

RTI and RTI-MP

Implementation Reference

Release 2021-A – May 2021

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

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

© 1999 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Reference	9
----------------------	---

RTI Dialog Reference	13
----------------------	----

Model Configuration Parameters Dialogs.....	14
Solver Dialog (Model Configuration Parameters Dialogs).....	15
Math and Data Types Dialog (Model Configuration Parameters Dialogs).....	19
Diagnostics Dialog (Model Configuration Parameters Dialogs).....	19
Hardware Implementation Dialog (Model Configuration Parameters Dialogs).....	21
Model Referencing Dialog (Model Configuration Parameters Dialogs).....	22
Simulation Target Dialog (Model Configuration Parameters Dialogs).....	24
Code Generation Dialog (Model Configuration Parameters Dialogs).....	25
Model Parameter Configuration Dialog.....	38
RTI Task Configuration Dialog.....	39
RTI Task Configuration Dialog - Task Groups.....	41
Configure Initial Step Size Dialog.....	42
Signal Properties Dialog.....	44

RTI-MP Dialog Reference	45
-------------------------	----

Communication Channels Display.....	47
Change Connection Dialog.....	47
Multiprocessor Setup Dialog.....	48
Main Page (Multiprocessor Setup Dialog).....	49
CPU's Page (Multiprocessor Setup Dialog).....	53
Advanced Page (Multiprocessor Setup Dialog).....	57
Documentation Page (Multiprocessor Setup Dialog).....	59
RTI Task Configuration Dialog (Multiprocessor Setup Dialog).....	59
RTI Task Configuration Dialog - Task Groups (Multiprocessor Setup Dialog).....	61
Configure Initial Step Size Dialog (Multiprocessor Setup Dialog).....	62
CPU Options Dialog.....	64
Build Options Page (CPU Options Dialog).....	64
Variable Description File Options Page (CPU Options Dialog).....	67
Rename CPU Dialog.....	70
Multiprocessor Topology Setup Dialog.....	71

Model Configuration Parameters Dialogs.....	73
Math and Data Types Dialog (Model Configuration Parameters Dialogs).....	74
Diagnostics Dialog (Model Configuration Parameters Dialogs).....	75
Hardware Implementation Dialog (Model Configuration Parameters Dialogs).....	76
Model Referencing Dialog (Model Configuration Parameters Dialogs).....	78
Simulation Target Dialog (Model Configuration Parameters Dialogs).....	80
Code Generation Dialog (Model Configuration Parameters Dialogs).....	80
Model Parameter Configuration Dialog.....	85
Signal Properties Dialog.....	86

RTI and RTI-MP Variable Reference 89

currentTime.....	90
errorNumber.....	91
finalTime.....	92
modelStepSize.....	92
overrunCheckType.....	93
overrunCount.....	94
overrunQueueCount.....	95
overrunQueueMax.....	95
priority.....	96
sampleTime.....	97
simState.....	97
state.....	98
taskCallCount.....	99
turnaroundTime.....	100

RTI and RTI-MP File Reference 103

File Overview.....	104
Simulink Files.....	104
Files Generated by RTI and RTI-MP.....	105
Files Controlling the Build and Download Process.....	105
Executable Real-Time Object and System Description Files.....	107
Variable Access Files.....	108
User-Supplied Files.....	109
File Specifics for Model Referencing.....	110
File Details.....	112
startup.m.....	113
dsstartup.m.....	114

dspoststartup.m.....	114
dsfinish.m.....	115
rti1xxx_template_rtimphook.m.....	116
User-Code File (USR.C File).....	118
User Makefile (USR.MK File).....	119
Linker Command File (LK File).....	122
User System Description File (USR.SDF File).....	123
Variable Description File (TRC File).....	123
Variable Description File Groups.....	125
Available Variables in the Variable Description File.....	127
Data Type Specifications.....	131
User Variable Description File (USR.TRC File).....	132
SDF File Syntax.....	133
SDF File Syntax: Sections and Keys.....	133
Example SDF Files.....	137
Syntax of the TRC File.....	138
Principles of the TRC File.....	139
Grouping.....	139
Variable Names.....	141
Comments.....	142
Error File.....	142
Keywords.....	144
_author.....	145
_description.....	146
_floating_point_type().....	146
_gendate.....	147
_genname.....	147
_genversion.....	147
_integer_type().....	148
_model.....	148
endgroup.....	149
endstruct.....	149
enum.....	150
group.....	150
sampling_period[host_service_index].....	151
struct.....	153
typedef.....	154
Variable and Group Properties.....	154
addr.....	156
alias.....	158

array-incr.....	159
bitmask.....	159
block.....	160
default.....	160
desc.....	161
flags.....	161
increment.....	161
offs.....	162
origin.....	163
range.....	163
refelem.....	163
refgroup.....	165
refvar.....	166
scale.....	166
scaleback.....	167
type (Data Type, Data Format and Type Definition).....	167
unit.....	171
value.....	171
Examples.....	172
Example of Accessing Custom Variables in ControlDesk.....	172
Example of a TRC File Generated by RTI.....	175
RTI and RTI-MP Command Reference	183
ds_trc_multiplelabeloccurrence.....	184
rti1xxx.....	185
rti_build2.....	185
rti_mdldcleanup.....	189
rti_optionget.....	189
rti_optionset.....	191
rti_sdfmerge.....	193
rtimp_blktargetcpunameget.....	193
rtimp_build2.....	194
rtimp_targetswitch.....	197
rtiver.....	198
set_rti.....	198
RTI and RTI-MP Task Configuration API Reference	201
Task Configuration Classes.....	202
TaskManager.....	202

MainModelTaskManager.....	203
SubModelTaskManager.....	205
Task Configuration Methods.....	207
Commit.....	208
GetAllTasks.....	209
GetGroupTasks.....	210
GetModelTasks.....	211
GetSubModelNames.....	212
GetSubModelTaskManager.....	213
GetTaskGroupNames.....	213
GetTaskGroupPriority.....	214
GetTaskNamesByGroup.....	215
GetTaskOverrunBehavior.....	217
GetTaskOverrunQueueLength.....	218
GetTaskPriority.....	219
GetTaskSampleTime.....	220
IsTaskConfigurationChanged.....	221
SetOverrunQueueLength.....	222
SetTaskPriority.....	224
SetTaskGroupPriority.....	225
SetTaskOverrunBehavior.....	226
TurnInfoMessageOff.....	228
TurnInfoMessageOn.....	228

General RTI and RTI-MP Libraries 231

TaskLib Block Reference.....	232
Overview of the Task Configuration Blockset.....	233
Background Block.....	234
Hardware Interrupt Block.....	235
Function-Call Subsystem Block.....	237
Software Interrupt Block.....	238
Timer Interrupt Block.....	239
Time-Trigger Set Block.....	242
Time-Triggered Task Block.....	243
Timetable Start Block.....	244
Timetable Task Block.....	245
Timer Task Assignment Block.....	246
Extras Block Reference.....	248
Overview of the Extras Blockset.....	249
Data Capture Block.....	249

Ramp Generator for Encoder Index Search Block.....	251
simState READ Block.....	252
simState SET Block.....	253
TRC Exclusion Block.....	254
RTI-MP Blockset Reference.....	256
Real-Time Interface for Multiprocessor Systems.....	256
Overview of the RTI-MP Blockset.....	256
Configuration Blocks.....	258
Default CPU.....	259
Multiprocessor Setup.....	260
Interprocessor Communication Blocks.....	261
IPCx.....	261
IPC Receive.....	265
IPC Send.....	267
Interprocessor Interrupt Blocks.....	268
IPI.....	268
IPI Receive.....	269
IPI Send.....	270
RTI Gigalink Blockset Reference.....	272
Overview of the RTI Gigalink Blockset.....	272
Gigalink_Interrupt.....	274
Gigalink_Receive.....	275
Gigalink_Send.....	277
Gigalink_Status.....	279
 Index.....	 281

About This Reference

Content

This reference contains information on the various dialogs, files, options, etc. of Real-Time Interface (RTI and RTI-MP) for dSPACE systems. It also describes RTI-MP blocks and the RTI blocks that are common to all supported platforms. RTI acts as the link between Simulink and the dSPACE hardware. RTI-MP enables you to partition your model and allocate the parts to the different CPUs of a multiprocessor system.

All the descriptions and screenshots in this reference are based on MATLAB R2021a. Relevant differences to earlier MATLAB versions are mentioned in the descriptions.

Supported MATLAB versions RCP and HIL software from dSPACE Release 2021-A supports the following MATLAB versions:


- R2021a
- R2020b
- R2020a
- R2019b





Note

- For information on the compatibility of all products from dSPACE Release 2021-A with the current MATLAB release and MATLAB releases that came out later than dSPACE Release 2021-A, visit www.dspace.com/go/compatibility.
- This document describes Real-Time Interface for all hardware platforms. Not all sections apply to all platforms. You can skip all paragraphs that are not relevant to your platform.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.

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

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Examples:

- Where you find terms such as **rti<XXXX>** replace them by the RTI platform support you are using, for example, **rti1007**.
- Where you find terms such as **<model>** or **<submodel>** in this document, replace them by the actual name of your model or submodel. For example, if the name of your Simulink model is **smd_1007_s1.slx** and you are asked to edit the **<model>_usr.c** file, you actually have to edit the **smd_1007_s1_usr.c** file.

RTI block name conventions All I/O blocks have default names based on dSPACE's board naming conventions:

- Most RTI block names start with the board name.
- A short description of functionality is added.
- Most RTI block names also have a suffix.

Suffix	Meaning
B	Board number (for PHS-bus-based systems)
M	Module number (for MicroAutoBox II)
C	Channel number
G	Group number
CON	Converter number

Suffix	Meaning
BL	Block number
P	Port number
I	Interrupt number

A suffix is followed by the appropriate number. For example, DS2201IN_B2_C14 represents a digital input block located on a DS2201 board. The suffix indicates board number 2 and channel number 14 of the block. For more general block naming, the numbers are replaced by variables (for example, DS2201IN_Bx_Cy).

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files


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

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

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

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

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

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

RTI Dialog Reference

Introduction The various dialogs of Simulink and RTI provide many options to customize the behavior of Simulink and code generation for the dSPACE real-time platforms.

Where to go from here

Information in this section

Model Configuration Parameters Dialogs..... 14

To specify simulation parameters for single-processor models.

Solver Dialog (Model Configuration Parameters Dialogs)..... 15

To specify parameters for the start and stop times, choose a solver, and define the timer task mode for single-processor RTI.

Math and Data Types Dialog (Model Configuration Parameters Dialogs)..... 19

To set options for specifying data types and net slope computations.

Diagnostics Dialog (Model Configuration Parameters Dialogs)..... 19

To set options for making specific model conditions generate messages.

Hardware Implementation Dialog (Model Configuration Parameters Dialogs)..... 21

To set options for the characteristics of the hardware to be used to implement the system represented by your model.

Model Referencing Dialog (Model Configuration Parameters Dialogs)..... 22

To set options for including a model in other models and vice versa.

Simulation Target Dialog (Model Configuration Parameters Dialogs)..... 24



To set options for configuring the simulation target.

Code Generation Dialog (Model Configuration Parameters Dialogs)..... 25

To specify the RTI driver programs that control the automatic build process; to set different Code Generation and RTI simulation, build, variable description file, and download options.

Model Parameter Configuration Dialog.....	38
To declare specific parameters as tunable parameters.	
RTI Task Configuration Dialog.....	39
To assign priorities to the tasks of a model and configure their overrun strategy.	
RTI Task Configuration Dialog - Task Groups.....	41
If your model contains blocks of the RTI FlexRay Configuration Blockset or of the RTI LIN MultiMessage Blockset, the RTI Task Configuration dialog is extended.	
Configure Initial Step Size Dialog.....	42
To configure the increase of the initial step size.	
Signal Properties Dialog.....	44
To specify the properties of the selected signal.	

Model Configuration Parameters Dialogs

Access	Ribbon/Menu	Simulink model <ul style="list-style-type: none"> For the active configuration set: MODELING - SETUP - Model Settings or SIMULATION - PREPARE - Model Settings For model-specific and reference configuration sets: MODELING - DESIGN - Model Explorer
	Context menu of	None
	Shortcut key	Model Explorer: Ctrl+H Model Configuration Parameters: Ctrl+E
	Icon	Configuration Parameters:  Model Explorer: 


Description

The Model Configuration Parameters dialog lets you specify model-specific configuration parameter settings used for simulation and code generation for the active configuration set of your model. For further information on model-specific configuration sets, refer to [How to Specify RTI-Specific Settings for a Model \(RTI and RTI-MP Implementation Guide !\[\]\(950a62bbddad88d64435fd35607dfc42_img.jpg\)](#)).

If you create a new model, the model configuration parameters are initially set to the specified configuration defaults.

Dialog pages

This dialog contains the following pages:

- [Solver Dialog \(Model Configuration Parameters Dialogs\)](#) on page 15 for setting options for the start and stop times, choosing the Solver, and defining the timer task mode.
- [Data Import/Export Dialog \(Model Configuration Parameters Dialogs\)](#) for setting options for writing simulation data to a MAT file during Simulink simulations. The settings of this page are not relevant to RTI. The ability to capture data in real time for applications built by RTI is provided by dSPACE's ControlDesk. For details, refer to [ControlDesk Measurement and Recording](#) .
- [Math and Data Types Dialog \(Model Configuration Parameters Dialogs\)](#) on page 19 for specifying options relevant to data types and net slope computations.
- [Diagnostics Dialog \(Model Configuration Parameters Dialogs\)](#) on page 19 for setting options for making specific model conditions generate messages.
- [Hardware Implementation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 21 for specifying the characteristics of the hardware to be used to implement the system represented by your model.
- [Model Referencing Dialog \(Model Configuration Parameters Dialogs\)](#) on page 22 for specifying options relevant for simulation and code generation if your model includes other models or is included by another model.
- [Simulation Target Dialog \(Model Configuration Parameters Dialogs\)](#) on page 24 for setting options for debugging and inserting custom code.
- [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25 for specifying the RTI driver programs that control the automatic build process, and for setting different code generation and RTI simulation, build, variable description file, and download options.

Related topics

HowTos

[How to Adapt the Default Configuration for Model Properties \(RTI and RTI-MP Implementation Guide !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)\)](#)
[How to Specify RTI-Specific Settings for a Model \(RTI and RTI-MP Implementation Guide !\[\]\(c1e4487e48462435243c9e117557e045_img.jpg\)\)](#)

Solver Dialog (Model Configuration Parameters Dialogs)

Access

This dialog is contained in the **Model Configuration Parameters** dialog, refer to [Model Configuration Parameters Dialogs](#) on page 14.

Purpose

To specify parameters for the start and stop times, choose a solver, and define the timer task mode for single-processor RTI.

Description

In this dialog you can specify the solver-related parameters for real-time simulation and the Simulink simulation.

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Solver dialog and its options, refer to the MATLAB documentation by MathWorks®.

Dialog settings

Start time Lets you specify a start time for the simulation.

For real-time applications, the start time must always be set to **0.0** seconds.

Stop time Lets you specify a stop time for the simulation.

For real-time simulations you should specify **inf** (meaning infinity) as the stop time.

