>IncludeERTFirstTime (ERT) string - off, on</p> <hr/> <p>Note The value of IncludeERTFirstTime is meaningful only if the target configuration parameter ERTFirstTimeCompliant is set to on for your selected target.</p> <hr/>	Not available	Specify whether Real-Time Workshop Embedded Coder software is to include the <code>firstTime</code> argument in the <code>model_initialize</code> function generated for a Simulink model.
<p>ModelStepFunctionPrototype- ControlCompliant (ERT) string - off, on</p>	Not available	Set in <code>SelectCallback</code> for a target to indicate whether the target supports the ability to control the function prototypes of initialize and step functions that are generated for a Simulink model. Default is <code>off</code> for non-ERT targets and <code>on</code> for ERT targets. (For more information, see “Supporting C Function Prototype Control” in the Real-Time Workshop Embedded Coder documentation.)

A

- addAdditionalHeaderFile function 3-2
- addAdditionalIncludePath function 3-4
- addAdditionalLinkObj function 3-6
- addAdditionalLinkObjPath function 3-7
- addAdditionalSourceFile function 3-8
- addAdditionalSourcePath function 3-10
- addArgConf method 3-12
- addConceptualArg function 3-16
- addEntry function 3-19
- addIOConf AutosarInterface method 3-22
- arxml.importer class 3-27
- arxml.importer constructor 3-29
- attachToModel AutosarInterface method 3-30
- attachToModel method 3-31 to 3-32
- AUTOSAR 3-30 3-94 3-103 to 3-104 3-107 to 3-116 3-121 to 3-122 3-126 to 3-129 3-248 3-257 to 3-260
 - addIOConf 3-22
 - AutosarInterface 3-165
 - createCalibrationComponentObjects 3-67
 - createComponentAsModel 3-68
 - createComponentAsSubsystem 3-70
 - createOperationAsConfigurableSubsystems 3-73
 - getCalibrationComponentNames 3-89
 - getComponentName 3-91
 - getComponentNames 3-92
 - getDataTypePackageName 3-93
 - getDependencies 3-98
 - getExecutionPeriod 3-99
 - getFile 3-100
 - getImplementationName 3-102
 - getInterfacePackageName 3-105
 - getInternalBehaviorName 3-106
 - getPortDefaultConf 3-123
 - importer 3-27 3-29
 - runValidation 3-210
 - setComponentName 3-238
 - setDependencies 3-239
 - setFile 3-240

- setInitEventName 3-242
- setInitRunnableName 3-243
- setIOAutosarPortName 3-244
- setIODataAccessMode 3-245
- setIODataElement 3-246
- setIOInterfaceName 3-247
- setPeriodicEventName 3-252
- setPeriodicRunnableName 3-253
- syncWithModel 3-282

AUTOSAR Code Generation Options pane 6-91

AUTOSAR Configuration

- RTW.AutosarInterface 3-162

B

blocks

- Custom M-file 5-2
- Data Object Wizard 5-4
- ERT (optimized for fixed-point) 5-6
- ERT (optimized for floating-point) 5-8
- GRT (debug for fixed/floating-point) 5-10
- GRT (optimized for fixed/floating-point) 5-12
- Invoke AUTOSAR Server Operation 5-14
- Mode Switch for Invoke AUTOSAR Server Operation 5-16

C

C++ encapsulation interface control

- attachToModel 3-31
- getArgCategory 3-77
- getArgName 3-80
- getArgPosition 3-83
- getArgQualifier 3-86
- getClassName 3-90
- getDefaultConf 3-95
- getNumArgs 3-117
- getStepMethodName 3-130
- RTW.configSubsystemBuild 3-169
- RTW.getEncapsulationInterfaceSpecification 3-1

