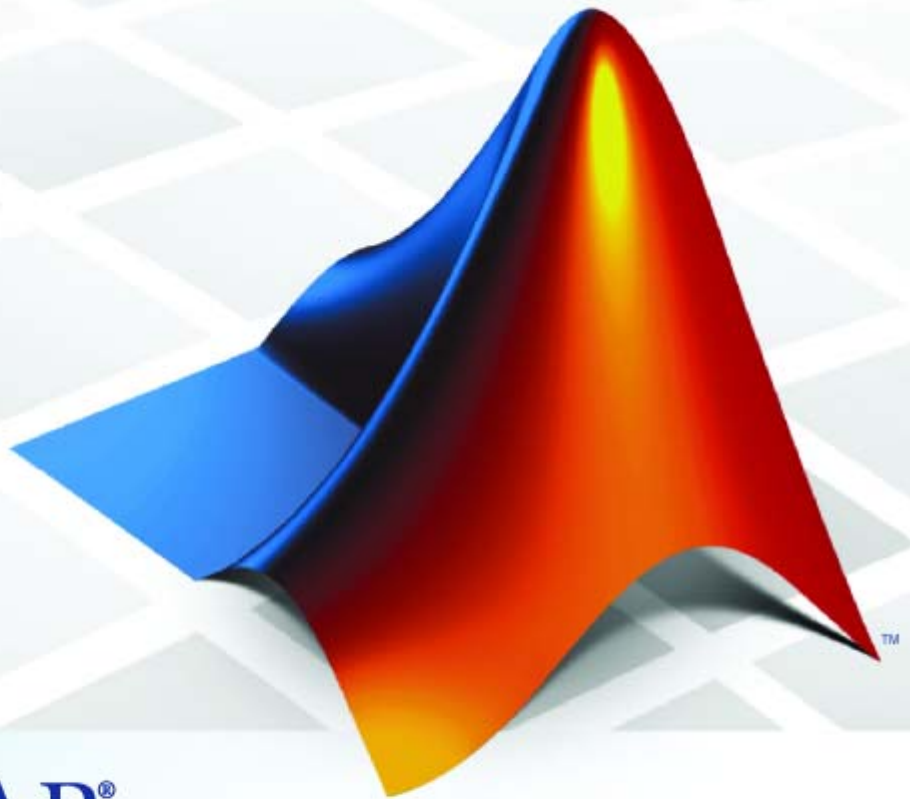


# Real-Time Workshop<sup>®</sup> Embedded Coder<sup>™</sup> 5 Reference



**MATLAB<sup>®</sup>**  
& **SIMULINK<sup>®</sup>**

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto: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](http://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](http://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

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)



## Function Reference

**1**

<b>AUTOSAR</b> .....	<b>1-2</b>
AUTOSAR Component Import .....	<b>1-2</b>
AUTOSAR Configuration .....	<b>1-3</b>
<b>C++ Encapsulation Interface Control</b> .....	<b>1-4</b>
<b>Function Prototype Control</b> .....	<b>1-7</b>
<b>Model Entry Points</b> .....	<b>1-9</b>
<b>System Target File Callback Interface</b> .....	<b>1-10</b>
<b>Target Function Library Table Creation</b> .....	<b>1-11</b>
<b>Processor-in-the-Loop</b> .....	<b>1-13</b>
Connectivity Configuration .....	<b>1-13</b>
Build .....	<b>1-13</b>
Execution Download, Start and Stop .....	<b>1-14</b>
Host and Target Communications .....	<b>1-14</b>
Host-Side Communications .....	<b>1-14</b>
Target-Side Communications .....	<b>1-14</b>

## Alphabetical List

---

2

## Block Reference

---

3

<b>Configuration Wizards</b> .....	3-2
<b>Module Packaging</b> .....	3-3

## Blocks — Alphabetical List

---

4

## Configuration Parameters

---

5

<b>Real-Time Workshop Pane: Code Style</b> .....	5-2
Real-Time Workshop: Code Style Tab Overview .....	5-4
Parentheses level .....	5-5
Preserve operand order in expression .....	5-7
Preserve condition expression in if statement .....	5-8
<b>Real-Time Workshop Pane: Templates</b> .....	5-10
Real-Time Workshop: Templates Tab Overview .....	5-12
Code templates: Source file (*.c) template .....	5-13
Code templates: Header file (*.h) template .....	5-14
Data templates: Source file (*.c) template .....	5-15
Data templates: Header file (*.h) template .....	5-16
File customization template .....	5-17
Generate an example main program .....	5-18
Target operating system .....	5-20
<b>Real-Time Workshop Pane: Data Placement</b> .....	5-22

Real-Time Workshop: Data Placement Tab Overview . . . .	5-24
Data definition . . . . .	5-25
Data definition filename . . . . .	5-27
Data declaration . . . . .	5-29
Data declaration filename . . . . .	5-31
#include file delimiter . . . . .	5-32
Module naming . . . . .	5-33
Module name . . . . .	5-35
Signal display level . . . . .	5-37
Parameter tune level . . . . .	5-39
<b>Real-Time Workshop Pane: Data Type Replacement . .</b>	<b>5-41</b>
Real-Time Workshop: Data Type Replacement Tab	
Overview . . . . .	5-43
Replace data type names in the generated code . . . . .	5-44
Replacement Name: double . . . . .	5-46
Replacement Name: single . . . . .	5-48
Replacement Name: int32 . . . . .	5-50
Replacement Name: int16 . . . . .	5-52
Replacement Name: int8 . . . . .	5-54
Replacement Name: uint32 . . . . .	5-56
Replacement Name: uint16 . . . . .	5-58
Replacement Name: uint8 . . . . .	5-60
Replacement Name: boolean . . . . .	5-62
Replacement Name: int . . . . .	5-64
Replacement Name: uint . . . . .	5-66
Replacement Name: char . . . . .	5-68
<b>Real-Time Workshop Pane: Memory Sections . . . . .</b>	<b>5-70</b>
Real-Time Workshop: Memory Sections Tab Overview . . .	5-72
Package . . . . .	5-73
Refresh package list . . . . .	5-75
Initialize/Terminate . . . . .	5-76
Execution . . . . .	5-77
Constants . . . . .	5-78
Inputs/Outputs . . . . .	5-80
Internal data . . . . .	5-82
Parameters . . . . .	5-84
Validation results . . . . .	5-86
<b>Real-Time Workshop Pane: AUTOSAR Code Generation</b>	
<b>Options . . . . .</b>	<b>5-87</b>

Real-Time Workshop: AUTOSAR Code Generation Options	
Tab Overview .....	<b>5-88</b>
Generate XML file from schema version .....	<b>5-89</b>
Configure AUTOSAR Interface .....	<b>5-90</b>
<b>Parameter Reference .....</b>	<b>5-91</b>
Recommended Settings Summary .....	<b>5-91</b>
Parameter Command-Line Information Summary .....	<b>5-102</b>

---

## **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. 1-4)	Control C++ encapsulation interfaces in generated code for ERT-based Simulink models
Function Prototype Control (p. 1-7)	Control step function prototypes in generated code for ERT-based Simulink models
Model Entry Points (p. 1-9)	Access entry points in generated code for ERT-based Simulink models
System Target File Callback Interface (p. 1-10)	Control Real-Time Workshop® configuration options in callbacks for ERT-based custom targets
Target Function Library Table Creation (p. 1-11)	Create function replacement tables that make up Real-Time Workshop target function libraries (TFLs)
Processor-in-the-Loop (p. 1-13)	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>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>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>importer</code>	Construct <code>arxml.importer</code> object
<code>setDependencies</code>	Set XML file dependencies
<code>setFile</code>	Set XML file name for <code>arxml.importer</code> object

## AUTOSAR Configuration

<code>addIOConf</code>	Add AUTOSAR I/O configuration to model
<code>attachToModel (AUTOSAR)</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 (AUTOSAR)</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
<code>getIOPortNumber</code>	Get I/O AUTOSAR port name
<code>getIOServiceInterface</code>	Get port I/O service interface
<code>getIOServiceName</code>	Get port I/O service name
<code>getIOServiceOperation</code>	Get port I/O service operation
<code>getPeriodicEventName</code>	Get periodic event name
<code>getPeriodicRunnableName</code>	Get periodic runnable name
<code>getPortDefaultConf</code>	Get port default configuration
<code>runValidation (AUTOSAR)</code>	Validate <code>RTW.AutosarInterface</code> 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
setPeriodicEventName	Set periodic event name
setPeriodicRunnableName	Set periodic runnable name
syncWithModel	Synchronize configuration with model

## **C++ Encapsulation Interface Control**

attachToModel (RTW.ModelCPPClass)	Attach model-specific C++ encapsulation interface to loaded ERT-based Simulink model
getArgCategory (RTW.ModelCPPArgsClass)	Get argument category for Simulink model port from model-specific C++ encapsulation interface
getArgName (RTW.ModelCPPArgsClass)	Get argument name for Simulink model port from model-specific C++ encapsulation interface
getArgPosition (RTW.ModelCPPArgsClass)	Get argument position for Simulink model port from model-specific C++ encapsulation interface
getArgQualifier (RTW.ModelCPPArgsClass)	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.getEncapsulationInterfaceSpecification</code>	Handle to model-specific C++ encapsulation interface control object
<code>RTW.ModelCPPArgsClass</code>	Control C++ encapsulation interfaces for models using I/O arguments style step method
<code>RTW.ModelCPPArgsClass</code>	Create C++ encapsulation interface object for configuring model class with I/O arguments style step method
<code>RTW.ModelCPPClass</code>	Control C++ encapsulation interfaces for models
<code>RTW.ModelCPPVoidClass</code>	Create C++ encapsulation interface object for configuring model class with void-void style step method
<code>RTW.ModelCPPVoidClass</code>	Control C++ encapsulation interfaces for models using 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

## Function Prototype Control

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

<code>slConfigUIGetVal</code>	Return current value for custom target configuration option
<code>slConfigUISetEnabled</code>	Enable or disable custom target configuration option
<code>slConfigUISetVal</code>	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. 1-13)	Define processor-in-the-loop (PIL) configuration
Build (p. 1-13)	Configure PIL build process
Execution Download, Start and Stop (p. 1-14)	Control downloading, starting and resetting PIL executable on target hardware
Host and Target Communications (p. 1-14)	Configure host-target communications
Host-Side Communications (p. 1-14)	Configure host-side communications channel and drivers
Target-Side Communications (p. 1-14)	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
--	--------------------------------------

# Alphabetical List

---

# addAdditionalHeaderFile

---

<b>Purpose</b>	Add additional header file to array of additional header files for TFL table entry
<b>Syntax</b>	<code>void addAdditionalHeaderFile(<i>hEntry</i>, <i>headerFile</i>)</code>
<b>Arguments</b>	<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.
<b>Description</b>	The <code>addAdditionalHeaderFile</code> function adds a specified additional header file to the array of additional header files for a TFL table entry.
<b>Example</b>	<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>
<b>See Also</b>	<code>addAdditionalIncludePath</code> , <code>addAdditionalSourceFile</code> , <code>addAdditionalSourcePath</code>



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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# addAdditionalIncludePath

---

<b>Purpose</b>	Add additional include path to array of additional include paths for TFL table entry
<b>Syntax</b>	<code>void addAdditionalIncludePath(<i>hEntry</i>, <i>path</i>)</code>
<b>Arguments</b>	<i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = RTW.TflCFunctionEntry or <i>hEntry</i> = RTW.TflCOperationEntry. <i>path</i> String specifying the full path to an additional header file.
<b>Description</b>	The <code>addAdditionalIncludePath</code> function adds a specified additional include path to the array of additional include paths for a TFL table entry.
<b>Example</b>	<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.TflCOperationEntry; . . . 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>
<b>See Also</b>	<code>addAdditionalHeaderFile</code> , <code>addAdditionalSourceFile</code> , <code>addAdditionalSourcePath</code>

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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# addAdditionalLinkObj

---

<b>Purpose</b>	Add additional link object to array of additional link objects for TFL table entry
<b>Syntax</b>	<code>void addAdditionalLinkObj(<i>hEntry</i>, <i>linkObj</i>)</code>
<b>Arguments</b>	<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.
<b>Description</b>	The <code>addAdditionalLinkObj</code> function adds a specified additional link object to the array of additional link objects for a TFL table entry.
<b>Example</b>	<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>
<b>See Also</b>	<code>addAdditionalLinkObjPath</code> “Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation “Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

<b>Purpose</b>	Add additional link object path to array of additional link object paths for TFL table entry
<b>Syntax</b>	<code>void addAdditionalLinkObjPath(<i>hEntry</i>, <i>path</i>)</code>
<b>Arguments</b>	<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>
<b>Description</b>	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.
<b>Example</b>	<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>
<b>See Also</b>	<p><code>addAdditionalLinkObj</code></p> <p>“Specifying Build Information for Function Replacements” in the Real-Time Workshop Embedded Coder documentation</p> <p>“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation</p>

# addAdditionalSourceFile

---

<b>Purpose</b>	Add additional source file to array of additional source files for TFL table entry
<b>Syntax</b>	<code>void addAdditionalSourceFile(<i>hEntry</i>, <i>sourceFile</i>)</code>
<b>Arguments</b>	<i>hEntry</i> Handle to a TFL table entry previously returned by <i>hEntry</i> = RTW.Tf1CFunctionEntry or <i>hEntry</i> = RTW.Tf1COperationEntry. <i>sourceFile</i> String specifying an additional source file.
<b>Description</b>	The <code>addAdditionalSourceFile</code> function adds a specified additional source file to the array of additional source files for a TFL table entry.
<b>Example</b>	<p>In the following example, the <code>addAdditionalSourceFile</code> function is used along with <code>addAdditionalHeaderFile</code>, <code>addAdditionalIncludePath</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>
<b>See Also</b>	<code>addAdditionalHeaderFile</code> , <code>addAdditionalIncludePath</code> , <code>addAdditionalSourcePath</code>

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

“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

“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.

<b>Inputs</b>	<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 Model Function Prototypes” — Explains how to configure model function prototypes in generated code

# 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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# addEntry

---

**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');
```

```
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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

## Purpose

Add AUTOSAR I/O configuration to model

## Syntax

```
autosarInterfaceObj.addIOConf(portName, dataAccessMode,  
    autosarPort, interfaceName, dataElement)  
addIOConf('SimulinkPortName', 'ImplicitReceive', 'PortName',  
    'InterfacePath', 'DataElement')  
addIOConf('SimulinkPortName', 'ExplicitReceive', 'PortName',  
    'InterfacePath', 'DataElement')  
addIOConf('SimulinkPortName', 'ImplicitSend', 'PortName',  
    'InterfacePath', 'DataElement')  
addIOConf('SimulinkPortName', 'ExplicitSend', 'PortName',  
    'InterfacePath', 'DataElement')  
addIOConf('SimulinkPortName', 'ErrorStatus',  
    'CorrespondingSimulinkPortName')  
addIOConf('SimulinkPortName', 'BasicSoftwarePort',  
    'ServiceName', 'ServiceOperation', 'ServiceInterfacePath')
```

## Description

addIOConf is a method of the class RTW.AutosarInterface.

You can designate inports and outports as data sender/receiver ports, error status receivers, or as access points to basic software.

For data sender/receiver ports, where dataAccessMode is either 'ImplicitSend', 'ImplicitReceive', 'ExplicitSend', or 'ExplicitReceive':

```
autosarInterfaceObj.addIOConf(portName, dataAccessMode,  
    autosarPort, interfaceName, dataElement)
```

adds an AUTOSAR I/O configuration, corresponding to the given port name, to autosarInterfaceObj, which is a model-specific RTW.AutosarInterface object. Note also that:

- portName is the inport/outport name (string).
- autosarPort is the Interface name (string).
- interfaceName is the Interface name (string).
- dataElement is the Interface name (string).