Type Specifies the type of solver to be used to solve the currently selected model, either Fixed-step or Variable-step.

Solver Lets you specify the solver type for calculating the model.

The higher the order of a solver, the higher its numerical precision and the more execution time it needs. With RTI you can use the following fixed-step solvers:

Solver	Description
discrete	No integration, suitable only for models without continuous states
ode1	Euler's method, suitable for most applications
ode2	Heun's method, also known as the improved Euler formula
ode3	The Bogacki-Shampine formula
ode4	The fourth-order Runge-Kutta formula
ode5	The Dormand-Prince (RK5) formula
ode8	The Dormand-Prince (RK8(7)) formula
ode14x	Combination of Newton's method and extrapolation from the current value
ode1be ¹⁾	Backward Euler method with additional setting for the number of Newton's iterations.

¹⁾ Only available as of MATLAB R2020a.

Fixed step size (fundamental sample time) Lets you specify the base sample time for the model. If your model includes continuous states, the fixed step size is also used for the integration method. The fixed step size has to be specified in seconds.

Note

The simulation step size during the real-time simulation and the fixed step size selected in the model might differ, depending on the resolution of the timer device that provides the simulation step size. This difference is a discretization error resulting from the approximation of the fixed step size by an integral multiple of the timer resolution. The discretization error is always smaller than the timer resolution and can be minimized by selecting a fixed step size that is an integral multiple of the timer resolution.

Consider the following example:

- Timer resolution = 0.015 μ s
- Fixed step size = 25 μ s

In this case the simulation step size is given by:

$$\text{floor}(25 \mu\text{s} / 0.015 \mu\text{s}) \cdot 0.015 \mu\text{s} = 24.99 \mu\text{s}$$

For further information on the timer devices, refer to [Timer Interrupt Block](#) on page 239.

Periodic sample time constraint Lets you specify constraints on the sample times defined by the model. For RTI, the following options are suitable:

Option	Description
Unconstrained	Specifies no constraints. Requires the specification of a base sample time for the model.
Ensure sample time independent	Ensures that the model is independent of a concrete sample time or a specific solver. Thus, the model can inherit its sample time from the context in which it is used, e.g., when it is referenced by another model. When selecting this option, Simulink disables the following solver options: <ul style="list-style-type: none"> ▪ Treat each discrete rate as a separate task ▪ Automatically handle rate transition for data transfer ▪ Higher priority value indicates higher task priority During simulation, Simulink checks to ensure that the model satisfies the constraints. If this is not the case, an error message is displayed.

Treat each discrete rate as a separate task Lets you specify the execution mode for calculating your model.

- If this option is set, the multi-tasking mode is enabled. The application contains the same number of timer tasks as it contains sample rates. Regardless of this option, the single-tasking mode is used in the following cases:
 - Your model contains one sample time.
 - Your model contains a continuous and a discrete sample time, and the fixed step size is equal to the discrete sample time.
- If this option is cleared, the single-tasking mode is enabled. The application contains just one timer task. This is the default setting.

For details on both execution modes, refer to Single Timer Task Mode, Multiple Timer Task Mode and Comparing the Execution Modes.

Allow tasks to execute concurrently on target This option is not supported by RTI. Make sure that it is not selected. Concurrent execution of tasks can be done by using RTI-MP for running tasks on multiprocessor or multicore systems.

Automatically handle rate transition for data transfer Lets you specify if Simulink automatically inserts hidden rate transition blocks between blocks that have different sample rates.

- If this checkbox is selected, Simulink inserts hidden rate transition blocks when rate transitions for asynchronous or periodic tasks are detected.
- If this checkbox is not selected (default), hidden rate transition blocks are not inserted when rate transitions are detected.

Deterministic data transfer Lets you control the level of data transfer determinism for periodic tasks. This parameter is available only if the **Automatically handle rate transition for data transfer** option is enabled. You can choose between the following options:


Option	Description
Whenever possible (default)	Specifies that the Ensure deterministic data transfer (maximum delay) block parameter is set for auto-inserted rate transition blocks whenever possible.
Always	Specifies that the Ensure deterministic data transfer (maximum delay) block parameter is always set for auto-inserted rate transition blocks.
Never (minimum delay)	Specifies that the Ensure deterministic data transfer (maximum delay) block parameter is never set for auto-inserted rate transition blocks.

For details on the **Ensure deterministic data transfer (maximum delay)** block parameter, refer to the MATLAB documentation by MathWorks.

Higher priority value indicates higher task priority If this checkbox is selected, the real-time system targeted by this model assigns a higher priority to tasks with higher priority values. If this checkbox is cleared, the real-time system targeted by this model assigns a higher priority to tasks with lower priority values. For RTI, this checkbox must be cleared.

Related topics

HowTos

[How to Adapt the Default Configuration for Model Properties \(RTI and RTI-MP Implementation Guide\)](#) 

Math and Data Types Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 14.
Purpose	To set options for specifying data types and net slope computations.
Description	In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Math and Data Types dialog and its options, refer to the MATLAB documentation by MathWorks®.
Dialog settings	<p>Implement logic signals as Boolean data (vs. double) Lets you specify whether the boolean or double data type is used for logical signals. This option can be used for compatibility with models from earlier Simulink versions where logical signals are implemented as double.</p> <ul style="list-style-type: none"> ▪ If selected, the Simulink blocks (e.g., Logical Operator) use the Boolean data type for logical signals (default). ▪ If cleared, the Simulink blocks in the model use the double data type for logical signals. <p>For reasons of downward compatibility, this setting is also respected by some RTI Bit I/O blocks.</p> <p>For general information, refer to the Simulink documentation.</p>

Diagnostics Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 14.
Purpose	To specify options that make specific model conditions generate either error messages or warnings.
Description	<p>On this page you can specify options to check the overall RTI model and the code generation process. The Diagnostics dialog contains the following pages:</p> <ul style="list-style-type: none"> ▪ The Solver page (Main page) lets you specify the diagnostic actions Simulink takes when it detects a solver-related error. ▪ The Sample Time page lets you specify the diagnostic actions Simulink takes when it detects an error relating to the sample times in your model.

- The **Data Validity** page lets you specify the diagnostic actions Simulink takes when it detects an error that could have an impact on the data integrity of your model.
- The **Type Conversion** page lets you specify the diagnostic actions Simulink takes when it detects a data type conversion error during compilation of your model.
- The **Connectivity** page lets you specify the diagnostic actions Simulink takes when it detects an error relating to block connections.
- The **Compatibility** page lets you specify the diagnostic actions Simulink takes when it detects an error relating to compatibility between the current Simulink version and your model.
- The **Model Referencing** page lets you specify the diagnostic actions Simulink takes when it detects an error relating to Model Referencing.
- The **Stateflow** page lets you specify the diagnostic actions Simulink takes when it detects an error relating to a Stateflow configuration.

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Diagnostics dialog and its options, refer to the *Using Simulink* by MathWorks®.

Data Validity page

Signal resolution Lets you select how Simulink software resolves signals to Simulink.Signal objects. The following settings are possible:

Setting	Description
Explicit only (default)	Performs only explicitly specified signal resolution. This is the recommended setting.
Explicit and implicit	Performs implicit signal resolution wherever possible, without posting any warnings about the implicit resolutions.
Explicit and warn implicit	Performs implicit signal resolution wherever possible, posting a warning of each implicit resolution that occurs.

Model Verification block enabling Lets you globally enable or disable all the Model Verification blocks or use the settings of the individual blocks.

- If *Enable all* is selected, all the **Model Verification** blocks are active.
- If *Disable all* is selected, none of the **Model Verification** blocks is active.
- If *Use local settings* is selected, only the **Model Verification** blocks with the *Enable assertion* setting selected are active (default).

To control the behavior of the Model Verification blocks in the real-time simulation, use RTI's Assertion mode setting. Depending on how you set this, one of the following actions is carried out if a verified signal leaves the desired range, meaning that the assertion occurs:

- OFF: No reaction (default).
- WARN: Issue a warning message in the Log Viewer of ControlDesk.
- STOP: Issue an error message and stop the simulation, i.e., set the simulation state to *STOP*.

For details on this technique, refer to [How to Use Simulink's Model Verification Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(dd161862f9164df98f62b726e9846241_img.jpg\)](#)).

Hardware Implementation Dialog (Model Configuration Parameters Dialogs)

Access This dialog is contained in the **Model Configuration Parameters** dialog, refer to [Model Configuration Parameters Dialogs](#) on page 14.

Purpose To set options for the characteristics of the hardware to be used to implement the system represented by your model.

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the **Hardware Implementation** dialog and its options, refer to the Simulink documentation.

Description The **Hardware Implementation** dialog lets you specify the characteristics of the hardware to be used to implement the system represented by your model. All the configuration settings are contained in the default model template that is used if you create a new model.

The following values are then automatically configured on the **Hardware Implementation** dialog, depending on your dSPACE board:

Property	DS1007 DS1104 MicroAutoBox II MicroLabBox	DS1006
Device vendor	Custom Processor	Custom Processor
Number of bits		
char	8	8
short	16	16
int	32	32
long	32	32
long long ¹⁾	64	64
native	32	32
pointer	32	32
size_t	32	32
ptrdiff_t	32	32
Byte ordering	Big Endian	Little Endian
Signed integer division rounds to	Zero	Zero
Shift right on a signed integer as arithmetic shift	True	True

¹⁾ To enable this data type, you must set the **Support long long** option in the dialog.

If the **Ensure deterministic data transfer** option of the **Rate Transition** block is cleared, and the signal size is less than or equal to the specified largest atomic size, the code generator removes double-buffering and semaphore protection

from the generated code. This optimization reduces memory consumption and improves execution speed.

RTI and RTI-MP check if the hardware implementation settings are specified correctly.

For the largest atomic sizes of integer and floating-point variables, dSPACE only supports the following combinations:

- Largest atomic size of integer variables: **Int**
- Largest atomic size of floating-point variables: **Float**

Note

To enable generation of 64-bit integer variables into the TRC file, you must set the Support long long option.

Model Referencing Dialog (Model Configuration Parameters Dialogs)

Access This dialog is contained in the Model Configuration Parameters dialog, refer to [Model Configuration Parameters Dialogs](#) on page 14.

Purpose To set options for including a model in other models and vice versa.

Description In this dialog you can specify options for building simulation (SIM) and Simulink Coder (RTW) targets relevant when:

- including this model in other models
- including other models in this model

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Model Referencing dialog and its options, refer to the Simulink documentation.

Options for all referenced models **Rebuild** Lets you determine whether SIM and RTW targets are rebuilt before updating, simulating, or generating code. The following options are available:

Option	Description
Always	SIM and RTW targets are always rebuilt before updating, simulating, or generating code from the model.
If any changes detected	SIM and RTW targets are rebuilt if Simulink detects any changes of any kind in the target's dependencies. This is the default setting.

Option	Description
If any changes in known dependencies	SIM and RTW targets are rebuilt if Simulink detects any changes in known dependencies since the target was last built. You can define dependencies via the Model dependencies option described below.
Never	SIM and RTW targets are never rebuilt before updating, simulating, or generating code from the model.

Tip

Use the *Never* option carefully since it may lead to incorrect results if referenced targets are not really up-to-date.

Never rebuild diagnostic Lets you specify the diagnostic action that Simulink should perform if it detects a target that needs to be rebuilt. This option is available only if you set the **Rebuild** option to *Never*. You can select the following options:

- Error if rebuild required (default)
- Warn if rebuild required
- None

Note

Selecting *None* accelerates updating, simulation and code generation, but can generate invalid results.

Enable parallel model reference builds Lets you specify whether to build the referenced models in parallel on a multicore PC. To enable the parallel build feature the Parallel Computing Toolbox is required.

MATLAB worker initialization for builds Lets you specify how the MATLAB workers will be initialized prior to building on them. It is enabled if you set the **Enable parallel model reference builds** option. You can choose *None* (default), *Copy base workspace* and *Load top model*.

Options for referencing this model

Total number of instances allowed per top model Lets you specify how often the model can be referenced in another model. You can choose between one, multiple (default) or zero instances. This setting influences the appearance of referenced models in the variable description file (TRC file). For details, refer to [Available Variables in the Variable Description File](#) on page 127.

With RTI-MP, the number of instances refers to the main model and each of the submodels.

Propagate sizes of variable-size signals Lets you specify when to propagate the sizes of variable-size signals. You can choose *Infer from blocks in model*, *Only when enabling* and *During execution*.

Minimize algebraic loop occurrences Lets you specify whether Simulink should eliminate algebraic loops involving the model from models that reference it.

Pass fixed-size scalar root inputs by value for code generation Lets you specify how a model that references this model passes scalar inputs to this model. If enabled, scalar inputs to this model are passed by value. Otherwise, scalar inputs are passed by reference (default). Passing roots by value can have advantages in performance. It can lead to incorrect simulation results, though.

Model dependencies Lets you specify on which files the model relies. They are typically `.mat` and `.m` files used to initialize and provide data. This information is used by Simulink and Simulink Coder to detect if the model must be rebuilt.

Related topics

Basics

[Limitations with Model Referencing \(RTI and RTI-MP Implementation Guide !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)\)](#)
[Referencing Models \(RTI and RTI-MP Implementation Guide !\[\]\(e658400d40ca763c7cf4c8c420885c6a_img.jpg\)\)](#)

Simulation Target Dialog (Model Configuration Parameters Dialogs)

Access This dialog is contained in the **Model Configuration Parameters** dialog, refer to [Model Configuration Parameters Dialogs](#) on page 14.

Purpose To set options for configuring the simulation target for a model that contains, e.g., MATLAB Function blocks.

Description In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the **Simulation Target** dialog and its options, refer to the MATLAB documentation by MathWorks®.

Dialog settings

Block reduction This option is not supported by RTI. Make sure that it is not selected.

Conditional input branch execution Lets you specify whether to execute only the blocks required to compute the control input and the data input selected by the control input of Switch and Multiport Switch blocks. If the option is set, ControlDesk display the last computed values of the output signals of those blocks, which are executed because of the current switch condition. To ensure that the blocks providing signals connected to your experiment software are computed, turn off the **Conditional input branch execution** option and rebuild the model.

Code Generation Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 14.
Purpose	To specify the RTI driver programs that control the automatic build process; to set different Code Generation and RTI simulation, build, variable description file, and download options.
Description	<p>On this page you can specify options relating to the code generation process. The following settings are relevant to RTI. For the settings relevant to RTI-MP, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) on page 80.</p> <p>For a detailed description of the Code Generation page and its options, refer to the Simulink documentation.</p>
Dialog settings	<p>The Code Generation dialog contains several pages. The following pages contain settings that are relevant to RTI:</p> <ul style="list-style-type: none"> ▪ General page on page 25 ▪ Optimization page on page 27 ▪ Identifiers page on page 28 ▪ Custom Code page on page 29 ▪ Interface page on page 29 ▪ RTI simulation options page on page 30 ▪ RTI general build options page on page 33 ▪ RTI load options page on page 34 ▪ RTI variable description file options page on page 35 <div> <p>Tip</p> <p>You can use the <code>rti_optionget</code> and <code>rti_optionset</code> commands to get or set the options supported by the RTI options pages. Refer to rti_optionget on page 189 and rti_optionset on page 191.</p> </div>
General page	<p>System target file Lets you specify the target file for your model used during code generation via the System Target File Browser. For example, to generate</p>

code for a DS1007 hardware platform select `rti1007.tlc`. For information on the system target file, refer to [rti<xxx>.tlc](#) on page 105.