- RTW.ModelCPPArgsClass 3-192
- RTW.ModelCPPClass 3-196
- RTW.ModelCPPVoidClass 3-198
- runValidation 3-218 3-220
- setArgCategory 3-223
- setArgName 3-227
- setArgPosition 3-230
- setArgQualifier 3-233
- setClassName 3-236
- setStepMethodName 3-261
- Code Style pane 6-2
- configuration parameters
 - code generation 6-107
 - impacts of settings 6-95
 - pane
 - Preserve extern keyword in function declarations 6-12
 - Real-Time Workshop pane: Code Style 6-4
 - Real-Time Workshop pane: Data Placement 6-28
 - Real-Time Workshop pane: Data Type Replacement 6-47
 - Real-Time Workshop pane: Memory Sections 6-76
 - Real-Time Workshop pane: Templates 6-16
- Configuration Parameters dialog box
 - Code Style pane
 - Convert if-elseif-else patterns to switch-case statements 6-10
 - Parentheses level 6-5
 - Preserve condition expression in if statement 6-8
 - Preserve operand order in expression 6-7
 - Data Placement pane
 - Data declaration 6-33
 - Data declaration filename 6-35
 - Data definition 6-29
 - Data definition filename 6-31
 - #include file identifier 6-36
 - Module name 6-39
 - Module naming 6-37
 - Parameter tune level 6-43
 - Signal display level 6-41
- Data Type Replacement pane
 - boolean Replacement Name 6-66
 - char Replacement Name 6-72
 - double Replacement Name 6-50
 - int Replacement Name 6-68
 - int16 Replacement Name 6-56
 - int32 Replacement Name 6-54
 - int8 replacement name 6-58
 - Replace data type names in the generated code 6-48
 - single Replacement Name 6-52
 - uint Replacement Name 6-70
 - uint16 Replacement Name 6-62
 - uint32 Replacement Name 6-60
 - uint8 Replacement Name 6-64
- Memory Sections pane
 - Constants 6-82
 - Execution 6-81
 - Initialize/Terminate 6-80
 - Inputs/Outputs 6-84
 - Internal data 6-86
 - Package 6-77
 - Parameters 6-88
 - Refresh package list 6-79
 - Validation results 6-90
- Real-Time Workshop (AUTOSAR Code Generation Options) 6-92
 - AUTOSAR Schema Version 6-93
 - Configure AUTOSAR Interface 6-94
- Templates pane
 - code templates: Header file (*.h) template 6-18
 - code templates: Source file (*.c) template 6-17
 - data templates: Header file (*.h) template 6-20

- data templates: Source file (*.c)
 - template 6-19
- File customization template 6-21
- Generate an example main program 6-22
- Target operating system 6-24
- copyConceptualArgsToImplementation
 - function 3-48
- createAndAddConceptualArg function 3-50
- createAndAddImplementationArg function 3-57
- createAndSetCImplementationReturn
 - function 3-62
- createComponentAsSubsystem arxml.importer
 - method 3-70
- createOperationAsConfigurableSubsystems
 - arxml.importer method 3-73
- Custom M-file block 5-2

D

- Data Object Wizard block 5-4
- Data Placement pane 6-26
- Data Type Replacement pane 6-45

E

- ERT (optimized for fixed-point) block 5-6
- ERT (optimized for floating-point) block 5-8

F

- function prototype control
 - addArgConf 3-12
 - attachToModel 3-32
 - getArgCategory 3-79
 - getArgName 3-82
 - getArgPosition 3-85
 - getArgQualifier 3-88
 - getDefaultConf 3-97
 - getFunctionName 3-101
 - getNumArgs 3-119
 - getPreview 3-124

- RTW.configSubsystemBuild 3-169
- RTW.getFunctionSpecification 3-191
- RTW.ModelSpecificCPrototype 3-201
- runValidation 3-222
- setArgCategory 3-225
- setArgName 3-229
- setArgPosition 3-232
- setArgQualifier 3-235
- setFunctionName 3-241

G

- getArgCategory method 3-77 3-79
- getArgName method 3-80 3-82
- getArgPosition method 3-83 3-85
- getArgQualifier method 3-86 3-88
- getCalibrationComponentNames
 - arxml.importer method 3-89
- getClassName method 3-90
- getComponentName AutosarInterface
 - method 3-91
- getComponentNames arxml.importer
 - method 3-92
- getDataTypePackageName AutosarInterface
 - method 3-93
- getDefaultConf AutosarInterface method 3-94
- getDefaultConf method 3-95 3-97
- getDependencies arxml.importer method 3-98
- getExecutionPeriod AutosarInterface
 - method 3-99
- getFile arxml.importer method 3-100
- getFunctionName method 3-101
- getImplementationName AutosarInterface
 - method 3-102
- getInitEventName AutosarInterface
 - method 3-103
- getInitRunnableName AutosarInterface
 - method 3-104
- getInterfacePackageName AutosarInterface
 - method 3-105