For data sender/receiver ports you can use any of the following:

- `addIOConf('SimulinkPortName', 'ImplicitReceive', 'PortName', 'InterfacePath', 'DataElement')`
- `addIOConf('SimulinkPortName', 'ExplicitReceive', 'PortName', 'InterfacePath', 'DataElement')`
- `addIOConf('SimulinkPortName', 'ImplicitSend', 'PortName', 'InterfacePath', 'DataElement')`
- `addIOConf('SimulinkPortName', 'ExplicitSend', 'PortName', 'InterfacePath', 'DataElement')`

For error status receivers (`dataAccessMode` is `'ErrorStatus'`) you can use the following syntax:

```
addIOConf('SimulinkPortName', 'ErrorStatus',  
'CorrespondingSimulinkPortName')
```

For access points to basic software (`dataAccessMode` is `'BasicSoftwarePort'`) you can use the following syntax:

```
addIOConf('SimulinkPortName', 'BasicSoftwarePort',  
'ServiceName', 'ServiceOperation', 'ServiceInterfacePath')
```

Defaults for data access mode parameters (other than `'ErrorStatus'`) are automatically created. These defaults are as follows:

- `ImplicitReceive/ExplicitReceive/ImplicitSend/ExplicitSend:`  
`'PortName' = 'SimulinkPortName'`
- `ImplicitReceive/ExplicitReceive/ImplicitSend/ExplicitSend:`  
`'InterfacePath' = 'SimulinkPortName'`
- `ImplicitReceive/ExplicitReceive/ImplicitSend/ExplicitSend:`  
`'DataElement' = 'SimulinkPortName'`
- `BasicSoftwarePort:` `'ServiceName' = 'SimulinkPortName'`
- `BasicSoftwarePort:` `'ServiceOperation' = 'SimulinkPortName'`

# addIOConf

---

- BasicSoftwarePort: 'ServiceInterfacePath' =  
'/AUTOSAR/Services/ServiceName'

**Purpose**

Attach RTW.AutosarInterface object to model

**Syntax**

```
autosarInterfaceObj.attachToModel(modelName)
```

**Description**

attachToModel is a method of the class RTW.AutosarInterface.

autosarInterfaceObj.attachToModel(modelName) attaches autosarInterfaceObj, an RTW.AutosarInterface object, to a loaded Simulink model with an ERT-based target.

modelName is the name of a loaded Simulink model to which the object is going to be attached (string).

# RTW.ModelCPPClass.attachToModel

---

**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
- “Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype.attachToModel

---

<b>Purpose</b>	Attach model-specific C function prototype to loaded ERT-based Simulink model				
<b>Syntax</b>	<code>attachToModel(obj, modelName)</code>				
<b>Description</b>	<code>attachToModel(obj, modelName)</code> attaches a model-specific C function prototype to a loaded ERT-based Simulink model.				
<b>Inputs</b>	<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.				
<b>Alternatives</b>	Click the <b>Configure Model Functions</b> button on the <b>Configuration Parameters &gt; Real-Time Workshop &gt; Interface</b> 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.				
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code				

# 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

“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.

*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'.

**IOType**  
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.

## 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(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',...
```

# createAndAddConceptualArg

---

```
'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.

## See Also

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

“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** `void createAndAddImplementationArg(hEntry,  
argType, varargin)`

**Arguments**

*hEntry*  
Handle to a TFL table entry previously returned by *hEntry* = RTW.TflCFunctionEntry or *hEntry* = RTW.TflCOperationEntry.

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

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

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

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

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

**IOType** String specifying the I/O type of the argument: 'RTW\_IO\_INPUT' for 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.

## 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.

# createAndAddImplementationArg

---

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' );
```

# createAndAddImplementationArg

---

```
% 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

“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.TflCFunctionEntry or *hEntry* = RTW.TflCOperationEntry.

*argType*  
String specifying the argument type to create:  
'RTW.TflArgNumeric' 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.

# createAndSetCImplementationReturn

---

## 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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# createCalibrationComponentObjects

---

**Purpose** Create Simulink calibration objects from AUTOSAR calibration component

**Syntax**

```
importerObj.createCalibrationComponentObjects(componentName)
[success] = createCalibrationComponentObjects(importerObj, componentName, CreateSimulinkObject, 'true')
[success] = createCalibrationComponentObjects(importerObj, componentName, Property1, Value1)
```

**Description** createCalibrationComponentObjects is a method of the class arxml.importer.

importerObj.createCalibrationComponentObjects(componentName) 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.

componentName is the absolute short name path of the calibration parameter component.

[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 the arxml.importer object importerObj.

[success] = createCalibrationComponentObjects(importerObj, componentName, Property1, Value1) can specify the following parameter/value pair:

CreateSimulinkObject

True or false (default is true). If it is true, this function creates the Simulink.AliasType and Simulink.NumericType corresponding to the AUTOSAR data types in the XML file.

Outputs:

success

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



# createCalibrationComponentObjects

---

## Example

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

## See Also

```
createComponentAsSubsystem; createComponentAsModel;  
getCalibrationComponentNames; getDependencies;  
setDependencies; setFile
```

# 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** createComponentAsModel is a method of the class `arxml.importer` that creates an AUTOSAR atomic software component as a Simulink model.

```
[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. ComponentName is the absolute  
Short-Name path of the atomic software component .
```

```
[modelH, success] =  
importerObj.createComponentAsModel(ComponentName,  
Property1, Value1, Property2, Value2...) creates the Simulink  
model and can specify the following parameter/value pairs:
```

**CreateSimulinkObject**

True or false (default is true). If it is true, this function creates the `Simulink.AliasType` and `Simulink.NumericType` corresponding to the AUTOSAR data types in the XML file.

**NameConflictAction**

'overwrite' or 'makenameunique' or 'error' (default is 'overwrite'). Use this parameter to control the action to take when a Simulink model with the same name as the component already exists.

**AutoSave**

True or false (default is false). If it is true, this function automatically saves the generated Simulink model.

Outputs:

`modelH`  
Model handle.

`success`  
True if this function is successful. Otherwise, it is false.

## Example

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

## See Also

`createComponentAsSubsystem`; `getComponentNames`; `getDependencies`;  
`setDependencies`; `setFile`

# createComponentAsSubsystem

---

## Purpose

Create AUTOSAR atomic software component as Simulink atomic subsystem

## Syntax

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

## Description

`createComponentAsSubsystem` is a method of the class `arxml.importer` that creates an AUTOSAR atomic software component as a Simulink atomic subsystem.

```
[subsysH, 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. ComponentName is the absolute  
 short name path of the atomic software component .
```

```
[subsysH, success] =  
 importerObj.createComponentAsSubsystem(ComponentName,  
 Property1, Value1, Property2, Value2...) creates the Simulink  
 subsystem and can specify the following parameter/value pairs:
```

### CreateSimulinkObject

Boolean value, true or false (default is true). If it is true, this function creates the `Simulink.AliasType` and `Simulink.NumericType` corresponding to the AUTOSAR data types in the XML file.

### NameConflictAction

'overwrite' or 'makenameunique' or 'error' (default is 'overwrite'). Use this parameter to control the action to take when a Simulink model with the same name as the component already exists.

## AutoSave

Boolean value, true or false (default is false). If it is true, this function automatically saves the generated Simulink model.

## Outputs:

### subsysH

Subsystem handle.

### success

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

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**.

## Example

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

## See Also

```
createComponentAsModel; getComponentNames; getDependencies;  
getFile; setDependencies; setFile
```

# RTW.ModelCPPArgsClass.getArgCategory

---

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

**Syntax** `category = getArgCategory(obj, portName)`

**Description** `category = getArgCategory(obj, portName)` gets the category — 'Value', 'Pointer', or 'Reference' — 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>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.

## **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

“Generating and Controlling C++ Encapsulation 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** `category = getArgCategory(obj, portName)`

**Description** `category = getArgCategory(obj, portName)` 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**

<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>category</code>	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 Model 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

## RTW.ModelCPPArgsClass.getArgName

---

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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype.getArgName

---

<b>Purpose</b>	Get argument name for Simulink model port from model-specific C function prototype	
<b>Syntax</b>	<code>argName = getArgName(obj, portName)</code>	
<b>Description</b>	<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.	
<b>Inputs</b>	<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.
<b>Outputs</b>	<code>argName</code>	String specifying the argument name for the specified Simulink model port.
<b>Alternatives</b>	Click the <b>Get Default Configuration</b> 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.	
<b>See Also</b>	“Controlling Model 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** `position = getArgPosition(obj, portName)`

**Description** `position = getArgPosition(obj, portName)` 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.

## **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

“Generating and Controlling C++ Encapsulation 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** `position = getArgPosition(obj, portName)`

**Description** `position = getArgPosition(obj, portName)` gets the position — 1 for first, 2 for second, etc. — of the argument corresponding to a specified Simulink model inport or output 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 output 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 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 Model Function Prototypes” — Explains how to configure model function prototypes in generated code

# RTW.ModelCPPArgsClass.getArgQualifier

---

<b>Purpose</b>	Get argument type qualifier for Simulink model port from model-specific C++ encapsulation interface	
<b>Syntax</b>	<code>qualifier = getArgQualifier(obj, portName)</code>	
<b>Description</b>	<code>qualifier = getArgQualifier(obj, portName)</code> 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.	
<b>Inputs</b>	<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.
<b>Outputs</b>	<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', 'const * const', or 'const &' — for the specified Simulink model port.
<b>Alternatives</b>	To view argument qualifiers in the Simulink Configuration Parameters graphical user interface, go to the <b>Interface</b> pane and click the <b>Configure C++ Encapsulation Interface</b> 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 <b>Get Default Configuration</b> 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.	

# RTW.ModelCPPArgsClass.getArgQualifier

---

## **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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation



# RTW.ModelSpecificCPrototype.getArgQualifier

---

<b>Purpose</b>	Get argument type qualifier for Simulink model port from model-specific C function prototype				
<b>Syntax</b>	<code>qualifier = getArgQualifier(obj, portName)</code>				
<b>Description</b>	<code>qualifier = getArgQualifier(obj, portName)</code> 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.				
<b>Inputs</b>	<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.				
<b>Outputs</b>	<table><tr><td><code>qualifier</code></td><td>String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const' — for the specified Simulink model port.</td></tr></table>	<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const' — for the specified Simulink model port.		
<code>qualifier</code>	String specifying the argument type qualifier — 'none', 'const', 'const *', or 'const * const' — for the specified Simulink model port.				
<b>Alternatives</b>	Click the <b>Get Default Configuration</b> 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.				
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code				

# getCalibrationComponentNames

---

**Purpose** Get calibration component names

**Syntax** `calibrationComponentNames = importerObj.getCalibrationComponentNames`

**Description** `getCalibrationComponentNames` is a method of the class `arxml.importer`.

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

`componentNames` is a cell array of strings in which each element is the absolute short name path of the corresponding calibration parameters component :

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

**See Also** `createCalibrationComponentObjects`;

<b>Purpose</b>	Get class name from model-specific C++ encapsulation interface	
<b>Syntax</b>	<code>clsName = getClassName(obj)</code>	
<b>Description</b>	<code>clsName = getClassName(obj)</code> gets the name of the class described by the specified model-specific C++ encapsulation interface.	
<b>Inputs</b>	<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>
<b>Outputs</b>	<code>clsName</code>	A string specifying the name of the class described by the specified model-specific C++ encapsulation interface.
<b>Alternatives</b>	To view the model class name in the Simulink Configuration Parameters graphical user interface, go to the <b>Interface</b> pane and click the <b>Configure C++ Encapsulation Interface</b> 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.	
<b>See Also</b>	“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 “Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation	

# GetComponentName

---

**Purpose** Get XML component name

**Syntax** `componentName = autosarInterfaceObj.GetComponentName`

**Description** `GetComponentName` is a method of the class `RTW.AutosarInterface`.  
`componentName = autosarInterfaceObj.GetComponentName` gets the XML component name of the `RTW.AutosarInterface` object specified by `autosarInterfaceObj`.  
`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.  
`componentName` is the name of the XML component specified by `autosarInterfaceObj`.

**See Also** `setComponentName`

**Purpose**

Get atomic software component names

**Syntax**

```
componentNames = importerObj.GetComponentNames
```

**Description**

`GetComponentNames` is a method of the class `arxml.importer`.