Generate code only Lets you generate only C code from your Simulink model.

- If selected, the model code is only generated. The code is neither compiled nor downloaded to the hardware. Files that are relevant to the generated real-time application, such as the TRC file, are not generated.
- If cleared, the model code is generated, compiled, and downloaded to the dSPACE hardware (default).

Template makefile This option is automatically adapted by RTI corresponding to the actual System target file setting, when the system target file is selected. You do not need to change this setting manually. For information on the related files, refer to [rti<xxx>.tmf](#) on page 105, [<\[sub\]model>.mk](#) on page 106, and [<\[sub\]model>_usr.mk](#) on page 106.

Make command This option is automatically adapted by RTI corresponding to the actual System target file setting when the system target file is selected. You do not need to change this setting manually.

Select objective Lets you select code generation objectives to be used with the Code Generation Advisor. The following settings are available:

Setting	Description
Unspecified	Specifies no objective (default). Do not optimize code generation settings using the Code Generation Advisor.
Debugging	Specifies a debugging objective. Optimize code generation settings for debugging the code generation build process using the Code Generation Advisor.

Check model before generating code Lets you choose whether to run Code Generation Advisor checks before generating code. The following settings are available:

Option	Description
Off	Generates code without checking whether the model meets code generation objectives (default).
On (proceed with warnings)	Checks whether the model meets code generation objectives. If the Code Generation Advisor reports a warning, they are displayed in the Model Advisor Window. The build process continues.
On (stop for warnings)	Checks whether the model meets code generation objectives. If the Code Generation Advisor reports a warning, they are displayed in the Model Advisor Window. The build process stops.

Check Model Lets you run Code Generation Adviser checks.

Note

Code Generation Adviser checks are general code generation checks. Running these checks does not necessarily check model settings that are relevant to RTI. RTI checks model settings relevant to RTI automatically during the build process, independently of any checks performed by the Code Generation Adviser.

You can start the build process via the **Build Model** command in the **Code - C/C++ Code** menu. For an overview of the build process, refer to [Introducing the Build and Download Process \(RTI and RTI-MP Implementation Guide\)](#).

Optimization page

Default parameter behavior Lets you inline the block parameters or generate them as variables.

- If **Inlined** is selected, the block parameters are inlined, so their concrete values are used in the generated C code, which makes them non-modifiable. This typically reduces memory consumption and execution time.

To generate individual parameters as tunable variables, see the **Configure** option below. Simulink.Parameter objects in the workspace can also be used to specify the tunability of a parameter, refer to *Rules for Simulink.Signal and Simulink.Parameter objects* in [Available Variables in the Variable Description File](#) on page 127.

- If **Tunable** is selected, the block parameters are generated as variables, so they are modifiable during the real-time simulation (default).

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Configure If the **Default parameter behavior** option is set to **Inlined**, you can make individual parameters tunable.

If the **Default parameter behavior** option is set to **Tunable**, you can specify the storage class for individual parameters.

For details, refer to [Model Parameter Configuration Dialog](#) on page 38.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Use memcpy for vector assignment Lets you optimize code generated for vector assignment by replacing for-loops with `memcpy` function calls.

- If the checkbox is selected, the associated **Memcpy threshold (bytes)** parameter is enabled (default). `memcpy` is used in the generated code if the number of array elements times the number of bytes per element is greater than or equal to the specified value for the **Memcpy threshold (bytes)** parameter. One byte equals the width of a character.
- If the checkbox is cleared, the use of `memcpy` for vector assignment is disabled.

Memcpy threshold (bytes) Lets you specify the array size in bytes at or above which `memcpy` function calls should replace for-loops in the generated code for vector assignment. The default array size is 64.

Signal storage reuse Lets you control the memory allocation for block output variables.

- If selected, Simulink Coder does not always use a separate block output variable for each block output but attempts to assign a single buffer to several block outputs. This typically reduces memory consumption. However, reused variables are not available in the variable description file.

You can exclude individual block outputs from this optimization to make them available in the variable description file.

- If cleared, a separate global variable is used for each block output (default).

For details, refer to the Simulink documentation.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Remove code from floating point to integer conversions with saturation that maps NaN to zero Lets you determine if code should be removed when mapping from NaN values to integer zero occurs.

- If the checkbox is selected, code is removed when NaN values are mapped to integer zero (default).
- If the checkbox is cleared, code is not removed when NaN values are mapped to integer zero.

Note

Selecting the checkbox produces results that do not match the simulation in the case of NaN values. If the checkbox is cleared, the results for the simulation and the execution of generated code match.

Use memset to initialize floats and doubles to 0.0 Lets you determine whether code is generated that explicitly initializes floating-point data to 0.0.

- If the checkbox is selected, `memset` is used to clear internal storage for floating-point data to integer bit pattern 0 (all bits 0), regardless of type (default).
- If the checkbox is cleared, code is generated that explicitly initializes storage for data of types float and double to 0.0. The generated code is slightly less efficient than code generated when you select the checkbox.

Identifiers page

Maximum identifier length Lets you specify the maximum number of characters used in functions, type definitions, and variable names in the range 31 ... 256. The default length is 31.

Use the same reserved names as Simulation Target Lets you specify whether to use the same reserved names as those defined for the simulation

target. Reserved names are names of variables or functions in the generated code that match the names of variables or functions specified in custom code.

- If the checkbox is selected, the same reserved names as those defined for the simulation target are used. When selected, this parameter disables the Reserved names parameter.
- If the checkbox is cleared, the reserved names defined for the simulation target are not used (default).

Reserved names Lets you enter reserved names. This action changes the names of variables or functions in the generated code to avoid name conflicts with identifiers in custom code. Reserved names must be shorter than 256 characters.

Custom Code page

Insert custom C code in generated: Lets you insert code fragments into the generated files.

Note

You can use the settings in the Insert custom C code in generated frame in Code Generation's Custom Code page in addition to the settings in the RTI User-Code file (USR.C).

Additional build information: Lets you include additional files and paths in the build process.

Note

- You can use the settings in the Additional build information frame in Code Generation's Custom Code page instead of, or in addition to, the settings in the RTI User Makefile (USR.MK).
- The Defines option is not supported by RTI and RTI-MP. Use the Compiler options setting in the RTI general build options page of the Code Generation Dialog.

Interface page

Standard math library Lets you specify which floating-point math library to use during code generation. The default value is **C99 (ISO)**. To avoid incompatibilities of the generated code with target compilers that do not support the C99 standard, RTI automatically switches to **C89/90 (ANSI)** when a dSPACE platform is selected.

Array layout Lets you specify whether the first index of a matrix indicates the column or the row.

RTI and RTI-MP currently supports only the **Column-major** layout.

External functions compatibility for row-major code generation Lets you specify how to react to incompatible functions for a row-major array layout.

For RTI and RTI-MP, which currently do not support the row-major array layout, this setting is set to **error**.

RTI simulation options page

All options described in the following are ignored for referenced models.

Initial simulation state Lets you define the initial simulation state for a model.

- If set to *RUN*, the simulation is started right after the download (default).
- If set to *PAUSE*, the simulation is initialized after the download but not started. The output devices are set to the Initialization values.

Note

The *PAUSE* simulation state is not supported by DS1007 and MicroLabBox (DS1202).

- If set to *STOP*, the simulation remains stopped after the download. The output devices are set to the Termination values.

The current simulation state of the real-time simulation is available in the variable description file (see [simState](#) on page 97).

Note

When you use RTI1007 or RTI1202, starting and stopping the simulation is controlled by the **CmdLoader** option. Rebuilding the real-time application is therefore not required to change the initial simulation state.

Execution mode Lets you specify whether the model should be calculated in real-time or non real-time execution mode.

- If *real-time* is selected, the model is calculated in real-time (default).
- If *time-scaled* is selected, the simulation is performed faster or more slowly, meaning that calculating one second in the simulation takes less or more than one real second.

You have to provide a Time scale factor (see below).

- If *as fast as possible* is selected, the simulation is performed as fast as possible without a specific time scale factor.

Note

The *as fast as possible* option is not supported by DS1007 and MicroLabBox (DS1202).

To execute your model in the *time-scaled* or *as fast as possible* execution modes, you have to build it in the single timer task mode. You can still use hardware and software interrupts.

In the multiple timer task mode, RTI only supports the *real-time* execution mode. The *time-scaled* and *as fast as possible* execution modes are not supported.

Time scale factor Lets you specify a time scale factor for the *time-scaled* execution mode. (Enabled only if the *time-scaled* execution mode is selected, see above.)

It must be a decimal value greater than 0.0.

For example, if you specify the *time-scaled* execution mode and time scale factor **0.2**, the simulation runs 5 times faster than in the *real-time* execution mode. A time scale factor of **1** corresponds to a real-time simulation. This is the default value of the time scale factor.

Assertion mode RTI supports Simulink's Model Verification blocks for real-time simulation.

To control the behavior of the Model Verification blocks in the real-time simulation, use RTI's Assertion mode setting. Depending on how you set this, one of the following actions is carried out if a verified signal leaves the desired range, meaning that the assertion occurs:

- OFF: No reaction (default).
- WARN: Issue a warning message in the Log Viewer of ControlDesk.
- STOP: Issue an error message and stop the simulation, i.e., set the simulation state to *STOP*.

For more information, refer to [How to Use Simulink's Model Verification Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(2b376d1a92330ab09dad2665d2f89bf5_img.jpg\)](#)).

Jitter and latency optimization Lets you select the task mode for the generated real-time application to optimize its latency for the different use cases.

Setting	Meaning
Standard (full functionality) tasks	The task execution is based on threads, which lets you include C++ and third-party code. The standard task mode is recommended for a task rate not faster than 1 ms.
Low jitter, low latency tasks	This is the default setting, which correlates to the task execution behavior of real-time applications generated with earlier dSPACE Releases. The task execution is based on interrupts, which lets you achieve a low trigger-to-task latency. The low jitter task mode is especially recommended for a task rate faster than 0.05 ms. Note: The support of C++ and third-party code is limited in low jitter task mode.

Note

- This option is only available for RTI1007 and RTI1202.
- When you use a DS1007 Processor Board or a MicroLabBox, you can configure to use the specified task mode when building the real-time application or to overwrite the specified task mode with the configuration in the board's web interface.

For more information, refer to:

- [Basics on the Web Interface \(DS1007 Hardware Installation and Configuration Guide !\[\]\(48a7667d09d5a06397e047ee4537bb6f_img.jpg\)\)](#)
- [Basics on the Web Interface \(MicroLabBox Hardware Installation and Configuration !\[\]\(3df135a685d1b545c4fa64a5f3516545_img.jpg\)\)](#)

If you use RTI-MP, you can specify the jitter and latency optimization in the Multiprocessor Setup block, refer to [Build Options Page \(CPU Options Dialog\)](#) on page 64.

Enable real-time testing Lets you enable or disable real-time testing for the generated real-time application.

To improve the startup time of the real-time application, the build process generates a binary TRZ file containing the MAP file and the TRC file.

Note

- This option is not available for RTI1104.
- If you want to use the Signal Generator or the Bus Navigator in ControlDesk, you have to set the Enable real-time testing option in your Simulink model. For new models this option is set by default. If the Enable real-time testing option is set, the size of the built real-time application increases. The time for loading the application from the flash memory therefore will be increased. In worst case, the rebuilt real-time application cannot be loaded because the board's memory is not sufficient.
- If you do not need the real-time testing support, clear the Enable real-time testing option in your model. For RTI1007 and RTI1202, this option is always set and cannot be modified.

Task Configuration Lets you assign priorities to the tasks of a model and specify their overrun behavior.

This button opens the RTI Task Configuration dialog, which lets you configure the tasks of a model. Refer to [RTI Task Configuration Dialog](#) on page 39.

Configure Initial Step Size Lets you configure the increase of the initial step size.

This button opens the Configure Initial Step Size dialog, which lets you configure the coefficients for the initial simulation steps increase and for the number of increased simulation steps. Refer to [Configure Initial Step Size Dialog](#) on page 42.

RTI general build options page

If you use model referencing, note the following:

- The Enable data set storage in application option is ignored for referenced models.

Compiler options Lets you specify the compiler options.

For example, if you want the compiler to produce debugging information.

Specify the following compiler option: `-g`

Compiler optimizations Lets you specify the compiler optimization level.

- If *Default* is selected, the model is compiled with the platform- and compiler-specific default optimization, which is the highest recommended level of optimization.
- If *User defined* is selected, you can define the optimization via the User-defined optimizations edit field.
- If *None* is selected, the compilation is performed without any optimization.

User-defined optimizations Lets you specify user-defined compiler options. (Available only if compiler optimizations is set to *User-defined*.)

- If empty, the model code is compiled with the default optimization setting of the compiler.
- Do not specify optimization levels higher than the default.

Do not specify compiler options that have an effect on data widths and memory alignments.

For example, do not use the following options:

- Microtec PowerPC Compiler:
 - `-KP`
 - `-Zm`
 - `-Zn`
 - Do not use the **packed** prefix for struct definitions within your source code for variables that will be included in the variable description file.
- GNU GCC C Compiler:
 - `-fpack-struct`
 - `-fshort-enums`
 - Do not use the `__attribute__ ((__packed__))` directive for struct and enum definitions within your source code for variables that will be included in the variable description file.

For information about compiler options, refer to the following manuals:

Board	Processor Type	Compiler Manual
DS1104 MicroAutoBox II	PowerPC	<i>C/C++ Compiler User's Guide by Microtec</i>

Board	Processor Type	Compiler Manual
DS1007 MicroLabBox	PowerPC	http://gcc.gnu.org/onlinedocs/gcc-4.8.3/gcc
DS1006	x86	http://gcc.gnu.org/onlinedocs/

Enable data set storage in application Lets you enable the storing of new data sets in an application image file.

- If the checkbox is selected, you can use ControlDesk to create a new application image file containing a new data set without rebuilding the application.
- If the checkbox is cleared, you cannot use ControlDesk to create a new application image file containing a new data set without rebuilding the application (default).

This option is ignored for models which are referenced by other models.

Note

- This option is only available for RTI1007, RTI1202 and RTI1401.
- For RTI1007 and RTI1202, the data set storage feature is always active. The checkbox for this option is therefore not available.

Allow usage of reserved host service numbers with Data Capture blocks

By default, the host services 28 to 31 are reserved for monitoring features when using bus support, for example, when using CAN or LIN. If you do not use a bus support, you can enable the reserved services by setting this option.

RTI load options page

All options described in the following are ignored for models which are referenced by other models.

Load application after build Loads the real-time application to the dSPACE hardware after it is built.

- If selected, the real-time application is automatically downloaded to the dSPACE hardware after it is built.
- If cleared, the real-time application is not downloaded automatically.

Load to Flash memory Lets you download a real-time application to the Flash memory.

- If selected, the real-time application is downloaded to the Flash memory (nonvolatile memory).
- If cleared, the real-time application is downloaded to the RAM (default).