- getInternalBehaviorName AutosarInterface
 - method 3-106
 - getIOAutosarPortName AutosarInterface
 - method 3-107
 - getIODataAccessMode AutosarInterface
 - method 3-108
 - getIODataElement AutosarInterface
 - method 3-109
 - getIOErrorStatusReceiver AutosarInterface
 - method 3-110
 - getIOInterfaceName AutosarInterface
 - method 3-111
 - getIOPortNumber AutosarInterface
 - method 3-112
 - getIOServiceInterface AutosarInterface
 - method 3-113
 - getIOServiceName AutosarInterface
 - method 3-114
 - getIOServiceOperation AutosarInterface
 - method 3-115
 - getIsServerOperation AutosarInterface
 - method 3-116
 - getNumArgs method 3-117 3-119
 - getPeriodicEventName AutosarInterface
 - method 3-121
 - getPeriodicRunnableName AutosarInterface
 - method 3-122
 - getPortDefaultConf AutosarInterface
 - method 3-123
 - getPreview method 3-124
 - getServerInterfaceName AutosarInterface
 - method 3-126
 - getServerOperationPrototype
 - AutosarInterface method 3-127
 - getServerPortName AutosarInterface
 - method 3-128
 - getServerType AutosarInterface method 3-129
 - getStepMethodName method 3-130
 - getTflArgFromString function 3-132
 - GRT (debug for fixed/floating-point) block 5-10
 - GRT (optimized for fixed/floating-point)
 - block 5-12
- I**
- Invoke AUTOSAR Server Operation block 5-14
- M**
- Memory Sections pane 6-74
 - Mode Switch for Invoke AUTOSAR Server
 - Operation block 5-16
 - model entry points
 - model_initialize 3-134
 - model_SetEventsForThisBaseStep 3-136
 - model_step 3-138
 - model_terminate 3-141
 - model_initialize function 3-134
 - model_output function 3-140
 - model_SetEventsForThisBaseStep
 - function 3-136
 - model_step function 3-138
 - model_terminate function 3-141
 - model_update function 3-140
 - models
 - parameters for configuring 6-107
- P**
- parameters
 - for configuring model code generation and targets 6-107
- R**
- registerCFunctionEntry function 3-144
 - registerCPromotableMacroEntry
 - function 3-147
 - RTW.AutosarInterface class 3-162
 - RTW.AutosarInterface constructor 3-165
 - RTW.configSubsystemBuild function 3-169

- rtw.connectivity.ComponentArgs 3-171
 - rtw.connectivity.Config 3-173
 - rtw.connectivity.ConfigRegistry 3-176
 - rtw.connectivity.Launcher 3-181
 - rtw.connectivity.MakefileBuilder 3-184
 - rtw.connectivity.RtIOStreamHostCommunicator 3-186
 - RTW.getEncapsulationInterfaceSpecification function 3-189
 - RTW.getFunctionSpecification function 3-191
 - RTW.ModelCPPArgsClass class 3-192
 - RTW.ModelCPPArgsClass constructor 3-195
 - RTW.ModelCPPClass class 3-196
 - RTW.ModelCPPVoidClass class 3-198
 - RTW.ModelCPPVoidClass constructor 3-200
 - RTW.ModelSpecificCPrototype class 3-201
 - RTW.ModelSpecificCPrototype constructor 3-204
 - rtw.pil.RtIOStreamApplicationFramework 3-206
 - runValidation AutosarInterface method 3-210
 - runValidation method 3-218 3-220 3-222
- S**
- setArgCategory method 3-223 3-225
 - setArgName method 3-227 3-229
 - setArgPosition method 3-230 3-232
 - setArgQualifier method 3-233 3-235
 - setClassName method 3-236
 - setComponentName AutosarInterface method 3-238
 - setDependencies arxml.importer method 3-239
 - setFile arxml.importer method 3-240
 - setFunctionName method 3-241
 - setInitEventName AutosarInterface method 3-242
 - setInitRunnableName AutosarInterface method 3-243
 - setIOAutosarPortName AutosarInterface method 3-244
 - setIODataAccessMode AutosarInterface method 3-245
 - setIODataElement AutosarInterface method 3-246
 - setIOInterfaceName AutosarInterface method 3-247
 - setIsServerOperation AutosarInterface method 3-248
 - setPeriodicEventName AutosarInterface method 3-252
 - setPeriodicRunnableName AutosarInterface method 3-253
 - setReservedIdentifiers function 3-254
 - setServerInterfaceName AutosarInterface method 3-257
 - setServerOperationPrototype AutosarInterface method 3-258
 - setServerPortName AutosarInterface method 3-259
 - setServerType AutosarInterface method 3-260
 - setStepMethodName method 3-261
 - setTfLCFunctionEntryParameters function 3-263
 - setTfLCOperationEntryParameters function 3-267
 - slConfigUIGetVal function 3-276
 - slConfigUISetEnabled function 3-278
 - slConfigUISetVal function 3-280
 - syncWithModel AutosarInterface method 3-282
- T**
- targets
 - parameters for configuring 6-107
 - Templates pane 6-14
 - TFL table creation
 - addAdditionalHeaderFile 3-2
 - addAdditionalIncludePath 3-4
 - addAdditionalLinkObj 3-6
 - addAdditionalLinkObjPath 3-7

addAdditionalSourceFile 3-8
addAdditionalSourcePath 3-10
addConceptualArg 3-16
addEntry 3-19
copyConceptualArgsToImplementation 3-48
createAndAddConceptualArg 3-50
createAndAddImplementationArg 3-57
createAndSetCImplementationReturn 3-62
getTflArgFromString 3-132
registerCFunctionEntry 3-144
registerCPromotableMacroEntry 3-147
setReservedIdentifiers 3-254
setTflCFunctionEntryParameters 3-263
setTflCOperationEntryParameters 3-267