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

`componentNames` is a 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 '
```

---

**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.

---

**See Also**

`createComponentAsModel`; `createComponentAsSubsystem`

# getDataTypeName

---

**Purpose** Get XML data type package name

**Syntax** `dataTypeName = autosarInterfaceObj.getDataTypeName`

**Description** `getDataTypeName` is a method of the class `RTW.AutosarInterface`.

`dataTypeName = getDataTypeName(autosarInterfaceObj)` gets the XML data type package name of the specified `RTW.AutosarInterface` object, `autosarInterfaceObj`.

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

`dataTypeName` is the name of the data type package specified by `autosarInterfaceObj`.

**Purpose** Get default configuration

**Syntax** `autosarInterfaceObj.getDefaultConf`

**Description** `getDefaultConf` is a method of the class `RTW.AutosarInterface`.  
`autosarInterfaceObj.getDefaultConf` gets the model's default configuration for this `RTW.AutosarInterface` object, `autosarInterfaceObj`, using the information from the model to which the object has already been 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.

# RTW.ModelCPPClass.getDefaultConf

---

<b>Purpose</b>	Get default configuration information for model-specific C++ encapsulation interface from Simulink model		
<b>Syntax</b>	<code>getDefaultConf(obj)</code>		
<b>Description</b>	<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>		
<b>Inputs</b>	<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> .		
<b>Alternatives</b>	To view C++ encapsulation interface default configuration information in the Simulink Configuration Parameters graphical user interface, go to the <b>Interface</b> pane and click the <b>Configure C++ Encapsulation Interface</b> 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 <b>Get Default Configuration</b> 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.		



## **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

“Generating and Controlling C++ Encapsulation 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** `getDefaultConf(obj)`

**Description** `getDefaultConf(obj)` 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.

Before calling this function, you must call `attachToModel`, to attach the function prototype 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> .
------------	--

**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 Model Function Prototypes” — Explains how to configure model function prototypes in generated code

<b>Purpose</b>	Get list of XML dependency files
<b>Syntax</b>	<code>Dependencies = importerObj.getDependencies()</code>
<b>Description</b>	<p><code>getDependencies</code> is a method of the class <code>arxml.importer</code>.</p> <p><code>Dependencies = importerObj.getDependencies()</code> returns the list of XML dependency files associated with the <code>arxml.importer</code> object, <code>importerObj</code>.</p> <p><code>Dependencies</code> is a cell array of strings.</p>
<b>See Also</b>	<code>getFile</code> ; <code>setDependencies</code> ; <code>setFile</code>

# getExecutionPeriod

---

**Purpose** Get runnable execution period

**Syntax** EP = autosarInterfaceObj.getExecutionPeriod

**Description** getExecutionPeriod is a method of the class RTW.AutosarInterface.  
EP = autosarInterfaceObj.getExecutionPeriod gets the execution period in the configuration.  
autosarInterfaceObj is a model-specific RTW.AutosarInterface object.

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

**Syntax** `filename = importerObj.getFile`

**Description** `getFile` is a method of the class `arxml.importer`.  
`filename = importerObj.getFile` returns the XML filename associated with the `arxml.importer` object, `importerObj`.

**See Also** `getDependencies`; `setDependencies`; `setFile`

# RTW.ModelSpecificCPrototype.getFunctionName

---

<b>Purpose</b>	Get function name from model-specific C function prototype	
<b>Syntax</b>	<code>fcnName = getFunctionName(obj, fcnType)</code>	
<b>Description</b>	<code>fcnName = getFunctionName(obj, fcnType)</code> gets the name of the step or initialize function described by the specified model-specific C function prototype.	
<b>Inputs</b>	<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.
<b>Outputs</b>	<code>fcnName</code>	A string specifying the name of the function described by the specified model-specific C function prototype.
<b>Alternatives</b>	Click the <b>Get Default Configuration</b> 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.	
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code	

**Purpose** Get XML implementation name

**Syntax** `implementationName = autosarInterfaceObj.getImplementationName`

**Description** `getImplementationName` is a method of the class `RTW.AutosarInterface`.

`implementationName = autosarInterfaceObj.getImplementationName` gets the XML implementation name of the `RTW.AutosarInterface` object specified by `autosarInterfaceObj`.

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

`implementationName` is the name of the implementation specified by `autosarInterfaceObj`.

# getInitEventName

---

**Purpose** Get initial event name

**Syntax** `initEventName = autosarInterfaceObj.getInitEventName`

**Description** `getInitEventName` is a method of the class `RTW.AutosarInterface`.  
`initEventName = autosarInterfaceObj.getInitEventName` gets the initial event name of the `RTW.AutosarInterface` object specified by `autosarInterfaceObj`.  
`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.  
`initEventName` is the name of the initial event specified by `autosarInterfaceObj`.

**See Also** `setInitEventName`



<b>Purpose</b>	Get initial runnable name
<b>Syntax</b>	<code>initRunnableName = autosarInterfaceObj.getInitRunnableName</code>
<b>Description</b>	<p><code>attachToModel</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>initRunnableName = autosarInterfaceObj.getInitRunnableName</code> gets the initial runnable name of the <code>RTW.AutosarInterface</code> object.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>initRunnableName</code> is the name of the initial runnable specified by <code>autosarInterfaceObj</code>.</p>
<b>See Also</b>	<code>setInitRunnableName</code>

# getInterfacePackageName

---

**Purpose** Get XML interface package name

**Syntax** `interfacePkgName = autosarInterfaceObj.getInterfacePackageName`

**Description** `getInterfacePackageName` is a method of the class `RTW.AutosarInterface`.

`interfacePkgName = autosarInterfaceObj.getInterfacePackageName` gets the XML interface package name of the `RTW.AutosarInterface` object specified by `autosarInterfaceObj`.

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

`interfacePkgName` is the name of the interface package specified by `autosarInterfaceObj`.

**Purpose** Get XML internal behavior name

**Syntax** `internalBehaviorName = autosarInterfaceObj.getInternalBehaviorName`

**Description** `getInternalBehaviorName` is a method of the class `RTW.AutosarInterface`.

`internalBehaviorName = autosarInterfaceObj.getInternalBehaviorName` gets the XML internal behavior name of the `RTW.AutosarInterface` object specified by `autosarInterfaceObj`.

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

`internalBehaviorName` is the name of the internal behavior specified by `autosarInterfaceObj`.

# getIOAutosarPortName

---

<b>Purpose</b>	Get I/O AUTOSAR port name
<b>Syntax</b>	<code>ioAutosarName = autosarInterfaceObj.getIOAutosarPortName(portName)</code>
<b>Description</b>	<p><code>getIOAutosarPortName</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>ioAutosarName = autosarInterfaceObj.getIOAutosarPortName(portName)</code> gets the I/O AUTOSAR port name in the configuration for the port corresponding to the given port name.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>ioAutosarName</code> is the AUTOSAR port name of the given <code>portName</code>.</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>
<b>See Also</b>	<code>setIOAutosarPortName</code>

<b>Purpose</b>	Get I/O data access mode
<b>Syntax</b>	<code>dataAccessMode = autosarInterfaceObj.getIODataAccessMode(portName)</code>
<b>Description</b>	<p><code>getIODataAccessMode</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <pre>dataAccessMode = autosarInterfaceObj.getIODataAccessMode(portName)</pre> <p>gets the data access mode of the I/O corresponding to the port as specified by the port name, for <code>autosarInterfaceObj</code>, a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>dataAccessMode</code> returns the data access mode of the given port, one of 'ImplicitSend', 'ImplicitReceive', or 'ExplicitSend', 'ExplicitReceive' (string).</p>
<b>See Also</b>	<code>setIODataAccessMode</code>

# getIODataElement

---

<b>Purpose</b>	Get I/O data element name
<b>Syntax</b>	<code>ioDataElement = autosarInterfaceObj.getIODataElement(portName)</code>
<b>Description</b>	<p><code>getIODataElement</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>ioDataElement = autosarInterfaceObj.getIODataElement(portName)</code> gets the I/O data element name in the configuration for the port corresponding to the given port name.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>ioDataElement</code> is the data element of the given <code>portName</code> (string).</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>
<b>See Also</b>	<code>setIODataElement</code>

<b>Purpose</b>	Get receiver port name
<b>Syntax</b>	<code>ESR = autosarInterfaceObj.getIOErrorStatusReceiver(PortName)</code>
<b>Description</b>	<p><code>getIOErrorStatusReceiver</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>ESR = autosarInterfaceObj.getIOErrorStatusReceiver(PortName)</code> gets the receiver port name in the configuration for the port corresponding to the given <code>PortName</code>.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>PortName</code> is the inport/outport name (string).</p> <p><code>ESR</code> is the receiver port name of the given <code>PortName</code>.</p>

# getIOInterfaceName

---

<b>Purpose</b>	Get I/O interface name
<b>Syntax</b>	<code>ioInterfaceName = autosarInterfaceObj.getIOInterfaceName(portName)</code>
<b>Description</b>	<p><code>getIOInterfaceName</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>ioInterfaceName = autosarInterfaceObj.getIOInterfaceName(portName)</code> gets the I/O interface name in the configuration for the port corresponding to the given port name.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>ioInterfaceName</code> is the I/O interface name of the given <code>portName</code>.</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>
<b>See Also</b>	<code>setIOInterfaceName</code>



**Purpose** Get I/O AUTOSAR port name

**Syntax** `IOPortNumber= autosarInterfaceObj.getIOPortNumber(PortName)`

**Description** `getIOPortNumber` is a method of the class `RTW.AutosarInterface`.  
`IOPortNumber= autosarInterfaceObj.getIOPortNumber(PortName)` gets the I/O AUTOSAR port name in the configuration for the port corresponding to the given `PortName`.  
`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.  
`PortName` is the inport/outport name (string).  
`IOPortNumber` is the port number of the given `PortName`.

# getIOServiceInterface

---

**Purpose**

Get port I/O service interface

**Syntax**

```
SI = autosarInterfaceObj.getIOServiceInterface(PortName)
```

**Description**

getIOServiceInterface is a method of the class RTW.AutosarInterface.

SI = autosarInterfaceObj.getIOServiceInterface(PortName) gets the I/O service interface in the configuration for the port corresponding to the given PortName.

autosarInterfaceObj is a model-specific RTW.AutosarInterface object.

PortName is the inport/outport name (string).

SI is the I/O service interface of the given PortName.

**Purpose** Get port I/O service name

**Syntax** SN = autosarInterfaceObj.getIOServiceName(PortName)

**Description** getIOServiceName is a method of the class RTW.AutosarInterface.

SN = autosarInterfaceObj.getIOServiceName(PortName) gets the I/O service name in the configuration for the port corresponding to the given PortName.

autosarInterfaceObj is a model-specific RTW.AutosarInterface object.

PortName is the inport/outport name (string).

SN is the I/O service name of the given PortName.

# getIOServiceOperation

---

**Purpose** Get port I/O service operation

**Syntax** `S0 = autosarInterfaceObj.getIOServiceOperation(PortName)`

**Description** `getIOServiceOperation` is a method of the class `RTW.AutosarInterface`.

`S0 = autosarInterfaceObj.getIOServiceOperation(PortName)` gets the I/O service operation in the configuration for the port corresponding to the given `PortName`.

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

`PortName` is the inport/outport name (string).

`S0` is the I/O service operation of the given `PortName`.

<b>Purpose</b>	Get number of step method arguments from model-specific C++ encapsulation interface		
<b>Syntax</b>	<code>num = getNumArgs(obj)</code>		
<b>Description</b>	<code>num = getNumArgs(obj)</code> gets the number of arguments for the step method described by the specified model-specific C++ encapsulation interface.		
<b>Inputs</b>	<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>		
<b>Outputs</b>	<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.		
<b>Alternatives</b>	To view the number of step method arguments in the Simulink Configuration Parameters graphical user interface, go to the <b>Interface</b> pane and click the <b>Configure C++ Encapsulation Interface</b> 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 <b>Get Default Configuration</b> 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.		
<b>See Also</b>	“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.getNumArgs**

---

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype.getNumArgs

---

<b>Purpose</b>	Get number of function arguments from model-specific C function prototype		
<b>Syntax</b>	<code>num = getNumArgs(obj)</code>		
<b>Description</b>	<code>num = getNumArgs(obj)</code> gets the number of function arguments for the function described by the specified model-specific C function prototype.		
<b>Inputs</b>	<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> .		
<b>Outputs</b>	<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.		
<b>Alternatives</b>	Click the <b>Get Default Configuration</b> button in the Model Interface dialog box to get arguments. See “Model Specific C Prototypes View” in the Real-Time Workshop Embedded Coder documentation.		
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code		

# getPeriodicEventName

---

<b>Purpose</b>	Get periodic event name
<b>Syntax</b>	<code>periodicEventName = autosarInterfaceObj.getPeriodicEventName</code>
<b>Description</b>	<p><code>getPeriodicEventName</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>periodicEventName = autosarInterfaceObj.getPeriodicEventName</code> gets the periodic event name specified by <code>autosarInterfaceObj</code>, the <code>RTW.AutosarInterface</code> object.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>periodicEventName</code> is the name of the periodic event specified by <code>autosarInterfaceObj</code>.</p> <p>For multiple runnables, use the <code>Children</code> property to access each individual runnable after building or GUI update, for example:</p> <pre>autosarInterfaceObj.Children(1).getPeriodicEventName()</pre>
<b>See Also</b>	<code>setPeriodicEventName</code>



<b>Purpose</b>	Get periodic runnable name
<b>Syntax</b>	<code>periodicRunnableName = autosarInterfaceObj.getPeriodicRunnableName</code>
<b>Description</b>	<p><code>getPeriodicRunnableName</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>periodicRunnableName = autosarInterfaceObj.getPeriodicRunnableName</code> gets the name of the periodic runnable specified in <code>autosarInterfaceObj</code>, an <code>RTW.AutosarInterface</code> object.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>periodicRunnableName</code> is the name of the periodic runnable specified by <code>autosarInterfaceObj</code>.</p> <p>For multiple runnables, use the <code>Children</code> property to access each individual runnable after building or GUI update, for example:</p> <pre>autosarInterfaceObj.Children(1).getPeriodicRunnableName()</pre>
<b>See Also</b>	<code>setPeriodicRunnableName</code>

# getPortDefaultConf

---

**Purpose** Get port default configuration

**Syntax** `[autosarPort, interfaceName, dataElement, dataAccessMode]=autosarInterfaceObj.getPortDefaultConf(portH)`

**Description** `getPortDefaultConf` is a method of the class `RTW.AutosarInterface`.

`[autosarPort, interfaceName, dataElement, mode]=autosarInterfaceObj.getPortDefaultConf(portH)` gets the port's default configuration for `autosarInterfaceObj`, the `RTW.AutosarInterface` object.

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

`autosarPort` is the AUTOSAR port name of the given `portH` (string).

`interfaceName` is the I/O interface name of the given `portH` (string).

`dataElement` is the data element of the given `portH` (string).

`dataAccessMode` returns the data access mode of the given port, one of 'ImplicitSend', 'ImplicitReceive', or 'ExplicitSend', 'ExplicitReceive' (string).

# RTW.ModelSpecificCPrototype.getPreview

---

<b>Purpose</b>	Get model-specific C function prototype code preview				
<b>Syntax</b>	<code>preview = getPreview(obj, fcnType)</code>				
<b>Description</b>	<code>preview = getPreview(obj, fcnType)</code> gets the model-specific C function prototype code preview.				
<b>Inputs</b>	<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(modeName)</code>.</td></tr><tr><td><code>fcnType</code></td><td>Optional. String specifying which function to preview. Valid strings are 'step' and 'init'. If <code>fcnType</code> is not specified, previews the step function.</td></tr></table>	<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modeName)</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.
<code>obj</code>	Handle to a model-specific C prototype function control object previously returned by <code>obj = RTW.getFunctionSpecification(modeName)</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.				
<b>Outputs</b>	<table><tr><td><code>preview</code></td><td>String specifying the function prototype for the step or initialization function.</td></tr></table>	<code>preview</code>	String specifying the function prototype for the step or initialization function.		
<code>preview</code>	String specifying the function prototype for the step or initialization function.				
<b>Alternatives</b>	Use the <b>Step function preview</b> 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.				
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code				

# 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  
“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

**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;
```

## getTflArgFromString

---

```
.  
. .  
. .  
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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

**Purpose** Construct `arxml.importer` object

**Syntax** `importer_obj = importer(filename)`

**Description** `importer` is a method of the class `arxml.importer`.  
`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.

---

**See Also** `createComponentAsModel`; `createComponentAsSubsystem`;  
`getDependencies`; `getFile`; `setDependencies`; `setFile`

# 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 the 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 Model Interfaces” and “Controlling Model 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*” 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

<b>Purpose</b>	Termination entry point in generated code for ERT-based Simulink model
<b>Syntax</b>	<code>void model_terminate(void)</code>
<b>Description</b>	<p>The Real-Time Workshop Embedded Coder software generates the <code>model_terminate</code> function for a Simulink model when the <b>Terminate function required</b> 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 <b>Terminate function required</b> configuration option in the Configuration Parameters dialog box.</p>
<b>See Also</b>	<code>model_initialize</code> , <code>model_SetEventsForThisBaseStep</code> , <code>model_step</code> “Model Entry Points” in the Real-Time Workshop Embedded Coder documentation

# 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	
<b>Copy Utility Function</b>			
memcpy			
<b>Nonfinite Support Utility Functions</b>			
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  
“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

<b>Purpose</b>	Create TFL promotable macro entry based on specified parameters and register in TFL table (for abs function replacement only)
<b>Syntax</b>	<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>
<b>Arguments</b>	<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.h</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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# 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 Your Target” in the Real-Time Workshop Embedded Coder documentation.

`rtwdemo_rtiostream`

`rtwdemo_custom_pil`

<b>Purpose</b>	Initialize communications channel with remote processor
<b>Syntax</b>	<pre>int rtIOStreamOpen(     int    argc,     void * argv[ ] )</pre>
<b>Arguments</b>	<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>
<b>Description</b>	<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>
<b>See Also</b>	rtIOStreamSend, rtIOStreamRecv, rtIOStreamClose, rtiostream_wrapper

## rtIOStreamOpen

---

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

rtwdemo\_rtiostream

rtwdemo\_custom\_pil

# RTW.configSubsystemBuild

---

<b>Purpose</b>	Configure C function prototype or C++ encapsulation interface for right-click build of specified subsystem		
<b>Syntax</b>	RTW.configSubsystemBuild( <i>block</i> )		
<b>Description</b>	<p>RTW.configSubsystemBuild(<i>block</i>) 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 <b>Language</b> value selected for your model on the <b>Real-Time Workshop</b> 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"><li>• Select the system target file <code>ert.tlc</code> for the model.</li><li>• Select the <b>Language</b> parameter value C++ (Encapsulated) for the model.</li><li>• 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>.</li></ul>		
<b>Inputs</b>	<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.		
<b>See Also</b>	<p>“Configuring Function Prototypes for Nonvirtual Subsystems” in the Real-Time Workshop Embedded Coder documentation</p> <p>“Controlling Model 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>		



“Generating and Controlling C++ Encapsulation Interfaces” in the  
Real-Time Workshop Embedded Coder documentation

# rtw.connectivity.ComponentArgs

---

**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	<code>componentPath = obj.getComponentPath</code>
	Returns the Simulink system path of the source component (e.g., the path of the referenced model that is under test).
getComponentCodePath	<code>componentCodePath = obj.getComponentCodePath</code>
	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).

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 Generated Code with Processor-in-the-Loop” in the Real-Time Workshop Embedded Coder documentation.

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

# rtw.connectivity.Config

---

**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 Feature 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 Generated Code with Processor-in-the-Loop” in the Real-Time Workshop Embedded Coder documentation.

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

`rtwdemo_custom_pil`

**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 2-131 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.

# rtw.connectivity.ConfigRegistry

---

## Properties of rtw.connectivity.ConfigRegistry (Continued)

Property Name	Description
SystemTargetFile	Cell array of strings listing System Target Files that support this ConfigRegistry. An empty cell array matches any System Target File. The model's SystemTargetFileConfiguration Parameter is validated against this cell array to determine if this ConfigRegistry is valid for use.
TemplateMakefile	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). The model's TemplateMakefile Configuration Parameter is validated against this cell array to determine if this ConfigRegistry is valid for use.
TargetHWDeviceType	Cell array of strings listing Hardware Device Types that support this ConfigRegistry. An empty cell array matches any Hardware Device Type. The model's TargetHWDeviceTypeConfiguration Parameter is validated against this cell array to determine if this ConfigRegistry is valid for use.



## 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: 1.1.4.7 $

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:

# rtw.connectivity.ConfigRegistry

---

- 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 Generated Code with Processor-in-the-Loop” in the Real-Time Workshop Embedded Coder documentation.

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

`rtwdemo_custom_pil`

# rtw.connectivity.Launcher

---

**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="912 317 1209 343"><code>obj.startApplication</code></p> <p data-bbox="912 361 1298 612">Abstract method that you must implemented 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="912 621 1268 803">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="949 838 1357 899">exe = this.getBuilder.get... ApplicationExecutable</pre>
stopApplication	<p data-bbox="912 954 1194 980"><code>obj.stopApplication</code></p> <p data-bbox="912 998 1298 1249">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 data-bbox="912 1258 1268 1440">Use the <code>getApplicationExecutable</code> method of the associated <code>rtw.connectivity.Builder</code> object to determine the application to stop, e.g.,</p>

# rtw.connectivity.Launcher

---

Method	Syntax and Description
	<pre>exe = this.getBuilder.get... ApplicationExecutable</pre>
getComponentArgs	<pre>componentArgs = obj.getComponentArgs</pre> <p>Returns the <code>rtw.connectivity.ComponentArgs</code> object associated with this Launcher object.</p>

## See Also

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

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

`rtwdemo_custom_pil`

**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)
```

# rtw.connectivity.MakefileBuilder

---

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 Generated Code with Processor-in-the-Loop” in the Real-Time Workshop Embedded Coder documentation.