Platform name Lets you enter the name of the platform used for downloading the real-time application.

You must enter the same name that you specified in ControlDesk's Platform Manager when you registered the platform.

For a new model or when you change the RTI target of a model, RTI selects the following default platform names for the related RTI targets:

- ds1006
- ds1007
- ds1104
- ds1202 (for MicroLabBox)
- ds1401 (for MicroAutoBox II)

Connection identification by This option is available for DS1007, MicroAutoBox II, and MicroLabBox (DS1202).

Lets you select whether to identify the connected platform by its platform name or by its IP address (network client).

Network client This option is available for DS1007, MicroAutoBox II, and MicroLabBox (DS1202).

Lets you specify the network client used for downloading the real-time application. To enter the IP address of the network client, you have to set the **Connection identification by** setting to *Network Client*. Using this option, you can start the download without DS1007, MicroAutoBox II, or MicroLabBox registered in ControlDesk.

RTI variable description file options page

If you use model referencing, note the following:

- The **Include only Simulink.Parameter and Simulink.Signal objects with global storage class** option can be used with model referencing. It must be the same for the top-level model and referenced models.
- The **Variable description file format** option must be the same for the top-level model and referenced models.
- The **Include states** option and the **Include derivatives** option are both not supported for model referencing.
- All the other options described below are supported for top-level and referenced models.

Variable description file format Lets you specify the variable description file format to be generated during the build process. By default, the TRC file format is generated. By selecting TRC and A2L in the list, you can additionally generate an A2L file.

This option corresponds to the VDFTYPE model parameter.

Include only Simulink.Parameter and Simulink.Signal objects with global storage class Lets you significantly reduce the time needed for code generation as it disables all the other variable description file options. It includes only parameters and signals in the variable description file which reference a Simulink.Parameter or a Simulink.Signal object in the MATLAB workspace. If you set this option, the variable description file only provides the Labels group, the Tunable Parameters group and the BusSystems group of the RTI CAN MultiMessage Blockset and the RTI LIN MultiMessage Blockset. It does not

contain the entire model hierarchy. For the Simulink.Parameter and Simulink.Signal objects the following must apply:

- They must specify a global storage class (ExportedGlobal, ImportedExtern, ImportedExternPointer).
- For Simulink.Signal objects:
 - Either the **Signal must resolve to Simulink.Signal** object option is selected in the signal's **Signal Properties** dialog.
 - Or the **Signal resolution** option is set to **Explicit and implicit** on the **Data Validity** page of the **Diagnostics** dialog.

In the variable description file, **Description**, **DocUnits**, **Min** and **Max** entries of Simulink.Signal objects and Simulink.Parameter objects are available.

Include signal labels Lets you make the signal labels of the model available in the variable description file.

- If selected, the signal labels of the model are generated into the variable description file.
- If cleared, the signal labels of the model are not generated into the variable description file. For large models especially, this might significantly reduce the time needed for the code generation process.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include virtual blocks Lets you make the block outputs of virtual blocks (Demux, From, etc.) available in the variable description file.

- If selected, the virtual blocks are available in the variable description file.
- If cleared, the virtual blocks are not available in the variable description file. This might significantly reduce the time needed for the code generation process.

If you just want to have the outputs of a few virtual blocks available in the variable description file, you can add test points, or **Signal Conversion** blocks with the **Output** property set to *Signal copy*. These outputs are available in the variable description file, even if you turn off the generation of virtual blocks.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include states Lets you make the states of blocks available in the variable description file.

- If selected, the block states are generated into the variable description file.
- If cleared, the block states are not generated into the variable description file (default).

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include derivatives Lets you make the derivatives of blocks available in the variable description file.

- If selected, the derivatives of blocks are generated into the variable description file.
- If cleared, the derivatives of blocks are not generated into the variable description file (default).

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include initial parameter values To make the initial parameter values available in the variable description file.

- If selected, the initial parameter values of the model are generated into the variable description file. This lets you carry out offline calibration for your model with ControlDesk.

This does not apply to:

- Stateflow parameters of the **State Machine Data** group
- If cleared, the initial parameter values of the model are not generated into the variable description file. For large models especially, this can significantly reduce the time needed for the code generation process (default).

Apply subsystem omission tags Considers **DsVdOmit** tags specified at subsystems for the variable description.

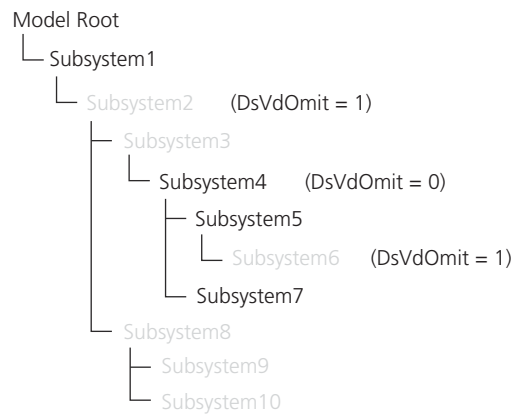
If selected, **DsVdOmit** tags of subsystems have the following effects on the generated variable description file:

Setting	Description
DsVdOmit tag is set to 1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. Use <code>set_param(gcb, 'Tag', 'DsVdOmit=1')</code> .
DsVdOmit tag is set to 0	The subsystem contents including all blocks beneath this subsystem do appear in the generated variable description file, even if a subsystem above this subsystem has set the DsVdOmit tag to 1. Use <code>set_param(gcb, 'Tag', 'DsVdOmit=0')</code> .
DsVdOmit tag is set to -1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. DsVdOmit settings of subsystems included in this subsystem are ignored. Use <code>set_param(gcb, 'Tag', 'DsVdOmit=-1')</code> .

If cleared, settings of the **DsVdOmit** tag are ignored for all subsystems in the model. All subsystems and their contents appear in the generated variable description file.

Exclusion is applied recursively through the model hierarchy. To reinclude subsystems, you can enter **DsVdOmit=0**. You can set the Omit flag by using workspace variables. For example, if **WSVar1** is a workspace variable, you can use `set_param(subsystemHandle, 'Tag', 'DsVdOmit=$(WSVar1)')` to let the Omit tag value being evaluated during build process.

If you set the **Apply subsystem omission tags** option, the variables of the grayed subsystems in the illustration below are not generated into the variable description file.

**Tip**

This option provides an alternative to the TRC Exclusion block from the Extras library to exclude a subsystem and its contents from the generated variable description file. It can be used together with the TRC Exclusion block in the same model.

Related files [Variable Description File \(TRC File\)](#) on page 123.

Model Parameter Configuration Dialog

Access	The Configure command to open this dialog is available on the Optimization page of the Code Generation dialog, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) on page 25.
Purpose	To declare specific parameters as tunable parameters.
Description	You can make individual parameters tunable if the Default parameter behavior option is set to Inlined .
Dialog settings	This section gives only a brief summary of the dialog settings. For a detailed description of the Tunable Parameters dialog and its options, refer to the Simulink documentation.

Global (tunable) parameters frame**New** Adds a new tunable parameter.**Remove** Removes a tunable parameter.**Name** Enter the parameter name here.**Storage class** Select the storage class for the parameter.**Storage type qualifier** Enter the storage type qualifier here.**Source List frame**

From the drop-down list, select whether you want to have all *MATLAB workspace* variables or only the *Referenced workspace variables* displayed in the source list.

The source list displays all the workspace variables that correspond to the selection of the drop-down list. The variables that are made global (tunable) parameters are displayed in bold italics.

Refresh list Updates the list from the workspace/model.**Add to table >>** Adds the highlighted variable(s) to the Global (tunable) parameters list.**Related topics****References**

[Code Generation Dialog \(Model Configuration Parameters Dialogs\)..... 25](#)

RTI Task Configuration Dialog

Access

The **Task Configuration** command to open this dialog is available on the RTI simulation options page of the [Code Generation dialog](#), refer to [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25.

Note

If you opened the RTI Task Configuration dialog via the Multiprocessor Setup Dialog, refer to [RTI Task Configuration Dialog \(Multiprocessor Setup Dialog\)](#) on page 59.

When working with configuration references, the **Task Configuration** button of a configuration set in the MATLAB base workspace that is referenced by at least two open models opens a **Model Selection** dialog. The dialog lists all open models that have an activated configuration reference to the current configuration set in the MATLAB base workspace. Click **Select** to select the model you want to open the RTI Task Configuration dialog for.

With the Task Configuration API, you can automate the task configuration in your model. For more information, refer to [RTI and RTI-MP Task Configuration API Reference](#) on page 201.

Purpose	To assign priorities to the tasks of a model and configure their overrun strategy.
Description	You can assign a priority to each task of your model and configure its overrun strategy. Both are relevant only to real-time simulation.
Using model referencing	When you use model referencing, only the task configuration performed for the top-level model is taken into consideration. Task configurations for referenced models are ignored during the task configuration and the build process of the top-level model.
Dialog settings	<p>Tasks with configurable priority Displays the tasks of your model.</p> <ul style="list-style-type: none"> ▪ Priorities are ordered top-down. ▪ Priority numbers can range from 1 (highest priority) to 128 (lowest priority). ▪ The background task is not shown because it always has the lowest priority. ▪ Interrupt-driven tasks are named after the corresponding interrupt blocks. ▪ Timer tasks are named after their sample times. <p>The number of timer tasks depends on the execution mode. For details, refer to Treat each discrete rate as a separate task on page 17.</p> <p>Up Increases the priority of the highlighted task.</p> <p>As Previous Assigns the priority of the previous task to the currently selected task.</p> <p>Down Decreases the priority of the highlighted task.</p> <p>Task name Displays the name of the currently selected task.</p> <p>Interrupt source Displays the name of the interrupt source of the currently selected task. This can be either a hardware or a software interrupt or a timer interrupt.</p> <p>Interrupt block Displays the name and path of the hardware or software interrupt block if an interrupt block-driven task is selected.</p> <p>Priority Displays the priority of the currently selected task.</p> <p>Stop simulation The simulation is stopped whenever a task overrun occurs.</p> <p>Queue task before simulation stop When a task overrun occurs, the task instance is queued for later execution.</p> <p>Max. queued task instances Maximum number of queued task instances. If this limit is reached, the simulation is stopped.</p>

Ignore overrun When a task overrun occurs, the task concerned is not scheduled. As a result, task instances are discarded in overrun situations.

Note

If one of the overrun strategies Queue task before simulation stop or Ignore overrun is selected, a task overrun might lead to unexpected simulation results since tasks are not executed in real-time.

Related topics

HowTos

[How to Change the Task Priorities \(RTI and RTI-MP Implementation Guide !\[\]\(f58128c41dc307543fa2591fa073e87a_img.jpg\)\)](#)
[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(f5126919fb264baa65afc980ba29ad65_img.jpg\)\)](#)

References

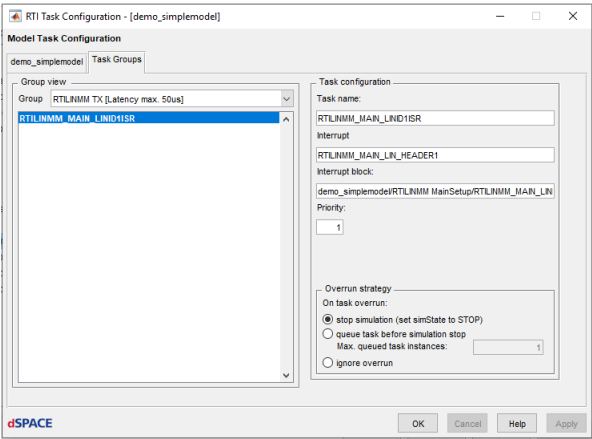
priority.....	96
RTI Task Configuration Dialog - Task Groups.....	41
RTI Task Configuration Dialog (Multiprocessor Setup Dialog).....	59

RTI Task Configuration Dialog - Task Groups

Introduction

If your model contains blocks of the RTI FlexRay Configuration Blockset or of the RTI LIN MultiMessage Blockset, the RTI Task Configuration dialog is extended by an additional Task Groups page.

The following illustration shows the page if your model contains blocks of the RTI LIN MultiMessage Blockset.



The second page of the RTI Task Configuration dialog shows the task groups:

Group view Lists all tasks in the selected task group.

Group Lets you choose a task group whose elements are listed below the drop-down list. The task group contains all the time-triggered tasks, which are processed on a last-come first-served basis. The sequence of tasks within a task group is determined by its start times and cannot be changed.

Related topics

References

[RTI Task Configuration Dialog](#)..... 39

Configure Initial Step Size Dialog

Access

The **Configure Initial Step Size** command to open this dialog is available on the RTI simulation options page of the Code Generation dialog, refer to [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25.

Note

To configure the initial step size for an RTI-MP model, refer to [Advanced Page \(Multiprocessor Setup Dialog\)](#) on page 57.

Purpose

To configure the increase of the initial step size.

Description

To avoid task overruns in the first simulation steps, you can specify the coefficients for the increase of the initial step size and the number of initial simulation steps to be increased.

Note

The timing of I/O signals does not match the timing of the real-time simulation while the specified increase of the first simulation steps is applied.

If you use multiple configuration sets, you must select the configuration set to be used for configuring the initial simulation steps before the dialog is opened.

Up to dSPACE Release 2019-B, you configured the behavior of the initial simulation steps with the **-DFIRST_SIMSTEP_INCREASEMENT** and **-DNUM_INCREASED_SIMSTEPS** compiler options. As of dSPACE Release 2020-A, these compiler options are discontinued. If you open or load an RTI model

created with Release 2019-B or earlier, the compiler options are automatically migrated to the new coefficients. Before the migration starts, the values of the compiler options are stored in
`<WorkingDirectory>\<ModelName>_migration_info.txt`.

Dialog settings

The dialog provides the following settings.

Coefficient of initial simsteps increasement Lets you enter an integer value ≥ 0 to specify the increase of the simulation steps. The resulting step size is displayed in the Increased step size setting, refer to [Increased step size](#) on page 43.

Coefficient of number of increased simsteps Lets you enter an integer value ≥ 1 to specify the number of the simulation steps the increased step size are to be applied to.

Real-time step size Displays the step size that you specified for the real-time simulation. The real-time step size is the step size on which the increase is based.

Increased step size Displays the step size resulting in the increase.

$$StepSize_{Increased} = (1 + Coeff_{Increasement}) \cdot StepSize_{Real-time}$$

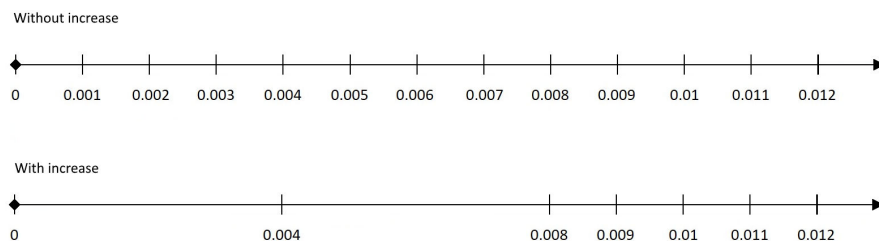
Regular step size after Displays the time after which the real-time step size is applied again.

$$StepSize_{After} = Coeff_{NumberOfSteps} \cdot StepSize_{Increased}$$

Example

The figure shows an example using the following settings:

- Real-time step size: 0.001 s
- Coefficient of initial simsteps increasement: 3
This results in an increased step size of 0.004 s $(= (1+3) \cdot 0.001 \text{ s})$.
- Coefficient of number of increased simsteps: 2
This results in a Regular step size after value of 0.008 s $(= 2 \cdot 0.004 \text{ s})$.