“Creating a Connectivity Configuration for Your 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

# rtw.connectivity.RtIOStreamHostCommunicator

---

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.

## See Also

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

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

“Creating a Connectivity Configuration for Your 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** `obj = RTW.getEncapsulationInterfaceSpecification(modelName)`

**Description** `obj = RTW.getEncapsulationInterfaceSpecification(modelName)` 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

---

“Generating and Controlling C++ Encapsulation Interfaces” in the  
Real-Time Workshop Embedded Coder documentation

<b>Purpose</b>	Get handle to model-specific C prototype function control object	
<b>Syntax</b>	<code>obj = RTW.getFunctionSpecification(modelName)</code>	
<b>Description</b>	<code>obj = RTW.getFunctionSpecification(modelName)</code> returns a handle to the model-specific C function prototype control object.	
<b>Inputs</b>	<i>modelName</i>	String specifying the name of a loaded ERT-based Simulink model.
<b>Outputs</b>	<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 [].
<b>Alternatives</b>	The <b>Configure Model Functions</b> button on the <b>Interface</b> 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.	
<b>See Also</b>	“Controlling Model 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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation



<b>Purpose</b>	Create C++ encapsulation interface object for configuring model class with I/O arguments style step method
<b>Syntax</b>	<code>obj = RTW.ModelCPPArgsClass</code>
<b>Description</b>	<code>obj = RTW.ModelCPPArgsClass</code> returns a handle, <i>obj</i> , to a newly created object of class <code>RTW.ModelCPPArgsClass</code> .
<b>Outputs</b>	<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.
<b>Alternatives</b>	The <b>Configure C++ Encapsulation Interface</b> button on the <b>Interface</b> 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.
<b>See Also</b>	“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 “Generating and Controlling C++ Encapsulation 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.

<b>Methods</b>		
	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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelCPPVoidClass class

---

<b>Superclasses</b>	ModelCPPClass
<b>Purpose</b>	Control C++ encapsulation interfaces for models using void-void style step method
<b>Description</b>	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.
<b>Construction</b>	RTW.ModelCPPVoidClass      Create C++ encapsulation interface object for configuring model class with void-void style step method
<b>Methods</b>	See the methods of the base class RTW.ModelCPPClass, plus the following method.  runValidation      Validate model-specific C++ encapsulation interface against Simulink model
<b>Copy Semantics</b>	Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.
<b>Alternatives</b>	The <b>Configure C++ Encapsulation Interface</b> button on the <b>Interface</b> 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

“Generating and Controlling C++ Encapsulation 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  
“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype class

---

<b>Purpose</b>	Describe signatures of functions for model	
<b>Description</b>	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.	
<b>Construction</b>	<code>RTW.ModelSpecificCPrototype</code>	Create model-specific C prototype object
<b>Methods</b>	<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 Model 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 Model 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 Generated Code with Processor-in-the-Loop” in the Real-Time Workshop Embedded Coder documentation.

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

“Build Information Object” in the Real-Time Workshop documentation.

`rtwdemo_custom_pil`

## Purpose

Validate RTW.AutosarInterface object against model

## Syntax

```
[status, msg] = autosarInterfaceObj.runValidation
```

## Description

runValidation is a method of the class RTW.AutosarInterface.

[Status, Message] = autosarInterfaceObj.runValidation runs a validation check for the RTW.AutosarInterface object against the model to which the object is attached. Before calling this method, you must call the attachToModel method.

autosarInterfaceObj is a model-specific RTW.AutosarInterface object.

Status returns a status flag indicating whether the configuration is valid. If valid, status is true; otherwise, it is false.

Message: If the returned status flag is false, Message stores the explanation of why the configuration is invalid.

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).

# runValidation (AUTOSAR)

---

## Validation Checks (Continued)

Group	Check
	<p>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).</p> <p>AUTOSAR XML options for the interface package name and data type package name must be valid AUTOSAR path identifiers (see definition 3 following).</p>
Valid names and paths for sender/receiver ports	<p>For sender/receiver ports (Implicit or explicit data access mode):</p> <ul style="list-style-type: none"><li>• Simulink ports may have duplicated AUTOSAR port names, however the AUTOSAR Interface name must also be the same.</li><li>• A Simulink inport and an outport cannot have the same AUTOSAR port name.</li><li>• For any duplicated AUTOSAR port name and AUTOSAR Interface name, the Data element names must be unique.</li><li>• Sender/receiver ports AUTOSAR port name cannot be the same as the ServiceName of a basic software port.</li><li>• Sender/receiver ports AUTOSAR port name and Interface cannot be the same as the port name or interface of a calibration object.</li><li>• Sender/receiver ports Interface plus XML Option Interface package (e.g., of the form AUTOSAR/Service/servicename) cannot be</li></ul>



## Validation Checks (Continued)

Group	Check
	the same as the <code>ServiceInterface</code> of a basic software port.
Valid names and paths for basic software ports	<p>For basic software ports:</p> <ul style="list-style-type: none"><li>• <code>ServiceName</code> and <code>ServiceOperation</code> must be valid AUTOSAR short name identifiers (see definition 1 following); and <code>ServiceInterface</code> must be a valid AUTOSAR path identifier (see definition 3 following).</li><li>• Simulink ports may have duplicated <code>ServiceName</code>, however the <code>ServiceInterface</code> must also be the same.</li><li>• For any duplicated <code>ServiceName</code> and <code>ServiceInterface</code>, the <code>ServiceOperation</code> must be unique.</li><li>• For duplicated <code>ServiceOperation</code> and <code>ServiceInterface</code>, the <code>ServiceName</code> must be unique.</li><li>• Basic software port <code>ServiceName</code> name and <code>ServiceInterface</code> cannot be the same as the port name or interface of a calibration object.</li></ul>

# runValidation (AUTOSAR)

---

## Validation Checks (Continued)

Group	Check
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.
	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.

Definitions of 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.

## 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.)
	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 interrunnable variable.

# runValidation (AUTOSAR)

---

## Multiple Runnable Validation Checks (Continued)

Group	Check
	<p>A signal connected to an output of the wrapper subsystem cannot have an inport of that subsystem as its source.</p>
	<p>All data store memory blocks referenced from subsystems must be contained in the subsystems, to prevent data integrity issues.</p>
	<p>All lines must be contiguous. No line in the wrapper subsystem can be an output of a virtual Bus Creator or Mux block</p>
	<p>Constant blocks are not allowed in the wrapper subsystem.</p>
	<p>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.</p>
Wrapper level Merge block validation	<p>Merge blocks have some restrictions at wrapper level:</p> <ul style="list-style-type: none"><li data-bbox="654 1003 1243 1130">• A merge block is only allowed in the wrapper subsystem when the merge block output is connected to a diagram outputport (not another Merge block).</li><li data-bbox="654 1147 1265 1239">• The input to a Merge block in the wrapper subsystem must be connected to a function-call subsystem outputport.</li><li data-bbox="654 1256 1218 1321">• The input to a Merge block in the wrapper subsystem does not need a label.</li><li data-bbox="654 1338 1277 1402">• A merge block in the wrapper subsystem cannot merge signals of unequal widths.</li></ul>

## Multiple Runnable Validation Checks (Continued)

Group	Check
	<ul style="list-style-type: none"> <li>You cannot connect a Merge block in the wrapper subsystem to more than one output of any given function-call subsystem.</li> </ul>
Other multiple runnable validation checks	<p>All runnable names, event names, and interrunnable variable names must be unique. Lines representing interrunnable variables must be labeled 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>
	<p>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.</p>
	<p>The blocks in the top diagram of the wrapper subsystem must not have unconnected ports.</p>
	<p>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.</p>
	<p>The sample time of the inport associated with an error status must be the same sample time as its corresponding data port.</p>

# runValidation (AUTOSAR)

---

## Multiple Runnable Validation Checks (Continued)

Group	Check
	Each function call subsystem being exported as a runnable entity must specify an AUTOSAR interface.

# RTW.ModelCPPArgsClass.runValidation

---

<b>Purpose</b>	Validate model-specific C++ encapsulation interface against Simulink model				
<b>Syntax</b>	<code>[status, msg] = runValidation(obj)</code>				
<b>Description</b>	<p><code>[status, msg] = runValidation(obj)</code> runs a validation check of the specified model-specific C++ encapsulation interface against the ERT-based Simulink model to which it is attached.</p> <p>Before calling this function, you must call either <code>attachToModel</code>, to attach a function prototype to a loaded model, or <code>RTW.getEncapsulationInterfaceSpecification</code>, to get the handle to a function prototype previously attached to a loaded model.</p>				
<b>Inputs</b>	<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></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>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>				
<b>Outputs</b>	<table><tr><td><i>status</i></td><td>Boolean value; true for a valid configuration, false otherwise.</td></tr><tr><td><i>msg</i></td><td>If <i>status</i> is false, <i>msg</i> contains a string of information describing why the configuration is invalid.</td></tr></table>	<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.
<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.				
<b>Alternatives</b>	To validate a C++ encapsulation interface in the Simulink Configuration Parameters graphical user interface, go to the <b>Interface</b> pane and click the <b>Configure C++ Encapsulation Interface</b> 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 <b>Validate</b> button to validate your current model step function configuration. The <b>Validation</b> pane displays success or failure				

# RTW.ModelCPPArgsClass.runValidation

---

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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation



# RTW.ModelCPPVoidClass.runValidation

---

<b>Purpose</b>	Validate model-specific C++ encapsulation interface against Simulink model				
<b>Syntax</b>	<code>[status, msg] = runValidation(obj)</code>				
<b>Description</b>	<p><code>[status, msg] = runValidation(obj)</code> runs a validation check of the specified model-specific C++ encapsulation interface against the ERT-based Simulink model to which it is attached.</p> <p>Before calling this function, you must call either <code>attachToModel</code>, to attach a function prototype to a loaded model, or <code>RTW.getEncapsulationInterfaceSpecification</code>, to get the handle to a function prototype previously attached to a loaded model.</p>				
<b>Inputs</b>	<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.ModelCPPVoidClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</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.ModelCPPVoidClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</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.ModelCPPVoidClass</code> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(modelName)</code>				
<b>Outputs</b>	<table><tr><td><i>status</i></td><td>Boolean value; true for a valid configuration, false otherwise.</td></tr><tr><td><i>msg</i></td><td>If <i>status</i> is false, <i>msg</i> contains a string of information describing why the configuration is invalid.</td></tr></table>	<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.
<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.				
<b>Alternatives</b>	To validate a C++ encapsulation interface in the Simulink Configuration Parameters graphical user interface, go to the <b>Interface</b> pane and click the <b>Configure C++ Encapsulation Interface</b> 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 <b>Validate</b> button to validate your current model step function configuration. The <b>Validation</b> pane displays success or failure				

# RTW.ModelCPPVoidClass.runValidation

---

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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype.runValidation

---

<b>Purpose</b>	Validate model-specific C function prototype against Simulink model	
<b>Syntax</b>	<code>[status, msg] = runValidation(obj)</code>	
<b>Description</b>	<p><code>[status, msg] = runValidation(obj)</code> runs a validation check of the specified model-specific C function prototype against the ERT-based Simulink model to which it is attached.</p> <p>Before calling this function, you must call either <code>attachToModel</code>, to attach a function prototype to a loaded model, or <code>RTW.getFunctionSpecification</code>, to get the handle to a function prototype previously attached to a loaded model.</p>	
<b>Inputs</b>	<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> .
<b>Outputs</b>	<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.
<b>Alternatives</b>	Click the <b>Validate</b> 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.	
<b>See Also</b>	“Controlling Model 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** `setArgCategory(obj, portName, category)`

**Description** `setArgCategory(obj, portName, category)` sets the category — 'Value', 'Pointer', or 'Reference' — of the argument corresponding to a specified Simulink model inport or outport in a 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> or <i>obj</i> = <code>RTW.getEncapsulationInterfaceSpecification(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', 'Pointer', or 'Reference' — to be set for the specified Simulink model port.

---

**Note** If you change the argument category for an outport 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 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

“Generating and Controlling C++ Encapsulation 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.

**See Also**

“Controlling Model 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** `setArgName(obj, portName, argName)`

**Description** `setArgName(obj, portName, argName)` sets the argument name that corresponds to a specified Simulink model inport or outport in 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.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>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



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

“Generating and Controlling C++ Encapsulation 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 Model 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

`setArgPosition(obj, portName, position)`

## Description

`setArgPosition(obj, portName, position)` 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

*obj*

Handle to a model-specific C++ encapsulation interface control object, such as a handle previously returned by *obj* = `RTW.ModelCPPArgsClass` or *obj* = `RTW.getEncapsulationInterfaceSpecification(modelName)`

*portName*

String specifying the name of an inport or outport in your Simulink model.

*position*

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

# RTW.ModelCPPArgsClass.setArgPosition

---

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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype.setArgPosition

---

<b>Purpose</b>	Set argument position for Simulink model port in model-specific C function prototype						
<b>Syntax</b>	<code>setArgPosition(obj, portName, position)</code>						
<b>Description</b>	<code>setArgPosition(obj, portName, position)</code> 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.						
<b>Inputs</b>	<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>position</i></td><td>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.</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>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.
<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.						
<b>Alternatives</b>	Use the <b>Step function arguments</b> 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.						
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code						

# RTW.ModelCPPArgsClass.setArgQualifier

---

**Purpose** Set argument type qualifier for Simulink model port in model-specific C++ encapsulation interface

**Syntax** `setArgQualifier(obj, portName, qualifier)`

**Description** `setArgQualifier(obj, portName, qualifier)` sets the type qualifier — 'none', 'const', 'const \*', 'const \* const', or 'const &' — of the argument that corresponds to a specified Simulink model inport or output in a specified model-specific C++ encapsulation interface.

<b>Inputs</b>	<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 name of an inport or output in your Simulink model.
	<i>qualifier</i>	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.