Signal Properties Dialog

Access

Menu bar	None
Context menu of	Signal
Shortcut key	None
Icon	None

Purpose

To specify the properties of the selected signal.

Description

You can make a signal displayable even if certain code optimization options are selected.

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the **Signal Properties** dialog and its options, refer to the Simulink documentation.

The Simulink Signal Properties dialog contains the following pages:

- [Logging and accessibility page](#) on page 44
- [Code Generation page](#) on page 44
- *Documentation* page (not relevant to RTI)

Signal name Specify the name of the signal if desired.

Logging and accessibility page

Test Point Select this checkbox if you want to make the signal global.

Code Generation page

Storage class Select the storage class for the signal if desired.

RTI-MP Dialog Reference

Introduction

The various dialogs of Simulink and RTI-MP provide many options to customize the behavior of Simulink and code generation for the dSPACE real-time platforms.

Where to go from here

Information in this section

Interprocessor communication

[Communication Channels Display](#).....47

To display all communication connections and their parameters.

[Change Connection Dialog](#).....47

To change the CPU names and channel number of the specified IPC block.

Model configuration, multiprocessor setup and simulation parameters

[Multiprocessor Setup Dialog](#).....48

To customize and start automatic program building for multiprocessor systems.

[Main Page \(Multiprocessor Setup Dialog\)](#).....49

To configure the global options for the real-time code generation of a Simulink model for a multiprocessor system, display the main settings of the application for each CPU, and start the build and download process.

[CPU's Page \(Multiprocessor Setup Dialog\)](#).....53

To configure the real-time code generation options of the specified CPU in a multiprocessor Simulink model and define the application names, the solver, the real-time step size and the build options for this CPU. You will find a separate CPU's page for any CPU in your multiprocessor system.

Advanced Page (Multiprocessor Setup Dialog).....	57
To define global RTI-MP settings, such as the multiprocessor target and the network client.	
Documentation Page (Multiprocessor Setup Dialog).....	59
To show additional information about the current model.	
RTI Task Configuration Dialog (Multiprocessor Setup Dialog).....	59
To assign priorities to the tasks of a model and configure their overrun strategy.	
RTI Task Configuration Dialog - Task Groups (Multiprocessor Setup Dialog).....	61
If your model contains blocks of the RTI FlexRay Configuration Blockset or of the RTI LIN MultiMessage Blockset, the RTI Task Configuration dialog is extended.	
Configure Initial Step Size Dialog (Multiprocessor Setup Dialog).....	62
To configure the increasement of the initial step size.	
CPU Options Dialog.....	64
To specify simulation parameters for multiprocessor models.	
Variable Description File Options Page (CPU Options Dialog).....	67
To set options that control the generation of the variable description file.	
Rename CPU Dialog.....	70
To change the name of the CPU.	
Multiprocessor Topology Setup Dialog.....	71
To specify the topology of a multiprocessor system.	
Model Configuration Parameters Dialogs.....	73
To specify configuration parameters for multiprocessor models.	
Code Generation Dialog (Model Configuration Parameters Dialogs).....	80
To set Simulink Coder and RTI-MP code generation options.	
Model Parameter Configuration Dialog.....	85
To declare specific parameters as tunable parameters.	
 Build properties	
Build Options Page (CPU Options Dialog).....	64
To set compiler and other build options.	
 Diagnostic properties	
Diagnostics Dialog (Model Configuration Parameters Dialogs).....	75
To specify options that make specific model conditions generate either error messages or warnings.	

Signal properties	
Signal Properties Dialog	86
To specify the properties of the selected signal.	

Communication Channels Display

Access	Menu bar	None
	Context menu of	None
	Shortcut key	None
	Icon	None
	Others	Multiprocessor Setup block
		<ul style="list-style-type: none"> Multiprocessor Setup dialog - Advanced page - Show all IPC channels of the Model

Purpose	To display all communication connections and their parameters.
Description	All communication connections and their parameters can be changed by double-clicking the corresponding IPC blocks in the Simulink model.

Related topics	References
	Advanced Page (Multiprocessor Setup Dialog) 57

Change Connection Dialog

Access	Menu bar	None
	Context menu of	None
	Shortcut key	None
	Icon	None
	Others	IPCx block
		<ul style="list-style-type: none"> Communication Channel Setup dialog - IPC Channel Setup page - Change Connection

Purpose To change the CPU names and channel number of the specified IPC block.

Dialog settings

Connection: Source CPU Lets you specify the name of the source CPU (origin of the IPCx block). CPU names are restricted to eight alphanumeric characters.

To add another CPU to the multiprocessor model, simply enter a new CPU name. The parameters of the new CPU, such as application name and solver, can be entered the next time the **Multiprocessor Setup** dialog is opened.

Connection: Destination CPU Lets you specify the name of the target CPU. CPU names are restricted to eight alphanumeric characters.

To add another CPU to the multiprocessor model, simply enter a new CPU name. The parameters of the new CPU, such as application name and solver, can be entered the next time the **Multiprocessor Setup** dialog is opened.

Connection: Channel Lets you specify the channel number. You can select any non-negative integer number as the channel number.

Related topics

References

IPCx..... 261

Multiprocessor Setup Dialog

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Double-click the Multiprocessor Setup block

Description

Automatic program building for multiprocessor systems can be customized and started from the **Multiprocessor Setup** dialog. The settings on this dialog are necessary to convert a Simulink model into a real-time application.

The **Multiprocessor Setup** block must be located on the root level of an RTI-MP model.

Dialog pages

This dialog contains the following pages:

- [Main Page \(Multiprocessor Setup Dialog\)](#) on page 49 for setting the global parameters of the multiprocessor Simulink model.
- [CPU's Page \(Multiprocessor Setup Dialog\)](#) on page 53 for setting the CPU-specific parameters of the multiprocessor Simulink model.
- [Advanced Page \(Multiprocessor Setup Dialog\)](#) on page 57 for changing the target multiprocessor system and setting advanced global options.
- [Documentation Page \(Multiprocessor Setup Dialog\)](#) on page 59 for getting further information about your model and adding comments.

Related topics**HowTos**

[How to Specify Options for the Build Process \(RTI and RTI-MP Implementation Guide !\[\]\(e3f8612927870f2e0f9f5989e6dd3064_img.jpg\)\)](#)

References

[Model Configuration Parameters Dialogs.....](#) 73
[Multiprocessor Setup.....](#) 260

Main Page (Multiprocessor Setup Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog

Purpose

To configure the global options for the real-time code generation of a Simulink model for a multiprocessor system, display the main settings of the application for each CPU, and start the build and download process.

Available option frames

You find the following option frames:

- Simulation options
Refer to [Simulation options frame](#) on page 50.
- Global build options
Refer to [Global build options frame](#) on page 51.

- Global variable description file options
Refer to [Global variable description file options frame](#) on page 51.
- Application setup
Refer to [Application setup frame](#) on page 52.

Simulation options frame

Initial simulation state Lets you define the initial simulation state for a model.

- If set to *RUN*, the simulation is started right after the download (default).
- If set to *STOP*, the simulation remains stopped after the download. The output devices are set to the Termination values.

The current simulation state of the real-time simulation is available in the variable description file (see [simState](#) on page 97).

Stop time Lets you specify a stop time for the simulation.

For real-time simulations you should specify *inf* (meaning infinity) as the stop time.

Scheduler mode Lets you specify the execution mode for calculating your model.

- If *single timer task* is selected, the application of each CPU contains just one timer task. It is suitable for most applications (default).
- If *multiple timer task* is selected, the application of each CPU contains the same number of timer tasks as it does sample times.

For details on both execution modes, refer to [Single Timer Task Mode \(RTI and RTI-MP Implementation Guide !\[\]\(c50c8b7b2cc2cf9ff925edec0ee94c0d_img.jpg\)](#)), [Multiple Timer Task Mode \(RTI and RTI-MP Implementation Guide !\[\]\(8bed43dc33ecdde61e2f76c8f5517125_img.jpg\)](#)) and [Comparing the Execution Modes \(RTI and RTI-MP Implementation Guide !\[\]\(047f882704cdc566325d0a83645d692e_img.jpg\)](#)).

Basic step size Lets you specify the basic step size for a multiprocessor system. You must specify the basic step size as the greatest common divisor for the step sizes of all CPUs.

The step size of a particular CPU equals the product of the basic step size and the step size multiple of that CPU.

The synchronous timing of multiprocessor systems is possible only if the step sizes of all the CPUs have a common divisor: the basic step size. You can define an individual step size for each CPU (see [Step size](#) on page 52). The product of the basic step size and a CPU's step size multiple defines the actual step size for the CPU, which is also available in the corresponding variable description files (see [modelStepSize](#) on page 92). The unit of the basic step size is seconds.

Execution mode Lets you specify whether the model must be calculated in real-time or non real-time execution mode.

- If *real-time* is selected, the model is calculated in real time (default).
- If *time-scaled* is selected, the simulation is performed faster or more slowly according to the Time scale factor specified (see below). This means that calculating one second in the simulation takes less or more than one real second.

RTI-MP does not provide an *as fast as possible* setting because multiprocessor applications require well-defined synchronization among all CPUs.

To execute your model in the *time-scaled* execution modes, you have to build it in the single timer task mode. You can still use hardware and software interrupts.

Time scale factor Lets you specify a time scale factor for the *time-scaled* execution mode. (Enabled only if the *time-scaled* execution mode is selected, see above.)

It must be a decimal value greater than 0.0.

For example, if you specify the *time-scaled* execution mode and time scale factor **0.2**, the simulation runs 5 times faster than in the *real-time* execution mode. A time scale factor of **1** corresponds to a real-time simulation. This is the default value of the time scale factor.

Global build options frame

Enable real-time testing Lets you enable or disable real-time testing for the generated real-time application.

To improve the startup time of the real-time application, the build process generates a binary TRZ file containing the MAP file and the TRC file.

Note

- This option is only available for RTI1006, RTI1007, and RTI1202.
- If you want to use the Signal Generator or the Bus Navigator in ControlDesk, you have to set the Enable real-time testing option in your Simulink model. For new models this option is set by default. If the Enable real-time testing option is set, the size of the built real-time application increases. The time for loading the application from the flash memory therefore will be increased. In worst case, the rebuilt real-time application cannot be loaded because the board's memory is not sufficient.
- If you do not need the real-time testing support, clear the Enable real-time testing option in your model. For RTI1007 and RTI1202, this option is always set and cannot be modified.

Global variable description file options frame

Include initial parameter values To make the initial parameter values available in the variable description file.

- If selected, the initial parameter values of the model are generated into the variable description file. This lets you carry out offline calibration for your model with ControlDesk.

This does not apply to:

- Stateflow parameters of the State Machine Data group
- If cleared, the initial parameter values of the model are not generated into the variable description file. For large models especially, this can significantly reduce the time needed for the code generation process (default).

Application setup frame

CPU Displays the names of the CPUs that are defined in the model. The master CPU is marked with an (M).

Note

RTI-MP automatically selects the CPU with the name **master** as the master CPU. If no CPU is named **master**, RTI-MP selects the CPU with the name that is the first in the alphabet.

Application Displays the names of the applications assigned to the CPUs of the model. You can change the application name of a CPU via the Application name setting on the [CPU's Page \(Multiprocessor Setup Dialog\)](#) on page 53.

Board type Displays the board types for the CPUs of the model. To change the current settings, see [Board type](#) on page 53.

Solver Displays the selected solvers for the CPUs of the model. To change the current settings, see [Solver](#) on page 52.

Step size Displays the real-time step sizes (*step size multiple* × *basic step size*) for the CPUs of the model. To change the current settings, see [Step size](#) on page 54 and [Basic step size](#) on page 50.

Build/Build OK Lets you start the build procedure for the selected CPU.

- If the label is *Build*, the application for the selected CPU still needs to be built.
- If the label is *Build OK*, the application for the selected CPU matches the RTI-MP configuration.

For an overview of the build process, refer to [Introducing the Build and Download Process \(RTI and RTI-MP Implementation Guide\)](#).

Build All Lets you start the build procedure for all CPUs. This is the same as building the applications for all the CPUs successively.

Download Lets you generate the system description file (SDF file) for the model and download the application to the multiprocessor system.

If you provide a user system description file with user-specific statements, RTI-MP automatically merges it into the system description file. For information on the SDF file syntax, refer to [SDF File Syntax](#) on page 133.

For an overview of the build process, refer to [Introducing the Build and Download Process \(RTI and RTI-MP Implementation Guide\)](#).

Related topics

Basics

- [Comparing the Execution Modes \(RTI and RTI-MP Implementation Guide !\[\]\(86b7331e04fe40a56bcff2e9c065738b_img.jpg\)\)](#)
- [Introducing the Build and Download Process \(RTI and RTI-MP Implementation Guide !\[\]\(92f87f30b7499b35d0173f4346c498d6_img.jpg\)\)](#)
- [Multiple Timer Task Mode \(RTI and RTI-MP Implementation Guide !\[\]\(497b6684f704c0aa6fbea9f0fd4d56c7_img.jpg\)\)](#)
- [Simulation Control \(RUN/STOP Mechanism\) \(RTI and RTI-MP Implementation Guide !\[\]\(4320279ad715106747262028f44bd102_img.jpg\)\)](#)
- [Single Timer Task Mode \(RTI and RTI-MP Implementation Guide !\[\]\(25e9c4c673069177325c65bf4771169e_img.jpg\)\)](#)

References

Multiprocessor Setup.....	260
---	---------------------

CPU's Page (Multiprocessor Setup Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none">▪ Multiprocessor Setup dialog

Purpose

To configure the real-time code generation options of the specified CPU in a multiprocessor Simulink model and define the application names, the solver, the real-time step size and the build options for this CPU. You will find a separate CPU's page for any CPU in your multiprocessor system.

Basic options frame

Application name Lets you specify the name of the application (= *submodel*) running on the selected CPU.

Note

The name specified in this field is used for the object file and other files that are generated by RTI-MP and Simulink Coder. It must therefore be a valid file name, which means that only alphanumeric characters are allowed.

Board type Lets you select the board type for all CPUs of the model. If you change the board type for one CPU, all the other CPUs are updated to the same board type automatically. The following board types are available for multiprocessor systems:

Board Type	Processor
DS1006 ¹⁾	AMD Opteron™ Processor Model 2xx
DS1007 ¹⁾	Freescall P5020
MicroLabBox (DS1202) ²⁾	Freescall P5020

¹⁾ DS1006 Processor Boards and DS1007 PPC Processor Boards cannot be mixed within a multiprocessor system using RTI-MP.

²⁾ RTI-MP only supports the multicore features of MicroLabBox.

RTI-MP automatically chooses the matching *system target file* controlling the generation of the application code, and the template makefile controlling the compilation and linkage process for all source modules of an application (see [rti<xxx>.tlc](#) on page 105).