## **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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelSpecificCPrototype.setArgQualifier

---

<b>Purpose</b>	Set argument type qualifier for Simulink model port in model-specific C function prototype						
<b>Syntax</b>	<code>setArgQualifier(obj, portName, qualifier)</code>						
<b>Description</b>	<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.						
<b>Inputs</b>	<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.						
<b>Alternatives</b>	Use the <b>Step function arguments</b> 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.						
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code						



**Purpose** Set class name in model-specific C++ encapsulation interface

**Syntax** `setClassName(obj, className)`

**Description** `setClassName(obj, className)` 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> = RTW.ModelCPPArgsClass, <i>obj</i> = RTW.ModelCPPVoidClass, or <i>obj</i> = RTW.getEncapsulationInterfaceSpecification( <i>modelName</i> )
<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

## **RTW.ModelCPPClass.setClassName**

---

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

“Generating and Controlling C++ Encapsulation Interfaces” in the Real-Time Workshop Embedded Coder documentation

**Purpose** Set XML component name

**Syntax** `componentName = autosarInterfaceObj.setComponentName`

**Description** `setComponentName` is a method of the class `RTW.AutosarInterface`.  
`componentName = autosarInterfaceObj.setComponentName` sets the XML component name of the `RTW.AutosarInterface` object specified by `autosarInterfaceObj`.  
`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.  
`componentName` is the name of the XML component to be set to `autosarInterfaceObj`

# setDependencies

---

**Purpose** Set XML file dependencies

**Syntax** `importerObj.setDependencies(Dependencies)`

**Description** `setDependencies` is a method of the class `arxml.importer`.  
`importerObj.setDependencies(Dependencies)` sets the XML file dependencies `Dependencies` associated with the `arxml.importer` object, `importerObj`.

`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** `getDependencies`; `getFile`; `setFile`

**Purpose** Set XML file name for `arxml.importer` object

**Syntax** `importerObj.setFile(filename)`

**Description** `setFile` is a method of the class `arxml.importer`.  
`filename = importerObj.setFile` sets the XML filename associated with the `arxml.importer` object, `importerObj`.

---

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

---

**See Also** `getDependencies`; `getFile`; `setDependencies`

# RTW.ModelSpecificCPrototype.setFunctionName

---

<b>Purpose</b>	Set function name in model-specific C function prototype						
<b>Syntax</b>	<code>setFunctionName(obj, fcnName, fcnType)</code>						
<b>Description</b>	<code>setFunctionName(obj, fcnName, fcnType)</code> sets the step or initialization function name in the specified function control object.						
<b>Inputs</b>	<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.						
<b>Alternatives</b>	Use the <b>Initialize function name</b> and <b>Step function name</b> 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.						
<b>See Also</b>	“Controlling Model Function Prototypes” — Explains how to configure model function prototypes in generated code						

**Purpose** Set initial event name

**Syntax** `autosarInterfaceObj.setInitEventName(initEventName)`

**Description** `setInitEventName` is a method of the class `RTW.AutosarInterface`.  
`autosarInterfaceObj.setInitEventName(initEventName)` sets the initial event name to the specified `AutosarInterface` object.  
`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.  
`initEventName` is the name of the initial event to be set to `autosarInterfaceObj`.

# setInitRunnableName

---

**Purpose** Set initial runnable name

**Syntax** `autosarInterfaceObj.setInitRunnableName(initRunnableName)`

**Description** `setInitRunnableName` is a method of the class `RTW.AutosarInterface`.  
`autosarInterfaceObj.setInitRunnableName(initRunnableName)` sets the initial runnable name for the specified `AutosarInterface` object.

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

`initRunnableName` is the initial runnable name to be set to `autosarInterfaceObj`.



<b>Purpose</b>	Set AUTOSAR port name
<b>Syntax</b>	<code>autosarInterfaceObj.setIOAutosarPortName(portName, autosarPort)</code>
<b>Description</b>	<p><code>setIOAutosarPortName</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>autosarInterfaceObj.setIOAutosarPortName(portName, autosarPort)</code> updates the AUTOSAR port name in the configuration for the specified port.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>autosarPort</code> is the AUTOSAR port name to be set (string).</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>
<b>See Also</b>	<code>getIOAutosarPortName</code>

# setIODataAccessMode

---

<b>Purpose</b>	Set I/O data access mode
<b>Syntax</b>	<code>autosarInterfaceObj.setIODataAccessMode(portName,dataAccessMode)</code>
<b>Description</b>	<p><code>setIODataAccessMode</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>autosarInterfaceObj.setIODataAccessMode(portName,dataAccessMode)</code> sets the data access mode in the configuration for the specified port.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>dataAccessMode</code> is the data access mode to be set, one of 'ImplicitSend', 'ImplicitReceive', or 'ExplicitSend', 'ExplicitReceive' (string).</p>

<b>Purpose</b>	Set I/O data element
<b>Syntax</b>	<code>autosarInterfaceObj.setIODataElement(portName,dataElement)</code>
<b>Description</b>	<p><code>setIODataElement</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>autosarInterfaceObj.setIODataElement(portName,dataElement)</code> updates the Data Element in the configuration for the specified port.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string)</p> <p><code>dataElement</code> is the data element to be set (string).</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>
<b>See Also</b>	<code>getIODataElement</code>

# setIOInterfaceName

---

<b>Purpose</b>	Set I/O interface name
<b>Syntax</b>	<code>autosarInterfaceObj.setIOInterfaceName(portName, interfaceName)</code>
<b>Description</b>	<p><code>setIOInterfaceName</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>autosarInterfaceObj.setIOInterfaceName(portName, interfaceName)</code> updates the I/O interface name in the configuration for the specified port.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p> <p><code>portName</code> is the inport/outport name (string).</p> <p><code>interfaceName</code> is the I/O interface name to be set (string).</p> <p>By default the AUTOSAR port name, data element name, and interface name are the same as the Simulink port name.</p>
<b>See Also</b>	<code>getIOInterfaceName</code>

**Purpose** Set periodic event name

**Syntax** `autosarInterfaceObj.setPeriodicEventName(periodicEventName)`

**Description** `setPeriodicEventName` is a method of the class `RTW.AutosarInterface`.  
`autosarInterfaceObj.setPeriodicEventName(periodicEventName)` sets the periodic event name to the `AutosarInterface` object.  
`autosarInterfaceObj` is a model-specific `RTW.AutosarInterface` object.  
`periodicEventName` is the name of the periodic event to be set to `autosarInterfaceObj`.

# setPeriodicRunnableName

---

**Purpose** Set periodic runnable name

**Syntax** `autosarInterfaceObj.setPeriodicRunnableName(periodicRunnableName)`

**Description** `setPeriodicRunnableName` is a method of the class `RTW.AutosarInterface`.

`autosarInterfaceObj.setPeriodicRunnableName(periodicRunnableName)` sets the periodic runnable name to the `RTW.AutosarInterface` object, `autosarInterfaceObj`.

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

`periodicRunnableName` is the periodic runnable name to be set to `autosarInterfaceObj`.

**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

# setReservedIdentifiers

---

identifiers can be associated with an arbitrary TFL, while the other three (if present) must be associated with ANSI<sup>®1</sup>, ISO<sup>®2</sup>, or GNU<sup>®3</sup> 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.Tf1Table;

% 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<sup>®</sup> is a registered trademark of the American National Standards Institute, Inc.
2. ISO<sup>®</sup> is a registered trademark of the International Organization for Standardization.
3. GNU<sup>®</sup> is a registered trademark of the Free Software Foundation.



```
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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# RTW.ModelCPPClass.setStepMethodName

---

**Purpose** Set step method name in model-specific C++ encapsulation interface

**Syntax** `setStepMethodName(obj, fcnName)`

**Description** `setStepMethodName(obj, fcnName)` sets the step method 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>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.

**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

“Generating and Controlling C++ Encapsulation 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			

## 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 ''.

# setTflCFunctionEntryParameters

---

## 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.

## 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 Floating-Point 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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# setTflCOperationEntryParameters

---

**Purpose** Set specified parameters for operator entry in TFL table

**Syntax** `void setTflCOperationEntryParameters(hEntry, varargin)`

**Arguments** *hEntry*  
Handle to a TFL table entry previously returned by *hEntry* = RTW.TflCOperationEntry.

---

**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.TflCOperationEntryGenerator` rather than `hEntry = RTW.TflCOperationEntry`. If you want to use `NetSlopeAdjustmentFactor` and `NetSlopeFixedExponent`, instantiate your table entry using `hEntry = RTW.TflCOperationEntryGenerator_NetSlope`.

---

*varargin*  
Parameter/value pairs for the operator entry. See `varargin` Parameters.

## **varargin Parameters**

The following operator entry parameters can be specified to the `setTflCOperationEntryParameters` function using parameter/value argument pairs. For example,

```
setTflCOperationEntryParameters(..., '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)



# setTfllCOperationEntryParameters

---

- 'RTW\_OP\_MUL' for \* (multiplication)
- 'RTW\_OP\_DIV' for / (division)

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'.

# setTflCOperationEntryParameters

---

## 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`.

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.Tf1COperationEntryGenerator` rather than `hEntry = RTW.Tf1COperationEntry`.

## 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 `NetSlopeFixedExponent` 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`.

# setTflCOperationEntryParameters

---

## NetSlopeFixedExponent

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`.

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.TflCOperationEntryGenerator_NetSlope` rather than `hEntry = RTW.TflCOperationEntry`.

## 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

# setTf1COperationEntryParameters

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);
```

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', ...
```