Solver Lets you specify the solver type for calculating the model.

The higher the order of a solver, the higher its numerical precision and the more execution time it needs. With RTI-MP you can use the following solvers:

Solver	Description
discrete	No integration, suitable only for models without continuous states
ode1	Euler's method, suitable for most applications
ode2	Heun's method, also known as the improved Euler formula
ode3	The Bogacki-Shampine formula
ode4	The fourth-order Runge-Kutta formula
ode5	The Dormand-Prince (RK5) formula
ode8	The Dormand-Prince (RK8(7)) formula
ode14x	Combination of Newton's method and extrapolation from the current value
ode1be ¹⁾	Backward Euler method with additional setting for the number of Newton's iterations.

¹⁾ Only available as of MATLAB R2020a.

Extrapolation order Lets you specify the extrapolation order used by the ode14x solver to compute a model's states at the next time step from the states at the current time step.

Number Newton's iterations Lets you specify the number of Newton's method iterations used by the ode14x solver to compute a model's states at the next time step from the states at the current time step.

Step size Lets you specify the step size multiples for the step sizes of the individual CPUs.

The actual step size of the selected CPU is the product of the step size multiple and the Basic step size (see page [Basic step size](#) on page 50).

The step size multiple of a CPU must be an integer multiple of the step size multiples of all other CPUs with a smaller step size multiple. For example, if your

system contains two CPUs and the step size multiple of CPU1 is 2, you can specify 4 as the step size multiple for CPU2.

Advanced options frame

Profile synchronized swinging buffer communication Lets you profile all the interprocessor communication channels that are operated with the synchronized swinging buffer protocol.

- If selected, the **rtimpdiag** diagnostic tool provides additional profiling information for all the interprocessor communication input channels of the current CPU that are operated with the synchronized swinging buffer protocol. This includes the number of input calls, the current waiting time, the maximum waiting time, and whether an overrun is caused while waiting. Profiling the interprocessor communication causes a negligibly small time overhead for each IPC input channel that is operated with the synchronized swinging buffer protocol.
- If cleared, the time overhead is eliminated. As a result, **rtimpdiag** does not provide the profiling information (default).

Treat as optional CPU Lets you make the selected CPU optional.

- If selected, the resulting real-time application can be executed even if the current CPU is unavailable, for example, switched off or removed from the multiprocessor system.
- If cleared, the current CPU must be available when you want to run the resulting real-time application (default).

Note

This option is not available for DS1007 and MicroLabBox (DS1202).

Making certain slave CPUs optional lets you reuse the same real-time application for different hardware configurations. As a result, you do not need to rebuild your model if you do not have all the slave CPUs available.

There are some points to note when using this technique. For details, see [Working with Subsets of a Multiprocessor Topology \(DS1006 Features !\[\]\(d0262bbe9d2356661a2e89321dfcc781_img.jpg\)](#)).

Use Simulink Coder Custom Code settings Lets you specify to have the Custom Code page settings of the main model transferred to the relevant submodel.

Use Model Workspace settings Lets you specify to transfer the Model Workspace settings of the main model to the CPU's submodel.

Always create a new submodel Lets you specify whether to create a new submodel for the current CPU by model separation of the entire MP model. If this option is cleared, the existing submodel for the current CPU will not be overwritten when you use **Show submodel**, **Task Configuration** or **Build**. If you start a build process, the following settings from the MP main model are transferred to a submodel:

- Model workspace
- MP-relevant options of the Configuration Parameter Set
- Configuration of the IPC and IPI blocks

The submodel can be explicitly updated by using **Update submodel**.

Gigalink background CRC Lets you specify the behavior of Gigalink cyclic redundancy check in the background. You can choose between the following settings:

- Off
- Warn at checksum mismatch
- Error at checksum mismatch

Note

To avoid a timeout during initialization, you can use the following compiler option:

```
-DRTIMP_GL_INIT_NUM_RETRIES=n
```

Submodel configuration frame

Rename CPU Lets you rename a CPU in the RTI-MP model.

This opens the **Rename CPU** dialog, which lets you enter a name for the selected CPU. Refer to [Rename CPU Dialog](#) on page 70.

Task Configuration Lets you assign priorities to the tasks of a submodel and specify their overrun behavior.

This opens the **RTI Task Configuration** dialog, which lets you configure the tasks of a submodel. Refer to [RTI Task Configuration Dialog \(Multiprocessor Setup Dialog\)](#) on page 59.

Show submodel Lets you extract all the blocks that are executed on the specified CPU and place them in a temporary Simulink model.

Update submodel Lets you explicitly update a submodel, if the **Always create a new submodel** option is disabled.

If the update detects new interface blocks in the entire MP model, this function creates a subsystem with the **Contains blocks created during 'Update submodel'** label. It contains the missing IPC Send, IPC Receive, IPI Send and IPI Receive blocks with the related configuration. You can copy them to your submodel to update the interfaces for communication and interrupt handling. The build process ignores the subsystem with the generated IPC and IPI blocks.

Build process frame

Build Options Lets you configure compiler and other build options for the selected CPU.

This opens the **CPU Options** dialog on the **Build Options** page, which lets you specify the settings. Refer to [Build Options Page \(CPU Options Dialog\)](#) on page 64.

Variable Description File Options Lets you configure the variable description file options for the selected CPU.

This opens the CPU Options dialog on the Variable Description File Options page, which lets you specify the settings. Refer to [Variable Description File Options Page \(CPU Options Dialog\)](#) on page 67.

Build/Build OK Lets you start the build procedure for the selected CPU.

- If the label is *Build*, the application for the selected CPU still needs to be built.
- If the label is *Build OK*, the application for the selected CPU matches the RTI-MP configuration.

For an overview of the build process, refer to [Introducing the Build and Download Process \(RTI and RTI-MP Implementation Guide !\[\]\(c3d993ca47bfe2a953c700506ce31fa0_img.jpg\)\)](#).

Related topics

Basics

[Handling Exceptions \(RTI and RTI-MP Implementation Guide !\[\]\(003082e50e3009141f59bd5df831749f_img.jpg\)\)](#)

References

[Files Controlling the Build and Download Process..... 105](#)

Advanced Page (Multiprocessor Setup Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog

Purpose

To define global RTI-MP settings, such as the multiprocessor target and the network client.

Communication and topology configuration frame

Multiprocessor Network Topology Lets you specify the topology of a multiprocessor system.

This opens the Multiprocessor Topology Setup dialog, which lets you configure the network topology. Refer to [Multiprocessor Topology Setup Dialog](#) on page 71.

Show all IPC Channels of the Model Lets you view all communication connections and their parameters.

This opens the Communication Channels display. Refer to [Communication Channels Display](#) on page 47.

Check Multiprocessor Model Lets you analyze the multiprocessor model and get a report of possible problems.

Load options frame

Load to Flash memory Lets you download a real-time application to the Flash memory.

- If selected, the real-time application is downloaded to the Flash memory (nonvolatile memory).
- If cleared, the real-time application is downloaded to the RAM (default).

Note

Downloading a multiprocessor application to the flash memories of the processor boards can take rather a long time, which usually results in a timeout. You can prevent this via the `-DRTIMP_GL_NO_TIMEOUT` option (see [Compiler options](#) on page 65).

Platform name Lets you enter the name of the platform used for downloading the real-time application.

You must enter the same name that you specified in ControlDesk's Platform Manager when you registered the multiprocessor platform.

RTI-MP selects the following default platform name:

- ds1007
- ds1202 (for MicroLabBox)
- Multiprocessor

Initial step size increasement frame

Configure Initial Step Size Lets you configure the increase of the initial step size per CPU.

This button opens the **Configure Initial Step Size** dialog, which lets you configure the coefficients for the increase of the initial step size and for the number of steps to be increased. Refer to [Configure Initial Step Size Dialog \(Multiprocessor Setup Dialog\)](#) on page 62.

Related topics

References

[Build Options Page \(CPU Options Dialog\)](#)..... 64

Documentation Page (Multiprocessor Setup Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog

Purpose

To show additional information about the current model.

Dialog settings

- Comments** Lets you specify comments for the current RTI-MP model.
- Hyperlink to URL** Lets you specify a URL for the current RTI-MP model.
- Open URL** Lets you open the URL specified as Hyperlink to URL.
- Print Report** Lets you generate and open a report for the current RTI-MP model containing all the parameters of the RTI-MP dialogs.

RTI Task Configuration Dialog (Multiprocessor Setup Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog - CPU's page - Task Configuration

With the Task Configuration API, you can automate the task configuration in your model. For more information, refer to [RTI and RTI-MP Task Configuration API Reference](#) on page 201.

Purpose

To assign priorities to the tasks of a model and configure their overrun strategy.

Description

You can assign a priority to each task of your model and configure its overrun strategy. Both are relevant only to real-time simulation.

Using model referencing

When you use model referencing, only the task configuration performed for the top-level model is taken into consideration. Task configurations for referenced models are ignored during the task configuration and the build process of the top-level model.

Dialog settings

Tasks with configurable priority Displays the tasks of your model.

- Priorities are ordered top-down.
- Priority numbers can range from 1 (highest priority) to 128 (lowest priority).
- The background task is not shown because it always has the lowest priority.
- Interrupt-driven tasks are named after the corresponding interrupt blocks.
- Timer tasks are named after their sample times.

The number of timer tasks depends on the execution mode. For details, refer to [Scheduler mode](#) on page 50.

Up Increases the priority of the highlighted task.

As Previous Assigns the priority of the previous task to the currently selected task.

Down Decreases the priority of the highlighted task.

Task name Displays the name of the currently selected task.

Interrupt source Displays the name of the interrupt source of the currently selected task. This can be either a hardware or a software interrupt or a timer interrupt.

Interrupt block Displays the name and path of the hardware or software interrupt block if an interrupt block-driven task is selected.

Priority Displays the priority of the currently selected task.

Stop simulation The simulation is stopped whenever a task overrun occurs.

Queue task before simulation stop When a task overrun occurs, the task instance is queued for later execution.

Max. queued task instances Maximum number of queued task instances. If this limit is reached, the simulation is stopped.

Ignore overrun When a task overrun occurs, the task concerned is not scheduled. As a result, task instances are discarded in overrun situations.

Note

If one of the overrun strategies **Queue task before simulation stop** or **Ignore overrun** is selected, a task overrun might lead to unexpected simulation results since tasks are not executed in real-time.

Related topics

HowTos

[How to Change the Task Priorities in RTI-MP Models \(RTI and RTI-MP Implementation Guide !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)\)](#)
[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(d4257ae6a3e163e6d467b3eb87960fa1_img.jpg\)\)](#)

References

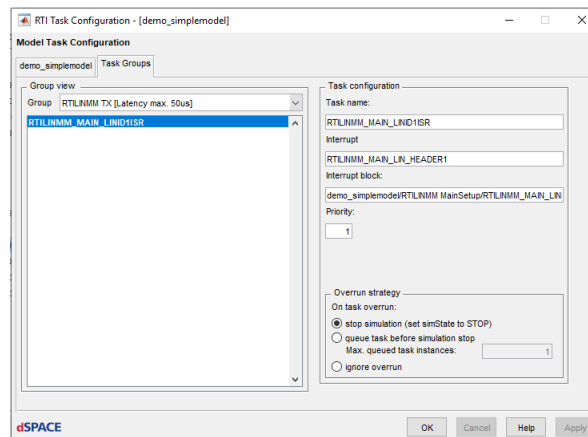
[CPU's Page \(Multiprocessor Setup Dialog\)..... 53](#)
[priority..... 96](#)

RTI Task Configuration Dialog - Task Groups (Multiprocessor Setup Dialog)

Introduction

If your model contains blocks of the RTI FlexRay Configuration Blockset or of the RTI LIN MultiMessage Blockset, the RTI Task Configuration dialog is extended by an additional Task Groups page.

The following illustration shows the page if your model contains blocks of the RTI LIN MultiMessage Blockset.



The second page of the RTI Task Configuration dialog shows the task groups:

Group view Lists all tasks in the selected task group.

Group Lets you choose a task group whose elements are listed below the drop-down list. The task group contains all the time-triggered tasks, which are processed on a last-come first-served basis. The sequence of tasks within a task group is determined by its start times and cannot be changed.

Related topics

References

[RTI Task Configuration Dialog \(Multiprocessor Setup Dialog\)..... 59](#)

Configure Initial Step Size Dialog (Multiprocessor Setup Dialog)

Access

The **Configure Initial Step Size** command to open this dialog is available on the Advanced page of the Multiprocessor Setup dialog, refer to Multiprocessor Setup Dialog.

Purpose

To configure the increase of the initial step size.

Description

To avoid task overruns in the first simulation steps, you can specify the coefficients for the increase of the initial step size and the number of initial simulation steps to be increased. The resulting values are separately displayed for each CPU.

Note

The timing of I/O signals does not match the timing of the real-time simulation while the specified increase of the first simulation steps is applied.

Up to dSPACE Release 2019-B, you configured the behavior of the initial simulation steps with the `-DFIRST_SIMSTEP_INCREASEMENT` and `-DNUM_INCREASED_SIMSTEPS` compiler options. As of dSPACE Release 2020-A, these compiler options are discontinued. If you open or load an RTI-MP model created with Release 2019-B or earlier, and you call `rtimp_build2` or open the Multiprocessor Setup block, the compiler options are automatically migrated to the new coefficients. Before the migration starts, the values of the compiler options are stored in `<WorkingDirectory>\<ModelName>_migration_info.txt`.

Dialog settings

The dialog provides the following settings.

Coefficient of initial simsteps increase Lets you enter an integer value ≥ 0 to specify the increase of the simulation steps. The resulting step size is displayed in the Increased step size setting, refer to [Increased step size](#) on page 63.

Coefficient of number of increased simsteps Lets you enter an integer value ≥ 1 to specify the number of the simulation steps the increased step size are to be applied to.

CPU Name Displays the names of the CPUs in the RTI-MP model, such as **master** and **slave**.

Real-time step size Displays the step size that you specified for the real-time simulation for each CPU in the RTI-MP model. The real-time step size is the step size on which the increase is based.

Increased step size Displays the step size resulting in the increase for each CPU in the RTI-MP model.

$$StepSize_{Increased} = (1 + Coeff_{Increase}) \cdot StepSize_{Real-time}$$

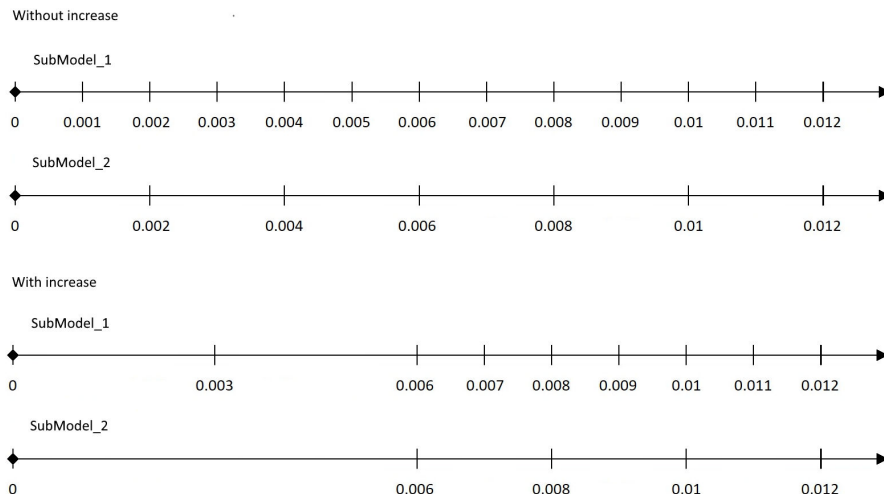
Regular step size after Displays the time after which the real-time step size is applied again for all CPUs in the RTI-MP model.

$$StepSize_{After} = Coeff_{NumberOfSteps} \cdot StepSize_{Increased}$$

Example

The figure shows an example using the following settings:

- Real-time step size:
 - SubModel_1: 0.001 s
 - SubModel_2: $2 \cdot 0.001 \text{ s} = 0.002 \text{ s}$
- Coefficient of initial simsteps increasement: 2
This results in an increased step size of:
 - SubModel_1: $0.003 \text{ s} (= (1+2) \cdot 0.001 \text{ s})$
 - SubModel_2: $0.006 \text{ s} (= (1+2) \cdot 0.002 \text{ s})$.
- Coefficient of number of increased simsteps: 1
This results in a Regular step size after value of $0.006 \text{ s} (= 1 \cdot 0.006 \text{ s})$.



CPU Options Dialog

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog - <CPU> page - Build Options or ▪ Multiprocessor Setup dialog - <CPU> page - Variable Description File Options

Dialog pages

This dialog contains the following pages:

- [Build Options Page \(CPU Options Dialog\)](#) on page 64 for setting compiler and other build options.
- [Variable Description File Options Page \(CPU Options Dialog\)](#) on page 67 for setting options that control the generation of the variable description file.

Build Options Page (CPU Options Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog - <CPU> page - Build Options

Purpose

To set compiler and other build options.

Dialog settings

Compiler options Lets you specify the compiler options. The following option is especially important:

- **-DRTIMP_GL_NO_TIMEOUT** If specified for a CPU, the timeout period for initializing the Gigalinks is set to infinity.

Downloading an application to the flash memory takes significantly longer than downloading it to the global memory. As a result, a timeout can occur when you download a multiprocessor application to the flash and one processor board attempts to synchronize its Gigalinks after the download while another is still busy with the download. The **-DRTIMP_GL_NO_TIMEOUT** option makes the multiprocessor system retry the initialization of the Gigalinks until it is successful.

Note

The initialization of the Gigalinks is not possible if the required connections are not available. If you did not specify the **-DRTIMP_GL_NO_TIMEOUT** option, a timeout results, which indicates that, for example, some Gigalinks are not connected correctly, or the expansion box is switched off. If you specified the **-DRTIMP_GL_NO_TIMEOUT** option, no timeout results. You should therefore use this option only if loading an application to the flash memory.

For more information, refer to the Load to Flash memory setting in [Advanced Page \(Multiprocessor Setup Dialog\)](#) on page 57.

The **NUM_INCREASED_SIMSTEPS** and **FIRST_SIMSTEP_INCREASEMENT** compiler options are discontinued with dSPACE Release 2020-A. The compiler options are automatically migrated. You can configure the initial step size increase in the Configure Initial Step Size Dialog.

Compiler optimizations Lets you specify the compiler optimization level.

- If *Default* is selected, the model is compiled with the platform- and compiler-specific default optimization, which is the highest recommended level of optimization.
- If *User defined* is selected, you can define the optimization via the User-defined optimizations edit field.
- If *None* is selected, the compilation is performed without any optimization.

User-defined optimizations Lets you specify user-defined compiler options. (Available only if compiler optimizations is set to *User-defined*.)

- If empty, the model code is compiled with the default optimization setting of the compiler.
- Do not specify optimization levels higher than the default.
Do not specify compiler options that have an effect on data widths and memory alignments.

For example, do not use the following options:

- Microtec PowerPC Compiler:
 - -KP
 - -Zm
 - -Zn
 - Do not use the **packed** prefix for struct definitions within your source code for variables that will be included in the variable description file.
- GNU GCC C Compiler:
 - -fpack-struct
 - -fshort-enums
 - Do not use the **__attribute__ ((__packed__))** directive for struct and enum definitions within your source code for variables that will be included in the variable description file.

For information about compiler options, refer to the following manuals:

Board	Processor Type	Compiler Manual
DS1104 MicroAutoBox II	PowerPC	<i>C/C++ Compiler User's Guide by Microtec</i>
DS1007 MicroLabBox	PowerPC	http://gcc.gnu.org/onlinedocs/gcc-4.8.3/gcc
DS1006	x86	http://gcc.gnu.org/onlinedocs/

Make options Lets you specify additional make options. There are only very few cases where additional make options are required.

Custom TLC options Lets you specify further code generation options.

Assertion mode RTI supports Simulink's Model Verification blocks for the real-time simulation.

To control the behavior of the Model Verification blocks in the real-time simulation, use RTI's **Assertion mode** setting. Depending on how you set this, one of the following actions is carried out if a verified signal leaves the desired range, meaning that the assertion occurs:

- OFF: No reaction (default).
- WARN: Issue a warning message in the Log Viewer of ControlDesk.
- STOP: Issue an error message and stop the simulation, i.e., set the simulation state to *STOP*.

For more information, refer to [How to Use Simulink's Model Verification Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(c50c8b7b2cc2cf9ff925edec0ee94c0d_img.jpg\)](#)).

Jitter and latency optimization Lets you select the task mode for the generated real-time application to optimize its latency for the different use cases.

Setting	Meaning
Standard (full functionality) tasks	The task execution is based on threads, which lets you include C++ and third-party code. The standard task mode is recommended for a task rate not faster than 1 ms.

Setting	Meaning
Low jitter, low latency tasks	<p>This is the default setting, which correlates to the task execution behavior of real-time applications generated with earlier dSPACE Releases.</p> <p>The task execution is based on interrupts, which lets you achieve a low trigger-to-task latency.</p> <p>The low jitter task mode is especially recommended for a task rate faster than 0.05 ms.</p> <p>Note: The support of C++ and third-party code is limited in low jitter task mode.</p>

Note

- This option is only available for RTI1007 and RTI1202.
- When you use a DS1007 Processor Board or a MicroLabBox, you can configure to use the specified task mode when building the real-time application or to overwrite the specified task mode with the configuration in the board's web interface.

For more information, refer to:

- [Basics on the Web Interface \(DS1007 Hardware Installation and Configuration Guide !\[\]\(511a36c244659513b679df9c639945de_img.jpg\)](#))
- [Basics on the Web Interface \(MicroLabBox Hardware Installation and Configuration !\[\]\(2c0783baf87a2728b2fe49eb1c34c456_img.jpg\)](#))

Related topics**References**

[Variable Description File Options Page \(CPU Options Dialog\)](#)..... 67

Variable Description File Options Page (CPU Options Dialog)

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	<p>Multiprocessor Setup block</p> <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog - <CPU> page - Variable Description File Options

Purpose

To set options that control the generation of the variable description file.

Variable description file options

Include only Simulink.Parameter and Simulink.Signal objects with global storage class

Lets you significantly reduce the time needed for code generation as it disables all the other variable description file options. It includes only parameters and signals in the variable description file which reference a Simulink.Parameter or a Simulink.Signal object in the MATLAB workspace. If you set this option, the variable description file only provides the Labels group, the Tunable Parameters group and the BusSystems group of the RTI CAN MultiMessage Blockset and the RTI LIN MultiMessage Blockset. It does not contain the entire model hierarchy. For the Simulink.Parameter and Simulink.Signal objects the following must apply:

- They must specify a global storage class (ExportedGlobal, ImportedExtern, ImportedExternPointer).
- For Simulink.Signal objects:
 - Either the Signal must resolve to Simulink.Signal object option is selected in the signal's Signal Properties dialog.
 - Or the Signal resolution option is set to Explicit and implicit on the Data Validity page of the Diagnostics dialog.

In the variable description file, Description, DocUnits, Min and Max entries of Simulink.Signal objects and Simulink.Parameter objects are available.

Include signal labels Lets you make the signal labels of the model available in the variable description file.

- If selected, the signal labels of the model are generated into the variable description file.
- If cleared, the signal labels of the model are not generated into the variable description file. For large models especially, this might significantly reduce the time needed for the code generation process.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include virtual blocks Lets you make the block outputs of virtual blocks (Demux, From, etc.) available in the variable description file.

- If selected, the virtual blocks are available in the variable description file.
- If cleared, the virtual blocks are not available in the variable description file. This might significantly reduce the time needed for the code generation process.

If you just want to have the outputs of a few virtual blocks available in the variable description file, you can add test points, or Signal Conversion blocks with the Output property set to *Signal copy*. These outputs are available in the variable description file, even if you turn off the generation of virtual blocks.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include states Lets you make the states of blocks available in the variable description file.

- If selected, the block states are generated into the variable description file.
- If cleared, the block states are not generated into the variable description file (default).

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Include derivatives Lets you make the derivatives of blocks available in the variable description file.

- If selected, the derivatives of blocks are generated into the variable description file.
- If cleared, the derivatives of blocks are not generated into the variable description file (default).

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Apply subsystem omission tags Considers **DsVdOmit** tags specified at subsystems for the variable description.

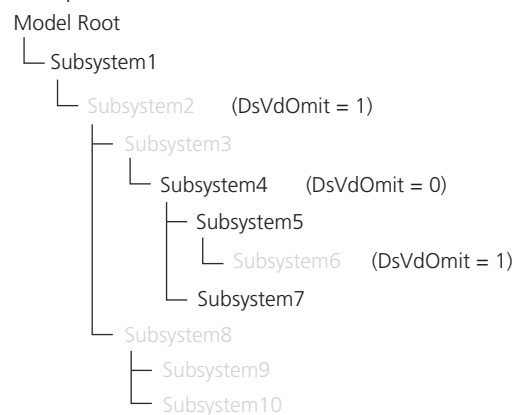
If selected, **DsVdOmit** tags of subsystems have the following effects on the generated variable description file:

Setting	Description
DsVdOmit tag is set to 1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. Use <code>set_param(gcb,'Tag','DsVdOmit=1')</code> .
DsVdOmit tag is set to 0	The subsystem contents including all blocks beneath this subsystem do appear in the generated variable description file, even if a subsystem above this subsystem has set the DsVdOmit tag to 1. Use <code>set_param(gcb,'Tag','DsVdOmit=0')</code> .
DsVdOmit tag is set to -1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. DsVdOmit settings of subsystems included in this subsystem are ignored. Use <code>set_param(gcb,'Tag','DsVdOmit=-1')</code> .

If cleared, settings of the **DsVdOmit** tag are ignored for all subsystems in the model. All subsystems and their contents appear in the generated variable description file.

Exclusion is applied recursively through the model hierarchy. To reinclude subsystems, you can enter **DsVdOmit=0**. You can set the Omit flag by using workspace variables. For example, if **WSVar1** is a workspace variable, you can use `set_param(subsystemHandle,'Tag','DsVdOmit=$(WSVar1)')` to let the Omit tag value being evaluated during build process.

If you set the **Apply subsystem omission tags** option, the variables of the grayed subsystems in the illustration below are not generated into the variable description file.



Tip

This option provides an alternative to the TRC Exclusion block from the Extras library to exclude a subsystem and its contents from the generated variable description file. It can be used together with the TRC Exclusion block in the same model.

Related files [Variable Description File \(TRC File\)](#) on page 123.

Related topics**References**

[Build Options Page \(CPU Options Dialog\)](#)..... 64

Rename CPU Dialog

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog - <CPU> page - Rename CPU

Purpose

To change the name of the CPU.

Dialog settings

Enter new CPU name Enter a name for the CPU.

Note

- The CPU names used with RTI-MP must be identical to the CPU names used with the Platform Manager of ControlDesk.
- The CPU names are restricted to eight alphanumeric characters.

Related topics**References**

[CPU's Page \(Multiprocessor Setup Dialog\)](#)..... 53

Multiprocessor Topology Setup Dialog

Access

Menu bar	None
Context menu of	None
Shortcut key	None
Icon	None
Others	Multiprocessor Setup block <ul style="list-style-type: none"> ▪ Multiprocessor Setup dialog - Advanced page - Multiprocessor Network Topology

Purpose

To specify the topology of a multiprocessor system.

Description

After you connect the Gigalinks and register your multiprocessor system, you can set up the network topology by clicking the **Read topology** button. This allows RTI-MP to build an application and download it correctly to your multiprocessor system. Each row in the **Communication Connection** frame represents one board.

Tip

As an alternative, you can specify a network topology before you register your multiprocessor system. This allows you to generate code for any multiprocessor topology. To do so, specify the number and names of the involved CPUs, and select their Gigalink connections. You can also click the **Read from model** button to get all the CPU names that are specified in the model. When the multiprocessor application is downloaded to the dSPACE hardware, the Gigalink connections formerly specified must be available.

Dialog settings

File will be saved to model directory Displays the path of the model. Each time you click **OK** or **Apply**, the current settings in the Multiprocessor Topology Setup dialog are written to the `<model>_tp.m` file located in the folder that also holds the model.

Read topology (Enabled after you registered your multiprocessor system.) Sets the number and names of the connected CPUs as well as their Gigalink connections in the Multiprocessor Topology Setup dialog according to the physical Gigalink connections.

The multiprocessor topology has to be generated beforehand. You do this via the **Check Gigalink Topology** command in ControlDesk.

Read from model Sets the CPU names in the Multiprocessor Topology Setup dialog according to the CPU names used in your current multiprocessor model. The values for Gigalink connections 0 ... 3 of all CPUs in your model are set to *NC* (not connected).

Browse Lets you choose any `<model>_tp.m` file so that you can easily import the topology (CPU names and Gigalink connections) of an existing model.

Communication Connection frame

CPU Names In this edit field, you can specify the names of the CPUs of your model.

- If you click the **Read topology** button, the CPU names are taken from the multiprocessor topology you created with ControlDesk, refer to [Check Gigalink Topology \(ControlDesk Platform Management !\[\]\(750841ae7100dc832cb0a4b3af4492f3_img.jpg\)](#)).
- If you click the **Read from model** button, the CPU names are taken from the current multiprocessor model.
- If you click the **Browse** button and select a different `<model>_tp.m` file, the CPU names are imported from that file.

Note

- If you want to download an application to your multiprocessor system, the CPU names used with RTI-MP must be identical to the CPU names used with the Platform Manager of ControlDesk.
- The CPU names are restricted to eight alphanumeric characters.
- All CPU names must be unique.

Gigalink 0 ... 3 Here you can specify the Gigalink number and CPU to which you want to connect the current Gigalink.

Note

- If you click the **Read topology** button, the values are set automatically according to the physical Gigalink connections.
- If you click the **Read from model** button, the values are set to *NC* (not connected).
- If you click the **Browse** button and select a different `<model>_tp.m` file, the values are imported from that file.

Remove Removes the corresponding CPU from the network topology. The Gigalinks that were connected to this CPU are disconnected (*NC*).