# setTflCOperationEntryParameters

---

```
'ImplementationSourceFile', 'u8_add_u8_u8.c' );
```

In the following example, the `setTflCOperationEntryParameters` function is used to set parameters for a TFL operator entry for 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.TflCOperationEntryGenerator;
op_entry.setTflCOperationEntryParameters( ...
    '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_rsf0p125', ...
    'ImplementationHeaderFile', 's16_div_s16_s16_rsf0p125.h', ...
    'ImplementationSourceFile', 's16_div_s16_s16_rsf0p125.c' );
```

In the following example, the `setTflCOperationEntryParameters` 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.TflCOperationEntryGenerator;
op_entry.setTflCOperationEntryParameters( ...
    '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 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

“Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation

# slConfigUIGetVal

---

<b>Purpose</b>	Return current value for custom target configuration option
<b>Syntax</b>	<code>value = slConfigUIGetVal(hDlg, hSrc, 'OptionName')</code>
<b>Arguments</b>	<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>
<b>Returns</b>	Current value of the specified option. The data type of the return value depends on the data type of the option.
<b>Description</b>	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.
<b>Example</b>	In the following example, the <code>slConfigUIGetVal</code> function returns the value of the <b>Terminate function required</b> option on the <b>Real-Time Workshop/Interface</b> 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

# slConfigUISetEnabled

---

<b>Purpose</b>	Enable or disable custom target configuration option
<b>Syntax</b>	<pre>slConfigUISetEnabled(hDlg, hSrc, 'OptionName', true) slConfigUISetEnabled(hDlg, hSrc, 'OptionName', false)</pre>
<b>Arguments</b>	<p><b>hDlg</b> 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><b>hSrc</b> 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><b>'OptionName'</b> Quoted name of the TLC variable defined for a custom target configuration option.</p> <p><b>true</b> Specifies that the option should be enabled.</p> <p><b>false</b> Specifies that the option should be disabled.</p>

**Description** The `slConfigUISetEnabled` 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 `slConfigUISetEnabled` to enable or disable a specified target option.

**Example** In the following example, the `slConfigUISetEnabled` 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

---

<b>Purpose</b>	Set value for custom target configuration option
<b>Syntax</b>	<code>slConfigUISetVal(hDlg, hSrc, 'OptionName', OptionValue)</code>
<b>Arguments</b>	<p><b>hDlg</b> 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><b>hSrc</b> 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><b>'OptionName'</b> Quoted name of the TLC variable defined for a custom target configuration option.</p> <p><b>OptionValue</b> 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

# syncWithModel

---

<b>Purpose</b>	Synchronize configuration with model
<b>Syntax</b>	<code>autosarInterfaceObj.syncWithModel</code>
<b>Description</b>	<p><code>syncWithModel</code> is a method of the class <code>RTW.AutosarInterface</code>.</p> <p><code>autosarInterfaceObj.syncWithModel</code> synchronizes the configuration with the model for <code>RTW.AutosarInterface</code> class.</p> <p><code>autosarInterfaceObj</code> is a model-specific <code>RTW.AutosarInterface</code> object.</p>

# Block Reference

---

Configuration Wizards (p. 3-2)

Automatically update configuration of parent Simulink model

Module Packaging (p. 3-3)

Create potential Simulink data objects

## 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

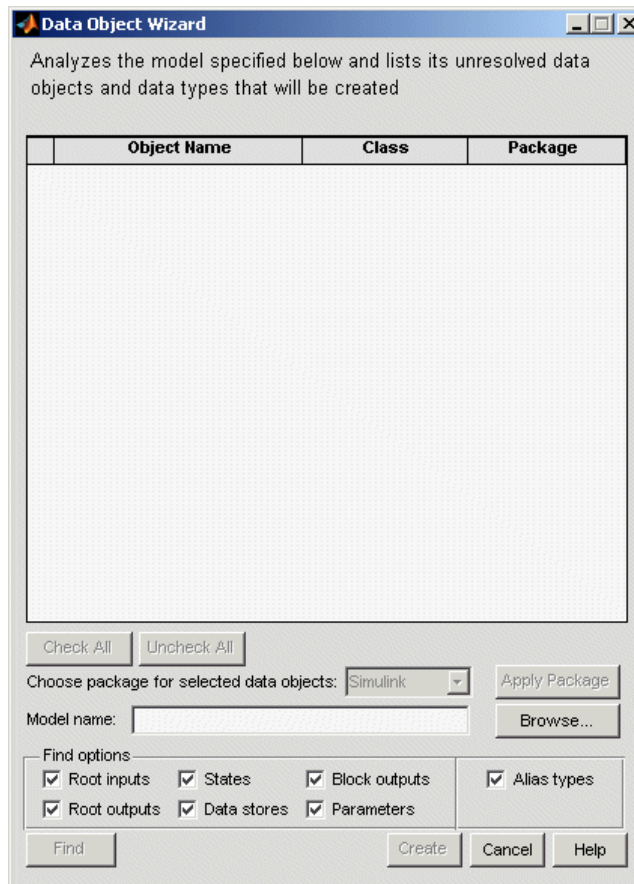
# Data Object Wizard

---

**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

“Creating a Data Dictionary for a Model” 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)

---

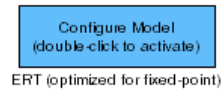
## Purpose

Automatically update active configuration parameters of parent model for ERT fixed-point code generation

## Library

Configuration Wizards

## Description



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)

---

<b>Purpose</b>	Automatically update active configuration parameters of parent model for GRT fixed- or floating-point code generation
<b>Library</b>	Configuration Wizards
<b>Description</b>	<p>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.</p> <p>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.</p>

---

**Note** You can include more than one Configuration Wizard block in your model. This provides a quick way to switch between configurations.

---

<b>Parameters</b>	<p><b>Configure the model for</b> Value selected from</p> <ul style="list-style-type: none"><li>• ERT (optimized for fixed-point)</li><li>• ERT (optimized for floating-point)</li><li>• GRT (optimized for fixed/floating-point)</li><li>• GRT (debug for fixed/floating-point)</li><li>• Custom</li></ul> <p>For this block, GRT (optimized for fixed/floating-point) is selected by default.</p> <p><b>Configuration function</b> Grayed out unless <b>Configure the model for</b> is set to Custom. This parameter is used with the Custom M-file block.</p>
-------------------	--

# 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

## **GRT (optimized for fixed/floating-point)**

---

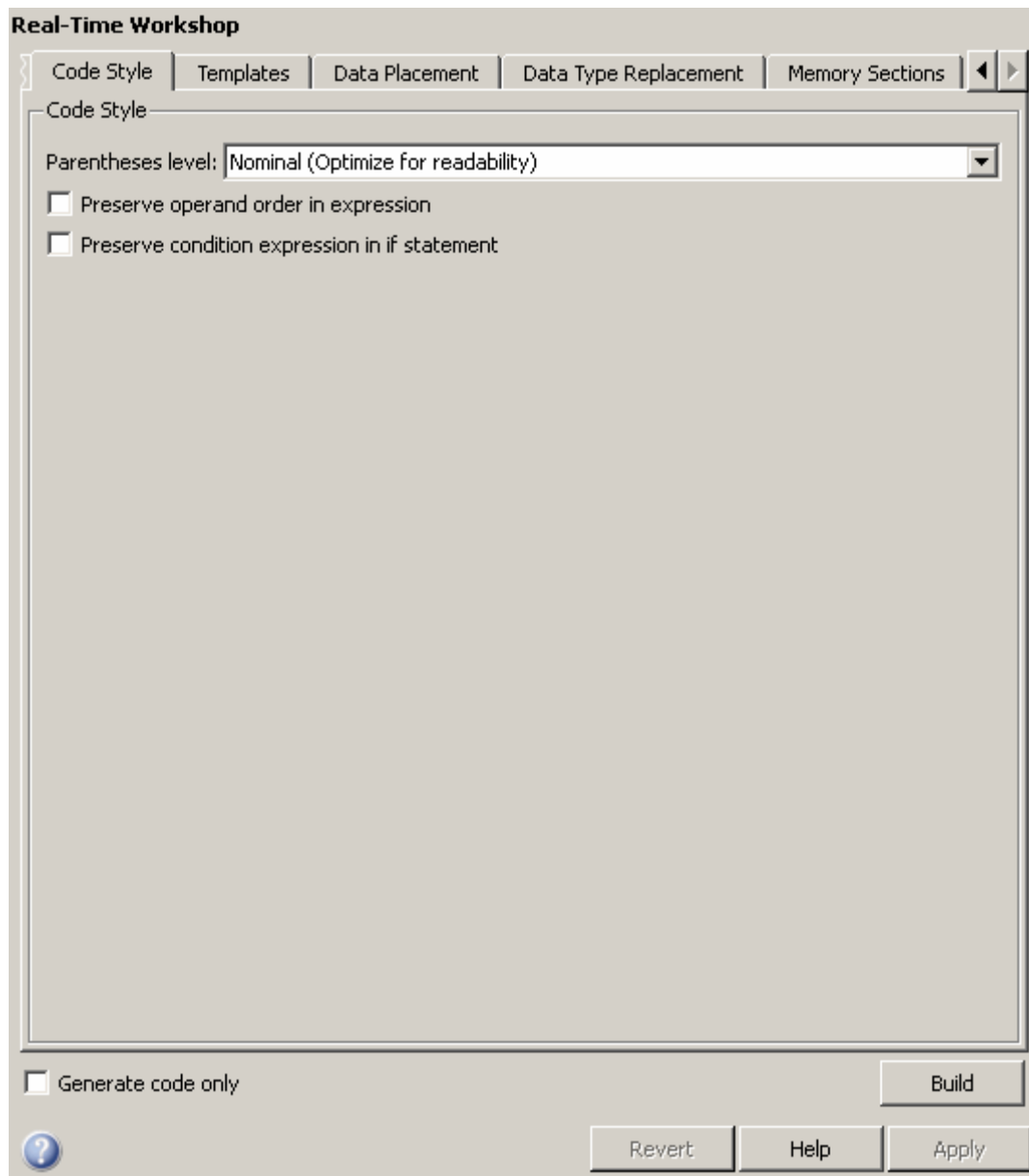


# Configuration Parameters

---

- “Real-Time Workshop Pane: Code Style” on page 5-2
- “Real-Time Workshop Pane: Templates” on page 5-10
- “Real-Time Workshop Pane: Data Placement” on page 5-22
- “Real-Time Workshop Pane: Data Type Replacement” on page 5-41
- “Real-Time Workshop Pane: Memory Sections” on page 5-70
- “Real-Time Workshop Pane: AUTOSAR Code Generation Options” on page 5-87
- “Parameter Reference” on page 5-91

## Real-Time Workshop Pane: Code Style



**In this section...**

“Real-Time Workshop: Code Style Tab Overview” on page 5-4

“Parentheses level” on page 5-5

“Preserve operand order in expression” on page 5-7

“Preserve condition expression in if statement” on page 5-8

### **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”

## 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 ANSI<sup>4</sup> 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<sup>®5</sup> 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<sup>®</sup> is a registered trademark of the American National Standards Institute, Inc.

5. MISRA<sup>®</sup> is a registered trademarks of MIRA Ltd, held on behalf of the MISRA<sup>®</sup> 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

<b>Application</b>	<b>Setting</b>
Debugging	On
Traceability	On
Efficiency	Off
Safety precaution	Off

## 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 5-12

“Code templates: Source file (\*.c) template” on page 5-13

“Code templates: Header file (\*.h) template” on page 5-14

“Data templates: Source file (\*.c) template” on page 5-15

“Data templates: Header file (\*.h) template” on page 5-16

“File customization template” on page 5-17

“Generate an example main program” on page 5-18

“Target operating system” on page 5-20

### **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**

Module Packaging Features

## 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<sup>®6</sup> 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 the Main Program Module
- Static Main Program Module
- Custom File Processing

---

6. VxWorks<sup>®</sup> is a registered trademark of Wind River<sup>®</sup> 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 the Main Program Module](#)
- [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 5-24

“Data definition” on page 5-25

“Data definition filename” on page 5-27

“Data declaration” on page 5-29

“Data declaration filename” on page 5-31

“#include file delimiter” on page 5-32

“Module naming” on page 5-33

“Module name” on page 5-35

“Signal display level” on page 5-37

“Parameter tune level” on page 5-39

### **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**

Module Packaging Features



## 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 File 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 File 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:** DataDefinitionFile

**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

<b>Application</b>	<b>Setting</b>
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 ANSI<sup>7</sup> 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<sup>®</sup> 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

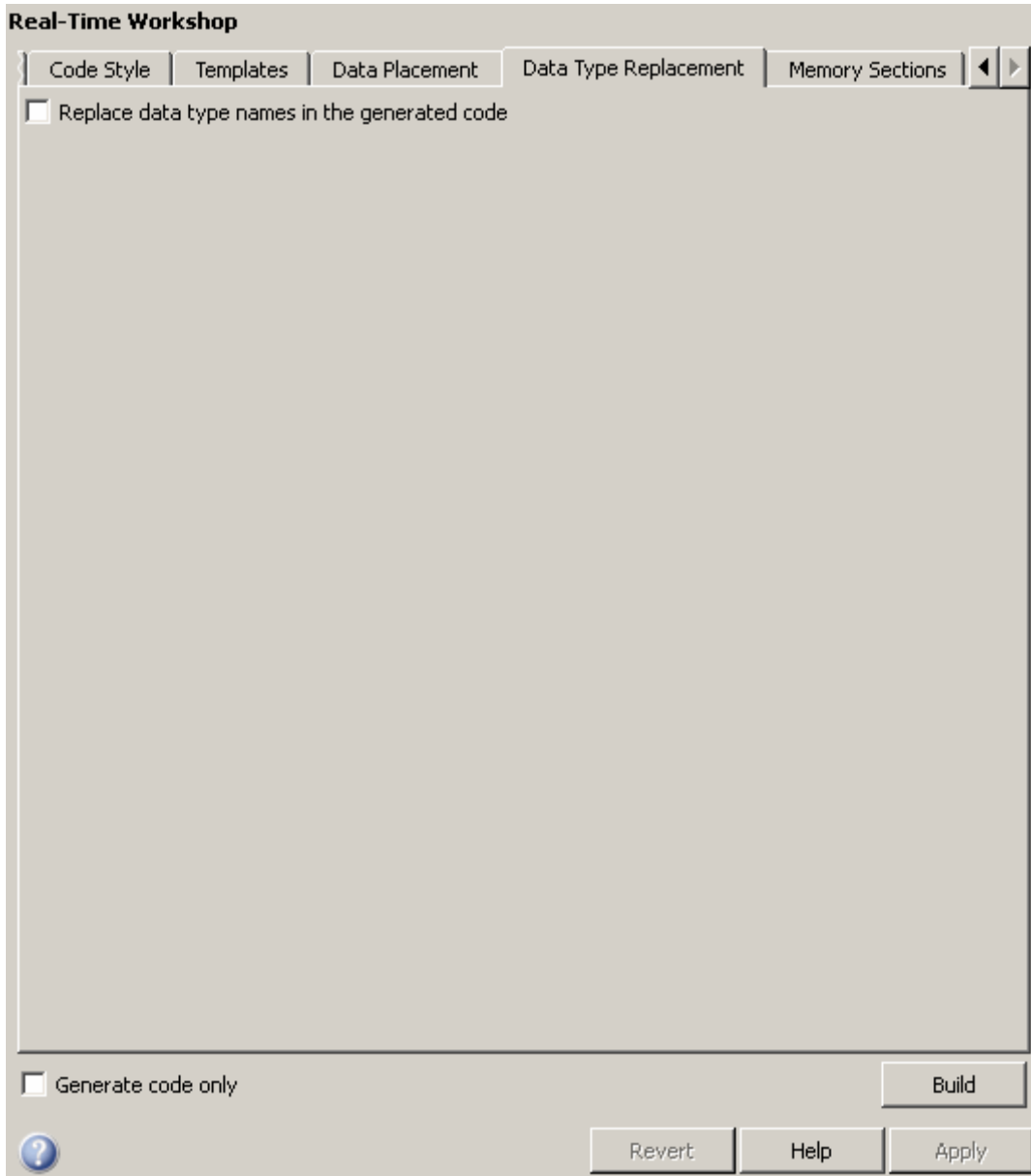
Application	Setting
Debugging	No impact
Traceability	Any valid integer
Efficiency	No impact
Safety precaution	No impact

### **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 5-43

“Replace data type names in the generated code” on page 5-44

“Replacement Name: double” on page 5-46

“Replacement Name: single” on page 5-48

“Replacement Name: int32” on page 5-50

“Replacement Name: int16” on page 5-52

“Replacement Name: int8” on page 5-54

“Replacement Name: uint32” on page 5-56

“Replacement Name: uint16” on page 5-58

“Replacement Name: uint8” on page 5-60

“Replacement Name: boolean” on page 5-62

“Replacement Name: int” on page 5-64

“Replacement Name: uint” on page 5-66

“Replacement Name: char” on page 5-68

## 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 Specify 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”

### 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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string



<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string



<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid value

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid value



<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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

**Type:** string

**Value:** any valid string

**Default:** ''

#### Recommended Settings

Application	Setting
Debugging	No impact
Traceability	Any valid string

<b>Application</b>	<b>Setting</b>
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 5-72

“Package” on page 5-73

“Refresh package list” on page 5-75

“Initialize/Terminate” on page 5-76

“Execution” on page 5-77

“Constants” on page 5-78

“Inputs/Outputs” on page 5-80

“Internal data” on page 5-82

“Parameters” on page 5-84

“Validation results” on page 5-86

### **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



## 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

<b>Application</b>	<b>Setting</b>
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

<b>Application</b>	<b>Setting</b>
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



<b>Application</b>	<b>Setting</b>
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

<b>Application</b>	<b>Setting</b>
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.

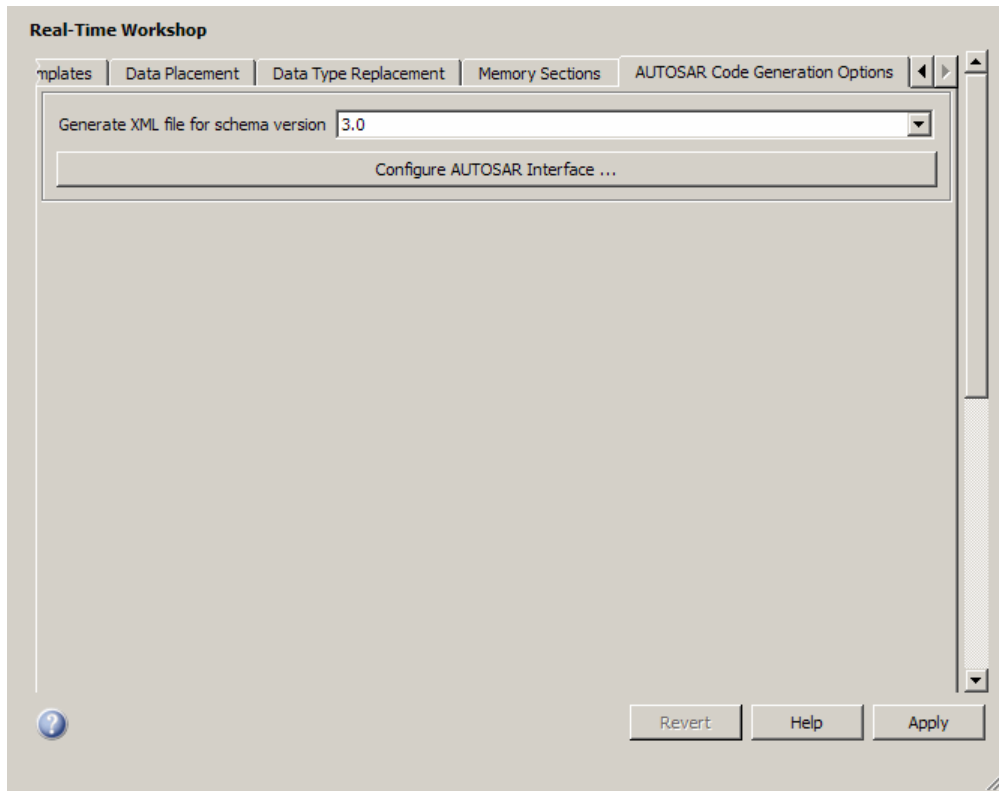
### Settings

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 5-88

“Generate XML file from schema version” on page 5-89

“Configure AUTOSAR Interface” on page 5-90

### **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



## 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 5-91

“Parameter Command-Line Information Summary” on page 5-102

### 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
“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
Remove internal data zero initialization	No impact	No impact	On	Off	Off

**Mapping of Application Requirements to the Optimization Pane (Continued)**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>Optimize initialization code for model reference</b>	No impact	No impact	On	No impact	On
<b>Remove code that protects against division arithmetic exceptions</b>	No impact	No impact	On	Off	Off

**Mapping of Application Requirements to the Real-Time Workshop Pane**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>Ignore custom storage classes</b>	No impact	No impact	No impact	No impact	Off

**Mapping of Application Requirements to the Real-Time Workshop Pane: Report Tab**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>Code-to-model</b>	On	On	No impact	On	Off
<b>Model-to-code</b>	On	On	No impact	On	Off
<b>Eliminated / virtual blocks</b>	On	On	No impact	On	Off
<b>Traceable Simulink blocks</b>	On	On	No impact	On	Off

### Mapping of Application Requirements to the Real-Time Workshop Pane: Report Tab (Continued)

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
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

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
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)**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
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**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
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
Local temporary variables	No impact	Any valid combination of tokens	No impact	\$N\$M	\$N\$M
Local block output variables	No impact	Any valid combination of tokens	No impact	rtb_ \$N\$M	rtb_ \$N\$M

### Mapping of Application Requirements to the Real-Time Workshop Pane: Symbols Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
Constant macros	No impact	Any valid combination of tokens	No impact	\$R\$N\$M	\$R\$N\$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)
Support complex numbers	No impact	No impact	Off for real only	No impact	On

**Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>Support non-finite numbers</b>	No impact	No impact	Off	Off	On
<b>Support absolute time</b>	No impact	No impact	Off	Off	On
<b>Support continuous time</b>	No impact	No impact	Off	Off	Off
<b>Support non-inlined S-functions</b>	No impact	No impact	Off	Off	Off
<b>“Multiword type definitions”</b>	No impact	No impact	Specifying User defined and a low value for <b>Maximum word length</b> reduces the size of the generated file <code>rtwtypes.h</code>	Use default	System defined
<b>“Maximum word length”</b>	No impact	No impact	Smaller values reduce the size of the generated file <code>rtwtypes.h</code>	Use default	256



### Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
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
“Private parameter and state members”	No impact	No impact	No impact	On	On
“Parameter and state access methods”	On	No impact	No impact	Off	Off
“Generate destructor”	No impact	No impact	No impact	Off	On
“I/O access methods”	On	No impact	No impact	Off	Off

**Mapping of Application Requirements to the Real-Time Workshop Pane: Interface Tab (Continued)**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
“Inline access methods”	On	On	On	No impact	Off
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
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**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
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)

### Mapping of Application Requirements to the Real-Time Workshop Pane: Code Style Tab (Continued)

Configuration Parameter	Debugging	Traceability	Efficiency	Safety Precaution	Factory Default
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: 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)**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>File customization template</b>	No impact	No impact	No impact	No impact	example_file_ - process.tlc
<b>Generate an example main program</b>	No impact	No impact	No impact	No impact	On
<b>Target operating system</b>	No impact	No impact	No impact	No impact	BareBoard-Example

**Mapping of Application Requirements to the Real-Time Workshop Pane: Data Placement Tab**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>Data definition</b>	No impact	Any valid value	No impact	No impact	Auto
<b>Data definition filename</b>	No impact	Any valid value	No impact	No impact	global.c
<b>Data declaration</b>	No impact	Any valid value	No impact	No impact	Auto
<b>Data declaration filename</b>	No impact	Any valid value	No impact	No impact	global.h
<b>#include file delimiter</b>	No impact	Any valid value	No impact	No impact	Auto
<b>Module naming</b>	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)**

<b>Configuration Parameter</b>	<b>Debugging</b>	<b>Traceability</b>	<b>Efficiency</b>	<b>Safety Precaution</b>	<b>Factory Default</b>
<b>Inputs/Outputs</b>	No impact	No impact	No impact	No impact	Default
<b>Internal data</b>	No impact	No impact	No impact	No impact	Default
<b>Parameters</b>	No impact	No impact	No impact	No impact	Default
<b>Validation results</b>	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” 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<sup>®</sup> product, or the Simulink<sup>®</sup> Fixed Point<sup>™</sup> 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**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
DataBitsets (Stateflow) <b>off</b> , on	<b>Optimization &gt; Use bitsets for storing boolean data</b>	Use bit sets for storing Boolean data.
InlinedParameterPlacement (ERT) <b>Hierarchical</b> , NonHierarchical	<b>Optimization &gt; Parameter structure</b>	Specify how generated code stores global (tunable) parameters. Specify NonHierarchical to trade off modularity for efficiency.
StrengthReduction (ERT) <b>off</b> , on	<b>Optimization &gt; Simplify array indexing</b>	Suppress generation of code that replaces multiply operations when accessing arrays in a loop.
PassReuseOutputArgsAs (ERT) <b>Structure reference</b> , Individual arguments	<b>Optimization &gt; Pass reusable subsystem output as</b>	Specify how a reusable subsystem passes outputs. Specify Individual arguments for efficiency.
NoFixptDivByZeroProtection (ERT) (Simulink Fixed Point) <b>off</b> , on	<b>Optimization &gt; Remove code that protects against division arithmetic exceptions</b>	Suppress generation of code that guards against division by zero for fixed-point data.
OptimizeModelRefInitCode (ERT) <b>off</b> , on	<b>Optimization &gt; Optimize initialization code for model reference</b>	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,



**Command-Line Information: Optimization Pane (Continued)**

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
		in which case you can disable this optimization.
StateBitsets (Stateflow) <b>off, on</b>	<b>Optimization &gt; Use bitsets for storing state configuration</b>	Use bit sets for storing state configuration.
ZeroExternalMemoryAtStartup (ERT) <b>off, on</b>	<b>Optimization &gt; Remove root level I/O zero initialization</b>	Suppress code that initializes root-level I/O data structures to zero.
ZeroInternalMemoryAtStartup (ERT) <b>off, on</b>	<b>Optimization &gt; Remove internal data zero initialization</b>	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> - <b>off, on</b>	<b>Real-Time Workshop &gt; General &gt; Ignore custom storage classes</b>	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> - <b>off, on</b>	<b>Real-Time Workshop &gt; Report &gt; Model-to-code</b>	Includes model-to-code traceability support in the generated HTML report.

**Command-Line Information: Real-Time Workshop Pane: Report Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
IncludeHyperlinkInReport (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Report &gt; Code-to-model</b>	Link code segments to the corresponding object in the model. This option increases code generation time for large models.
GenerateTraceReport (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Report &gt; Eliminated / virtual blocks</b>	Include summary of eliminated and virtual blocks in Code Generation report.
GenerateTraceReportS1 (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Report &gt; Traceable Simulink blocks</b>	Include summary of Simulink blocks in Code Generation report.
GenerateTraceReportSf (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Report &gt; Traceable Stateflow objects</b>	Include summary of Stateflow objects in Code Generation report.
GenerateTraceReportEm1 (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Report &gt; Traceable Embedded MATLAB functions</b>	Include summary of Embedded MATLAB functions in Code Generation report.

**Command-Line Information: Real-Time Workshop Pane: Comments Tab**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
CustomCommentsFcn (ERT) <i>string</i> -	<b>Real-Time Workshop &gt; Comments &gt; Custom comments function</b>	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)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
EnableCustomComments (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Comments &gt; Custom comments (MPT objects only)</b>	Add a comment above a signal's or parameter's identifier in the generated file.
InsertBlockDesc (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Comments &gt; Simulink block descriptions</b>	Insert the contents of the <b>Description</b> field from the Block Parameters dialog box into the generated code as a comment.
ReqsInCode (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Comments &gt; Requirements in block comments</b>	Include specified requirements in the generated code as a comment.
SFDataObjDesc (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Comments &gt; Stateflow object descriptions</b>	Insert Stateflow object descriptions into the generated code as a comment.
SimulinkDataObjDesc (ERT) <i>string</i> - <b>off</b> , on	<b>Real-Time Workshop &gt; Comments &gt; Simulink data object descriptions</b>	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> - <b>rtb_</b> <i>\$N</i> <b>\$M</b>	<b>Real-Time Workshop &gt; Symbols &gt; Local block output variables</b>	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> - <b>\$R</b> <i>\$N</i> <b>\$M</b> <i>\$F</i>	<b>Real-Time Workshop &gt; Symbols &gt; Subsystem methods</b>	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
CustomSymbolStrField (ERT) <i>string</i> - <b>\$N</b> <i>\$M</i>	<b>Real-Time Workshop &gt; Symbols &gt; Field name of global types</b>	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

**Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
CustomSymbolStrGlobalVar (ERT) <i>string</i> - <b>\$R\$N\$M</b>	<b>Real-Time Workshop &gt; Symbols &gt; Global variables</b>	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> - <b>\$R\$N\$M</b>	<b>Real-Time Workshop &gt; Symbols &gt; Constant macros</b>	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
CustomSymbolStrTmpVar (ERT) <i>string</i> - <b>\$N\$M</b>	<b>Real-Time Workshop &gt; Symbols &gt; Local temporary variables</b>	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) <i>string</i> - <b>\$N\$R\$M</b>	<b>Real-Time Workshop &gt; Symbols &gt; Global types</b>	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

**Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
DefineNamingFcn (ERT) <i>string</i> -	<b>Real-Time Workshop &gt; Symbols &gt; #define naming &gt; Custom M-function</b>	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) <i>string</i> - <b>None</b> , UpperCase, LowerCase, Custom	<b>Real-Time Workshop &gt; Symbols &gt; #define naming</b>	Specify the rule that changes the spelling of all #define names.
IncDataTypeInIds (ERT) <b>off</b> , on	<b>Real-Time Workshop &gt; Symbol &gt; Include data type acronym in identifiers</b>	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) <b>off</b> , on	<b>Real-Time Workshop &gt; Symbols &gt; Include system hierarchy number in identifiers</b>	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('&lt;S3&gt;')</code> at the MATLAB prompt.

**Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
		The argument specified with <code>hilite_system</code> requires an uppercase S.
InlinedPrmAccess (ERT) string - <b>Literals</b> , Macros	<b>Real-Time Workshop &gt; Symbols &gt; Generate scalar inlined parameters as</b>	Specify whether inlined parameters are coded as numeric constants or macros. Specify <b>Macros</b> for more efficient code.
MangleLength (ERT) int - <b>1</b>	<b>Real-Time Workshop &gt; Symbols &gt; Minimum mangle length</b>	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 - <b>None</b> , UpperCase, LowerCase, Custom	<b>Real-Time Workshop &gt; Symbols &gt; Parameter naming</b>	Select a rule that changes spelling of all parameter names.
PrefixModelToSubsysFcnNames (ERT) off, <b>on</b>	<b>Real-Time Workshop &gt; Symbols &gt; Prefix model name to global identifiers</b>	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

**Command-Line Information: Real-Time Workshop Pane: Symbols Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
		when combining multiple models.
SignalNamingRule (ERT) string - <b>None</b> , UpperCase, LowerCase, Custom	<b>Real-Time Workshop &gt; Symbols &gt; Signal naming</b>	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**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
CombineOutputUpdateFcns (ERT) string - off, <b>on</b>	<b>Real-Time Workshop &gt; Interface &gt; Single output/update function</b>	Generate a model's output and update routines into a single-step function.
ERTMaxMultiwordLength (ERT) int - <b>256</b>	<b>Real-Time Workshop &gt; Interface &gt; Maximum word length</b>	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 - <b>System defined</b> , User defined	<b>Real-Time Workshop &gt; Interface &gt; Multiword type definitions</b>	Specify whether to use system-defined or user-defined type definitions for multiword data types in generated code.



**Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
GenerateAccessMethods (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Parameter and state access methods</b>	Generate get/set access methods for non-I/O model structures, including states and parameters, in C++ (Encapsulated) model code.
GenerateDestructor (ERT) string - off, <b>on</b>	<b>Real-Time Workshop &gt; Interface &gt; Generate destructor</b>	Generate a destructor for the model class in C++ (Encapsulated) model code.
GenerateErtSFunction (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Create Simulink (S-Function) block</b>	Wrap the generated code inside an S-Function block. This allows you to validate the generated code in a Simulink model.
GenerateIOAccessMethods (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; I/O access methods</b>	Generate access methods for root-level I/O signals (if possible) in C++ (Encapsulated) model code.
GeneratePrivateDataMembers (ERT) string - off, <b>on</b>	<b>Real-Time Workshop &gt; Interface &gt; Private parameter and state members</b>	Specify whether to generate non-I/O model structures, including states and parameters, as private data members in C++ (Encapsulated) model code.
GRTInterface (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; GRT compatible call interface</b>	Include a code interface (wrapper) that is compatible with the GRT target.

**Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
IncludeMdlTerminateFcn (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Terminate function required</b>	Generate a terminate function for the model.
InlineAccessMethods (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Inline access methods</b>	Inline generated access methods in C++ (Encapsulated) model code.
MatFileLogging (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; MAT-file logging</b>	Generate code that logs data to a MAT-file.
MultiInstanceErrorCode (ERT) string - None, Warning, <b>Error</b>	<b>Real-Time Workshop &gt; Interface &gt; Reusable code error diagnostic</b>	Specify the error diagnostic behavior for cases when data defined in the model violates the requirements for generation of reusable code.
MultiInstanceERTCode (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Reusable code error diagnostic</b>	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)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
PortableWordSizes (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Enable portable word sizes</b>	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 - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; floating-point numbers</b>	Support floating-point data types in the generated code. This option is forced on when SupportNonInlinedSFcns is on.
RootIOFormat (ERT) string - <b>Individual arguments</b> , Structure reference	<b>Real-Time Workshop &gt; Interface &gt; Pass root-level I/O as</b>	Specify how the code generator is to pass root-level I/O data into a reusable function.
SupportAbsoluteTime (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; absolute time</b>	Support absolute time in the generated code. Blocks such as the Discrete Integrator might require absolute time.
SupportComplex (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; complex numbers</b>	Support complex data types in the generated code.

**Command-Line Information: Real-Time Workshop Pane: Interface Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
SupportContinuousTime (ERT) string - off, <b>on</b>	<b>Real-Time Workshop &gt; Interface &gt; continuous time</b>	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 - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; nonfinite numbers</b>	Support nonfinite values (inf, nan, -inf) in the generated code. This option is forced on when SupportNonInlinedSFcns is on.
SuppressErrorStatus (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Interface &gt; Suppress error status in real-time model data structure</b>	Remove the error status field of the real-time model data structure to preserve memory. When on, SupportContinuousTime is off.

**Command-Line Information: Real-Time Workshop Pane: Code Style Tab**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
ParenthesesLevel (ERT) string - Minimum, <b>Nominal</b> , Maximum	<b>Real-Time Workshop &gt; Code Style &gt; Parentheses Level</b>	Control existence of optional parentheses in generated code.

**Command-Line Information: Real-Time Workshop Pane: Code Style Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
PreserveExpressionOrder (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Code Style &gt; Preserve operand order in expression</b>	Control reordering of commutable expressions.
PreserveIfCondition (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Code Style &gt; Preserve condition expression in if statement</b>	Control preservation of if statement conditions.

**Command-Line Information: Real-Time Workshop Pane: Templates Tab**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
ERTCustomFileTemplate (ERT) string - <b>example_file_process.tlc</b>	<b>Real-Time Workshop &gt; Templates &gt; File customization template</b>	Specify a TLC callback script for customizing the generated code.
ERTDataHdrFileTemplate (ERT) string - <b>ert_code_template.cgt</b>	<b>Real-Time Workshop &gt; Templates &gt; Header file (*.h) template</b>	Specify a template that organizes the generated data .h header files.
ERTDataSrcFileTemplate (ERT) string - <b>ert_code_template.cgt</b>	<b>Real-Time Workshop &gt; Templates &gt; Source file (*.c or *.cpp) template</b>	Specify a template that organizes the generated data .c source files.
ERTHdrFileBannerTemplate (ERT) string - <b>ert_code_template.cgt</b>	<b>Real-Time Workshop &gt; Templates &gt; Header file (*.h) template</b>	Specify a template that organizes the generated code .h header files.
ERTSrcFileBannerTemplate (ERT) string - <b>ert_code_template.cgt</b>	<b>Real-Time Workshop &gt; Templates &gt; Source file (*.c or *.cpp) template</b>	Specify a template that organizes the generated code .c or .cpp source files.

**Command-Line Information: Real-Time Workshop Pane: Templates Tab (Continued)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
GenerateSampleERTMain (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Templates &gt; Generate an example main program</b>	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 - <b>BareBoardExample</b> , VxWorksExample	<b>Real-Time Workshop &gt; Templates &gt; Target operating system</b>	Specify the target operating system for the example main <code>ert_main.c</code> or <code>ert_main.cpp</code> . <b>BareBoardExample</b> is a generic example that assumes no operating system. <b>VxWorksExample</b> is tailored to the VxWorks <sup>8</sup> real-time operating system.

**Command-Line Information: Real-Time Workshop Pane: Data Placement Tab**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
DataDefinitionFile (ERT) string - <b>global.c</b>	<b>Real-Time Workshop &gt; Data Placement &gt; Data definition filename</b>	Specify the name of a single separate <code>.c</code> or <code>.cpp</code> file that contains global data definitions.
DataReferenceFile (ERT) string - <b>global.h</b>	<b>Real-Time Workshop &gt; Data Placement &gt; Data declaration filename</b>	Specify the name of a single separate <code>.c</code> or <code>.cpp</code> file that contains global data references.

8. VxWorks<sup>®</sup> is a registered trademark of Wind River<sup>®</sup> 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 - <b>Auto</b> , InSourceFile, InSeparateSourceFile	<b>Real-Time Workshop &gt; Data Placement &gt; Data definition</b>	Select the .c or .cpp file where variables of global scope are defined.
GlobalDataReference (ERT) string - <b>Auto</b> , InSourceFile, InSeparateHeaderFile	<b>Real-Time Workshop &gt; Data Placement &gt; Data declaration</b>	Select the .h file where variables of global scope are declared (for example, <code>extern real_T globalvar;</code> ).
IncludeFileDelimiter (ERT) string - <b>Auto</b> , UseQuote, UseBracket	<b>Real-Time Workshop &gt; Data Placement &gt; #include file delimiter</b>	Specify the delimiter to be used for all data objects that do not have a delimiter specified in the IncludeFile property.
ModuleName (ERT) string -	<b>Real-Time Workshop &gt; Data Placement &gt; Module name</b>	Specify the name of the module that owns this model.
ModuleNamingRule (ERT) string - <b>Unspecified</b> , SameAsModel, UserSpecified	<b>Real-Time Workshop &gt; Data Placement &gt; Module naming</b>	Specify the rule to be used for naming the module.
ParamTuneLevel (ERT) int - <b>10</b>	<b>Real-Time Workshop &gt; Data Placement &gt; Parameter tune level</b>	Specify whether the code generator is to declare a parameter data object as tunable global data in the generated code.
SignalDisplayLevel (ERT) int - <b>10</b>	<b>Real-Time Workshop &gt; Data Placement &gt; Signal display level</b>	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**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
EnableUserReplacementTypes (ERT) string - <b>off</b> , on	<b>Real-Time Workshop &gt; Data Type Replacement</b>	Specify whether to replace built-in data type names with user-defined data type names in generated code.
ReplacementTypes (ERT) string -	<b>Real-Time Workshop &gt; Data Type Replacement &gt; Data type names</b>	Specify names to use for built-in data types in generated code.

**Command-Line Information: Real-Time Workshop Pane: Memory Sections Tab**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
MemSecPackage (ERT) string - --- <b>None</b> ---, Simulink, mpt	<b>Real-Time Workshop &gt; Memory Sections &gt; Package</b>	Specify the package that contains the memory sections that you want to apply.
MemSecFuncInitTerm (ERT) string - <b>Default</b> , MemConst, MemVolatile, MemConstVolatile	<b>Real-Time Workshop &gt; Memory Sections &gt; Initialize/Terminate</b>	Apply memory sections to: <ul style="list-style-type: none"> <li>• Initialize/Start functions</li> <li>• Terminate functions</li> </ul>
MemSecFuncExecute (ERT) string - <b>Default</b> , MemConst, MemVolatile, MemConstVolatile	<b>Real-Time Workshop &gt; Memory Sections &gt; Execution</b>	Apply memory sections to: <ul style="list-style-type: none"> <li>• Step functions</li> <li>• Run-time initialization functions</li> <li>• Derivative functions</li> <li>• Enable functions</li> <li>• Disable functions</li> </ul>



## Command-Line Information: Real-Time Workshop Pane: Memory Sections Tab (Continued)

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
MemSecDataConstants (ERT) string - <b>Default</b> , MemConst, MemVolatile, MemConstVolatile	<b>Real-Time Workshop &gt; Memory Sections &gt; Constants</b>	Apply memory sections to: <ul style="list-style-type: none"> <li>• Constant parameters</li> <li>• Constant block I/O</li> <li>• Zero representation</li> </ul>
MemSecDataIO (ERT) string - <b>Default</b> , MemConst, MemVolatile, MemConstVolatile	<b>Real-Time Workshop &gt; Memory Sections &gt; Inputs/Outputs</b>	Apply memory sections to: <ul style="list-style-type: none"> <li>• Root inputs</li> <li>• Root outputs</li> </ul>
MemSecDataInternal (ERT) string - <b>Default</b> , MemConst, MemVolatile, MemConstVolatile	<b>Real-Time Workshop &gt; Memory Sections &gt; Internal data</b>	Apply memory sections to: <ul style="list-style-type: none"> <li>• Block I/O</li> <li>• DWork vectors</li> <li>• Run-time model</li> <li>• Zero-crossings</li> </ul>
MemSecDataParameters (ERT) string - <b>Default</b> , MemConst, MemVolatile, MemConstVolatile	<b>Real-Time Workshop &gt; Memory Sections &gt; Parameters</b>	Apply memory sections to: <ul style="list-style-type: none"> <li>• Parameters</li> </ul>

**Command-Line Information: Not in GUI**

Parameter and Values	Configuration Parameters Dialog Box Equivalent	Description
CPPClassGenCompliant (ERT) string - off, <b>on</b>	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, <b>on</b>	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)**

<b>Parameter and Values</b>	<b>Configuration Parameters Dialog Box Equivalent</b>	<b>Description</b>
<p>IncludeERTFirstTime (ERT) string - <b>off</b>, on</p> <hr/> <p><b>Note</b> The value of IncludeERTFirstTime is meaningful only if the target configuration parameter ERTFirstTimeCompliant is set to on for your selected target.</p>	Not available	Specify whether Real-Time Workshop Embedded Coder software is to include the firstTime argument in the <i>model_initialize</i> function generated for a Simulink model.
<p>ModelStepFunctionPrototype- ControlCompliant (ERT) string - off, <b>on</b></p>	Not available	Set in SelectCallback 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 off for non-ERT targets and on for ERT targets. (For more information, see “Supporting C Function Prototype Control” in the Real-Time Workshop Embedded Coder documentation.)



## A

- addAdditionalHeaderFile function 2-2
- addAdditionalIncludePath function 2-4
- addAdditionalLinkObj function 2-6
- addAdditionalLinkObjPath function 2-7
- addAdditionalSourceFile function 2-8
- addAdditionalSourcePath function 2-10
- addArgConf method 2-12
- addConceptualArg function 2-14
- addEntry function 2-16
- addIOConf AutosarInterface method 2-18
- attachToModel AutosarInterface method 2-21
- attachToModel method 2-22 to 2-23
- AUTOSAR 2-21 2-65 2-74 to 2-75 2-78 to 2-86 2-90 to 2-91
  - addIOConf 2-18
  - createCalibrationComponentObjects 2-42
  - createComponentAsModel 2-44
  - createComponentAsSubsystem 2-46
  - getCalibrationComponentNames 2-60
  - getComponentName 2-62
  - getComponentNames 2-63
  - getDataTypePackageName 2-64
  - getDependencies 2-69
  - getExecutionPeriod 2-70
  - getFile 2-71
  - getImplementationName 2-73
  - getInterfacePackageName 2-76
  - getInternalBehaviorName 2-77
  - getPortDefaultConf 2-92
  - importer 2-97
  - runValidation 2-161
  - setComponentName 2-189
  - setDependencies 2-190
  - setFile 2-191
  - setInitEventName 2-193
  - setInitRunnableName 2-194
  - setIOAutosarPortName 2-195
  - setIODataAccessMode 2-196
  - setIODataElement 2-197

- setIOInterfaceName 2-198
- setPeriodicEventName 2-199
- setPeriodicRunnableName 2-200
- syncWithModel 2-224

- AUTOSAR Code Generation Options pane 5-87

## B

- blocks

- Custom M-file 4-2
- Data Object Wizard 4-4
- ERT (optimized for fixed-point) 4-6
- ERT (optimized for floating-point) 4-8
- GRT (debug for fixed/floating-point) 4-10
- GRT (optimized for fixed/floating-point) 4-12

## C

- C++ encapsulation interface control

- attachToModel 2-22
- getArgCategory 2-48
- getArgName 2-51
- getArgPosition 2-54
- getArgQualifier 2-57
- getClassName 2-61
- getDefaultConf 2-66
- getNumArgs 2-87
- getStepMethodName 2-94
- RTW.configSubsystemBuild 2-122
- RTW.getEncapsulationInterfaceSpecification 2-1
- RTW.ModelCPPArgsClass 2-144
- RTW.ModelCPPClass 2-148
- RTW.ModelCPPVoidClass 2-150
- runValidation 2-169 2-171
- setArgCategory 2-174
- setArgName 2-178
- setArgPosition 2-181
- setArgQualifier 2-184
- setClassName 2-187
- setStepMethodName 2-204

- Code Style pane 5-2
- configuration parameters
  - code generation 5-102
  - impacts of settings 5-91
  - Real-Time Workshop pane: Code Style 5-4
  - Real-Time Workshop pane: Data
    - Placement 5-24
  - Real-Time Workshop pane: Data Type
    - Replacement 5-43
  - Real-Time Workshop pane: Memory
    - Sections 5-72
  - Real-Time Workshop pane: Templates 5-12
- Configuration Parameters dialog box
  - Code Style pane
    - Parentheses level 5-5
    - Preserve condition expression in if statement 5-8
    - Preserve operand order in expression 5-7
  - Data Placement pane
    - Data declaration 5-29
    - Data declaration filename 5-31
    - Data definition 5-25
    - Data definition filename 5-27
    - #include file identifier 5-32
    - Module name 5-35
    - Module naming 5-33
    - Parameter tune level 5-39
    - Signal display level 5-37
  - Data Type Replacement pane
    - boolean Replacement Name 5-62
    - char Replacement Name 5-68
    - double Replacement Name 5-46
    - int Replacement Name 5-64
    - int16 Replacement Name 5-52
    - int32 Replacement Name 5-50
    - int8 replacement name 5-54
    - Replace data type names in the generated code 5-44
    - single Replacement Name 5-48
    - uint Replacement Name 5-66
    - uint16 Replacement Name 5-58
    - uint32 Replacement Name 5-56
    - uint8 Replacement Name 5-60
- Memory Sections pane
  - Constants 5-78
  - Execution 5-77
  - Initialize/Terminate 5-76
  - Inputs/Outputs 5-80
  - Internal data 5-82
  - Package 5-73
  - Parameters 5-84
  - Refresh package list 5-75
  - Validation results 5-86
- Real-Time Workshop (AUTOSAR Code Generation Options) 5-88
  - AUTOSAR Schema Version 5-89
  - Configure AUTOSAR Interface 5-90
- Templates pane
  - code templates: Header file (\*.h) template 5-14
  - code templates: Source file (\*.c) template 5-13
  - data templates: Header file (\*.h) template 5-16
  - data templates: Source file (\*.c) template 5-15
  - File customization template 5-17
  - Generate an example main program 5-18
  - Target operating system 5-20
- copyConceptualArgsToImplementation function 2-24
- createAndAddConceptualArg function 2-26
- createAndAddImplementationArg function 2-32
- createAndSetCImplementationReturn function 2-37
- createComponentAsSubsystem arxml.importer method 2-46
- Custom M-file block 4-2

**D**

Data Object Wizard block 4-4  
 Data Placement pane 5-22  
 Data Type Replacement pane 5-41

**E**

ERT (optimized for fixed-point) block 4-6  
 ERT (optimized for floating-point) block 4-8

**F**

function prototype control  
   addArgConf 2-12  
   attachToModel 2-23  
   getArgCategory 2-50  
   getArgName 2-53  
   getArgPosition 2-56  
   getArgQualifier 2-59  
   getDefaultConf 2-68  
   getFunctionName 2-72  
   getNumArgs 2-89  
   getPreview 2-93  
   RTW.configSubsystemBuild 2-122  
   RTW.getFunctionSpecification 2-143  
   RTW.ModelSpecificCPrototype 2-153  
   runValidation 2-173  
   setArgCategory 2-176  
   setArgName 2-180  
   setArgPosition 2-183  
   setArgQualifier 2-186  
   setFunctionName 2-192

**G**

getArgCategory method 2-48 2-50  
 getArgName method 2-51 2-53  
 getArgPosition method 2-54 2-56  
 getArgQualifier method 2-57 2-59

getCalibrationComponentNames  
   arxml.importer method 2-60  
 getClassName method 2-61  
 getComponentName AutosarInterface  
   method 2-62  
 getComponentNames arxml.importer  
   method 2-63  
 getDataTypePackageName AutosarInterface  
   method 2-64  
 getDefaultConf AutosarInterface method 2-65  
 getDefaultConf method 2-66 2-68  
 getDependencies arxml.importer method 2-69  
 getExecutionPeriod AutosarInterface  
   method 2-70  
 getFile arxml.importer method 2-71  
 getFunctionName method 2-72  
 getImplementationName AutosarInterface  
   method 2-73  
 getInitEventName AutosarInterface  
   method 2-74  
 getInitRunnableName AutosarInterface  
   method 2-75  
 getInterfacePackageName AutosarInterface  
   method 2-76  
 getInternalBehaviorName AutosarInterface  
   method 2-77  
 getIOAutosarPortName AutosarInterface  
   method 2-78  
 getIODataAccessMode AutosarInterface  
   method 2-79  
 getIODataElement AutosarInterface  
   method 2-80  
 getIOErrorStatusReceiver AutosarInterface  
   method 2-81  
 getIOInterfaceName AutosarInterface  
   method 2-82  
 getIOPortNumber AutosarInterface method 2-83  
 getIOServiceInterface AutosarInterface  
   method 2-84

- getIOServiceName AutosarInterface
  - method 2-85
- getIOServiceOperation AutosarInterface
  - method 2-86
- getNumArgs method 2-87 2-89
- getPeriodicEventName AutosarInterface
  - method 2-90
- getPeriodicRunnableName AutosarInterface
  - method 2-91
- getPortDefaultConf AutosarInterface
  - method 2-92
- getPreview method 2-93
- getStepMethodName method 2-94
- getTf1ArgFromString function 2-95
- GRT (debug for fixed/floating-point) block 4-10
- GRT (optimized for fixed/floating-point)
  - block 4-12

## I

- importer arxml.importer method 2-97

## M

- Memory Sections pane 5-70
- model entry points
  - model\_initialize 2-98
  - model\_SetEventsForThisBaseStep 2-100
  - model\_step 2-102
  - model\_terminate 2-105
- model\_initialize function 2-98
- model\_output function 2-104
- model\_SetEventsForThisBaseStep
  - function 2-100
- model\_step function 2-102
- model\_terminate function 2-105
- model\_update function 2-104
- models
  - parameters for configuring 5-102

## P

- parameters
  - for configuring model code generation and targets 5-102

## R

- registerCFunctionEntry function 2-106
- registerCPromotableMacroEntry
  - function 2-109
- RTW.configSubsystemBuild function 2-122
- rtw.connectivity.ComponentArgs 2-124
- rtw.connectivity.Config 2-126
- rtw.connectivity.ConfigRegistry 2-129
- rtw.connectivity.Launcher 2-134
- rtw.connectivity.MakefileBuilder 2-137
- rtw.connectivity.RtIOStreamHostCommunicator 2-139
- RTW.getEncapsulationInterfaceSpecification
  - function 2-141
- RTW.getFunctionSpecification function 2-143
- RTW.ModelCPPArgsClass class 2-144
- RTW.ModelCPPArgsClass constructor 2-147
- RTW.ModelCPPClass class 2-148
- RTW.ModelCPPVoidClass class 2-150
- RTW.ModelCPPVoidClass constructor 2-152
- RTW.ModelSpecificCPrototype class 2-153
- RTW.ModelSpecificCPrototype
  - constructor 2-156
- rtw.pil.RtIOStreamApplicationFramework 2-158
- runValidation AutosarInterface method 2-161
- runValidation method 2-169 2-171 2-173

## S

- setArgCategory method 2-174 2-176
- setArgName method 2-178 2-180
- setArgPosition method 2-181 2-183
- setArgQualifier method 2-184 2-186
- setClassName method 2-187



- setComponentName AutosarInterface
    - method 2-189
  - setDependencies arxml.importer method 2-190
  - setFile arxml.importer method 2-191
  - setFunctionName method 2-192
  - setInitEventName AutosarInterface
    - method 2-193
  - setInitRunnableName AutosarInterface
    - method 2-194
  - setIOAutosarPortName AutosarInterface
    - method 2-195
  - setIODataAccessMode AutosarInterface
    - method 2-196
  - setIODataElement AutosarInterface
    - method 2-197
  - setIOInterfaceName AutosarInterface
    - method 2-198
  - setPeriodicEventName AutosarInterface
    - method 2-199
  - setPeriodicRunnableName AutosarInterface
    - method 2-200
  - setReservedIdentifiers function 2-201
  - setStepMethodName method 2-204
  - setTflCFunctionEntryParameters
    - function 2-206
  - setTflCOperationEntryParameters
    - function 2-210
  - slConfigUIGetVal function 2-218
  - slConfigUISetEnabled function 2-220
  - slConfigUISetVal function 2-222
  - syncWithModel AutosarInterface method 2-224
- ## T
- targets
    - parameters for configuring 5-102
  - Templates pane 5-10
  - TFL table creation
    - addAdditionalHeaderFile 2-2
    - addAdditionalIncludePath 2-4
    - addAdditionalLinkObj 2-6
    - addAdditionalLinkObjPath 2-7
    - addAdditionalSourceFile 2-8
    - addAdditionalSourcePath 2-10
    - addConceptualArg 2-14
    - addEntry 2-16
    - copyConceptualArgsToImplementation 2-24
    - createAndAddConceptualArg 2-26
    - createAndAddImplementationArg 2-32
    - createAndSetCImplementationReturn 2-37
    - getTflArgFromString 2-95
    - registerCFunctionEntry 2-106
    - registerCPromotableMacroEntry 2-109
    - setReservedIdentifiers 2-201
    - setTflCFunctionEntryParameters 2-206
    - setTflCOperationEntryParameters 2-210