Add CPU Adds a CPU to the network topology. All Gigalinks of the new CPU are disconnected (*NC*).

Reset Gigalinks Resets all Gigalinks to the *NC* setting (not connected).

Related topics**Basics**



[Gigalink Connection \(RTI and RTI-MP Implementation Guide !\[\]\(666e09182d4cd268646ea700ea60dcdf_img.jpg\)\)](#)

References

[Advanced Page \(Multiprocessor Setup Dialog\).....](#) 57
[Files Controlling the Build and Download Process.....](#) 105

Model Configuration Parameters Dialogs

Access

Ribbon/Menu	Simulink model <ul style="list-style-type: none"> For the active configuration set: MODELING - SETUP - Model Settings or SIMULATION - PREPARE - Model Settings For model-specific and reference configuration sets: MODELING - DESIGN - Model Explorer
Context menu of	None
Shortcut key	Model Explorer: Ctrl+H Model Configuration Parameters: Ctrl+E
Icon	Configuration Parameters:  Model Explorer: 

Description

A few options for the program building of multiprocessor systems must be specified in the Model Explorer's **Configuration Parameter** dialogs. RTI-MP automatically specifies valid default settings for new models.

Dialog pages

The following dialog pages contain settings relevant to RTI-MP. These options are valid for the overall RTI-MP model.

- [Math and Data Types Dialog \(Model Configuration Parameters Dialogs\)](#) on page 74 for specifying data types and net slope computations.
- [Diagnostics Dialog \(Model Configuration Parameters Dialogs\)](#) on page 75 for making specific model conditions generate messages.
- [Hardware Implementation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 76 for setting options for the characteristics of the hardware to be used to implement the system represented by your model.

- [Model Referencing Dialog \(Model Configuration Parameters Dialogs\)](#) on page 78 for setting options for including a model in other models and vice versa.
- [Simulation Target Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80 for setting options for configuring the simulation target.
- [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80 for setting certain code generation options of the RTI-MP model.

Related topics

HowTos

[How to Specify Options for the Build Process \(RTI and RTI-MP Implementation Guide !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)\)](#)

References

[Multiprocessor Setup Dialog](#)..... 48
[set_rti](#)..... 198

Math and Data Types Dialog (Model Configuration Parameters Dialogs)

Access

This dialog is contained in the **Model Configuration Parameters** dialog, refer to [Model Configuration Parameters Dialogs](#) on page 73.

Purpose

To set options for specifying data types and net slope computations.

Description

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the **Math and Data Types** dialog and its options, refer to the MATLAB documentation by MathWorks®.

Dialog settings

Implement logic signals as Boolean data (vs. double) Lets you specify whether the boolean or double data type is used for logical signals. This option can be used for compatibility with models from earlier Simulink versions where logical signals are implemented as double.

- If selected, the Simulink blocks (e.g., Logical Operator) use the **Boolean** data type for logical signals (default).
- If cleared, the Simulink blocks in the model use the **double** data type for logical signals.

For reasons of downward compatibility, this setting is also respected by some RTI Bit I/O blocks.

For general information, refer to the Simulink documentation.

Diagnostics Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 73.
Purpose	To specify options that make specific model conditions generate either error messages or warnings.
Description	<p>On this page you can specify options to check the overall RTI-MP model and the code generation process. The Diagnostics dialog contains the following pages:</p> <ul style="list-style-type: none"> ▪ The Solver page lets you specify the diagnostic actions Simulink takes when it detects a solver-related error. ▪ The Sample Time page lets you specify the diagnostic actions Simulink takes when it detects an error relating to the sample times in your model. ▪ The Data Validity page lets you specify the diagnostic actions Simulink takes when it detects an error that could have an impact on the data integrity of your model. ▪ The Conversion page lets you specify the diagnostic actions Simulink takes when it detects a data type conversion error during compilation of your model. ▪ The Connectivity page lets you specify the diagnostic actions Simulink takes when it detects an error relating to block connections. ▪ The Compatibility page lets you specify the diagnostic actions Simulink takes when it detects an error relating to compatibility between the current Simulink version and your model. ▪ The Model Referencing page lets you specify the diagnostic actions Simulink takes when it detects an error relating to Model Referencing. <p>In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Diagnostics dialog and its options, refer to the Simulink documentation.</p>

Data Validity page **Signal resolution** Lets you select how Simulink software resolves signals to Simulink.Signal objects. The following settings are possible:

Setting	Description
Explicit only (default)	Performs only explicitly specified signal resolution. This is the recommended setting.
Explicit and implicit	Performs implicit signal resolution wherever possible, without posting any warnings about the implicit resolutions.
Explicit and warn implicit	Performs implicit signal resolution wherever possible, posting a warning of each implicit resolution that occurs.

Model Verification block enabling Lets you globally enable or disable all the Model Verification blocks or use the settings of the individual blocks.

- If *Enable all* is selected, all the **Model Verification** blocks are active.

- If *Disable all* is selected, none of the **Model Verification** blocks is active.
- If *Use local settings* is selected, only the **Model Verification** blocks with the *Enable assertion* setting selected are active (default).

To control the behavior of the **Model Verification** blocks in the real-time simulation, use RTI's **Assertion mode** setting. Depending on how you set this, one of the following actions is carried out if a verified signal leaves the desired range, meaning that the assertion occurs:

- **OFF**: No reaction (default).
- **WARN**: Issue a warning message in the Log Viewer of ControlDesk.
- **STOP**: Issue an error message and stop the simulation, i.e., set the simulation state to *STOP*.

For details on this technique, refer to [How to Use Simulink's Model Verification Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\)](#)).

Hardware Implementation Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 73.
Purpose	<p>To set options for the characteristics of the hardware to be used to implement the system represented by your model.</p> <p>In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Hardware Implementation dialog and its options, refer to the Simulink documentation.</p>
Description	The Hardware Implementation dialog lets you specify the characteristics of the hardware to be used to implement the system represented by your model. All the configuration settings are contained in the default model template that is used if you create a new model.

The following values are then automatically configured on the Hardware Implementation dialog, depending on your dSPACE board:

Property	DS1007 DS1104 MicroAutoBox II MicroLabBox	DS1006
Device vendor	Custom Processor	Custom Processor
Number of bits		
char	8	8
short	16	16
int	32	32
long	32	32
long long ¹⁾	64	64
native	32	32
pointer	32	32
size_t	32	32
ptrdiff_t	32	32
Byte ordering	Big Endian	Little Endian
Signed integer division rounds to	Zero	Zero
Shift right on a signed integer as arithmetic shift	True	True

¹⁾ To enable this data type, you must set the Support long long option in the dialog.

If the **Ensure deterministic data transfer** option of the Rate Transition block is cleared, and the signal size is less than or equal to the specified largest atomic size, the code generator removes double-buffering and semaphore protection from the generated code. This optimization reduces memory consumption and improves execution speed.

RTI and RTI-MP check if the hardware implementation settings are specified correctly.

For the largest atomic sizes of integer and floating-point variables, dSPACE only supports the following combinations:

- Largest atomic size of integer variables: **Int**
- Largest atomic size of floating-point variables: **Float**

Note

To enable generation of 64-bit integer variables into the TRC file, you must set the Support long long option.

Model Referencing Dialog (Model Configuration Parameters Dialogs)

Access This dialog is contained in the **Model Configuration Parameters** dialog, refer to [Model Configuration Parameters Dialogs](#) on page 73.

Purpose To set options for including a model in other models and vice versa.

Description In this dialog you can specify options for building simulation (SIM) and Simulink Coder (RTW) targets relevant when:

- including this model in other models
- including other models in this model

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the **Model Referencing** dialog and its options, refer to the Simulink documentation.

Options for all referenced models **Rebuild** Lets you determine whether SIM and RTW targets are rebuilt before updating, simulating, or generating code. The following options are available:

Option	Description
Always	SIM and RTW targets are always rebuilt before updating, simulating, or generating code from the model.
If any changes detected	SIM and RTW targets are rebuilt if Simulink detects any changes of any kind in the target's dependencies. This is the default setting.
If any changes in known dependencies	SIM and RTW targets are rebuilt if Simulink detects any changes in known dependencies since the target was last built. You can define dependencies via the Model dependencies option described below.
Never	SIM and RTW targets are never rebuilt before updating, simulating, or generating code from the model.

Tip

Use the *Never* option carefully since it may lead to incorrect results if referenced targets are not really up-to-date.

Never rebuild diagnostic Lets you specify the diagnostic action that Simulink should perform if it detects a target that needs to be rebuilt. This option is available only if you set the **Rebuild** option to *Never*. You can select the following options:

- Error if rebuild required (default)
- Warn if rebuild required

- None

Note

Selecting *None* accelerates updating, simulation and code generation, but can generate invalid results.

Enable parallel model reference builds Lets you specify whether to build the referenced models in parallel on a multicore PC. To enable the parallel build feature the Parallel Computing Toolbox is required.

MATLAB worker initialization for builds Lets you specify how the MATLAB workers will be initialized prior to building on them. It is enabled if you set the Enable parallel model reference builds option. You can choose *None* (default), *Copy base workspace* and *Load top model*.

Options for referencing this model

Total number of instances allowed per top model Lets you specify how often the model can be referenced in another model. You can choose between one, multiple (default) or zero instances. This setting influences the appearance of referenced models in the variable description file (TRC file). For details, refer to [Available Variables in the Variable Description File](#) on page 127.

With RTI-MP, the number of instances refers to the main model and each of the submodels.

Propagate sizes of variable-size signals Lets you specify when to propagate the sizes of variable-size signals. You can choose *Infer from blocks in model*, *Only when enabling* and *During execution*.

Minimize algebraic loop occurrences Lets you specify whether Simulink should eliminate algebraic loops involving the model from models that reference it.

Pass fixed-size scalar root inputs by value for code generation Lets you specify how a model that references this model passes scalar inputs to this model. If enabled, scalar inputs to this model are passed by value. Otherwise, scalar inputs are passed by reference (default). Passing roots by value can have advantages in performance. It can lead to incorrect simulation results, though.

Model dependencies Lets you specify on which files the model relies. They are typically **.mat** and **.m** files used to initialize and provide data. This information is used by Simulink and Simulink Coder to detect if the model must be rebuilt.

Related topics

Basics

[Limitations with Model Referencing \(RTI and RTI-MP Implementation Guide\)](#)

[Referencing Models \(RTI and RTI-MP Implementation Guide\)](#)

Simulation Target Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 73.
Purpose	To set options for configuring the simulation target for a model that contains, e.g., MATLAB Function blocks.
Description	In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the Simulation Target dialog and its options, refer to the MATLAB documentation by MathWorks®.
Dialog settings	<p>Block reduction This option is not supported by RTI. Make sure that it is not selected.</p> <p>Conditional input branch execution Lets you specify whether to execute only the blocks required to compute the control input and the data input selected by the control input of Switch and Multiport Switch blocks. If the option is set, ControlDesk display the last computed values of the output signals of those blocks, which are executed because of the current switch condition.</p> <p>To ensure that the blocks providing signals connected to your experiment software are computed, turn off the Conditional input branch execution option and rebuild the model.</p>

Code Generation Dialog (Model Configuration Parameters Dialogs)

Access	This dialog is contained in the Model Configuration Parameters dialog, refer to Model Configuration Parameters Dialogs on page 73.
Purpose	To set Simulink Coder and RTI-MP code generation options.
Description	<p>In this dialog you can specify options relating to the code generation process.</p> <p>The following settings are relevant to RTI-MP. For the settings relevant to RTI, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) on page 25. For a detailed description of the Code Generation dialog and its options, refer to the Simulink documentation.</p>

Dialog settings

Check model before generating code Lets you choose whether to run Code Generation Advisor checks before generating code. The following settings are available:

Option	Description
Off	Generates code without checking whether the model meets code generation objectives (default).
On (proceed with warnings)	Checks whether the model meets code generation objectives. If the Code Generation Advisor reports a warning, they are displayed in the Model Advisor Window. The build process continues.
On (stop for warnings)	Checks whether the model meets code generation objectives. If the Code Generation Advisor reports a warning, they are displayed in the Model Advisor Window. The build process stops.

Check Model Lets you run Code Generation Advisor checks.

Note

Code Generation Advisor checks are general code generation checks. Running these checks does not necessarily check model settings that are relevant to RTI. RTI checks model settings relevant to RTI automatically during the build process, independently of any checks performed by the Code Generation Advisor.

Generate code only Lets you generate only C code from your Simulink model.

- If selected, the model code is only generated. The code is neither compiled nor downloaded to the hardware. Files that are relevant to the generated real-time application, such as the TRC file, are not generated.
- If cleared, the model code is generated, compiled, and downloaded to the dSPACE hardware (default).

Optimization page

Default parameter behavior Lets you inline the block parameters or generate them as variables.

- If **Inlined** is selected, the block parameters are inlined, so their concrete values are used in the generated C code, which makes them non-modifiable. This typically reduces memory consumption and execution time.
To generate individual parameters as tunable variables, see the **Configure** option below. Simulink.Parameter objects in the workspace can also be used to specify the tunability of a parameter, refer to *Rules for Simulink.Signal and Simulink.Parameter objects* in [Available Variables in the Variable Description File](#) on page 127.
- If **Tunable** is selected, the block parameters are generated as variables, so they are modifiable during the real-time simulation (default).

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Configure If the **Default parameter behavior** option is set to **Inlined**, you can make individual parameters tunable.

If the **Default parameter behavior** option is set to **Tunable**, you can specify the storage class for individual parameters.

For details, refer to [Model Parameter Configuration Dialog](#) on page 85.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Use memcpy for vector assignment Lets you optimize code generated for vector assignment by replacing for-loops with **memcpy** function calls.

- If the checkbox is selected, the associated **Memcpy threshold (bytes)** parameter is enabled (default). **memcpy** is used in the generated code if the number of array elements times the number of bytes per element is greater than or equal to the specified value for the **Memcpy threshold (bytes)** parameter. One byte equals the width of a character.
- If the checkbox is cleared, the use of **memcpy** for vector assignment is disabled.

Memcpy threshold (bytes) Lets you specify the array size in bytes at or above which **memcpy** function calls should replace for-loops in the generated code for vector assignment. The default array size is 64.

Signal storage reuse Lets you control the memory allocation for block output variables.

- If selected, Simulink Coder does not always use a separate block output variable for each block output but attempts to assign a single buffer to several block outputs. This typically reduces memory consumption. However, reused variables are not available in the variable description file.

You can exclude individual block outputs from this optimization to make them available in the variable description file.

Note

You can also use the *Enable local block outputs* option to generate block output variables as local function variables as far as possible.

- If cleared, a separate global variable is used for each block output (default).

For details, refer to the Simulink documentation.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

Loop unrolling threshold Lets you make Simulink Coder place a sequence of individual statements in a **for** loop, if required.

To configure a CPU-specific loop rolling threshold, you can add the **-aRollThreshold=<positive_number>** option for the CPU to the Custom TLC options edit field on the Build Options Page (CPU Options Dialog) of the [CPU Options Dialog](#) on page 64.

Inline invariant signals Lets you inline invariant signals.

- If selected, the invariant signals of the model are inlined.
- If cleared, invariant signals are generated as variables (default).

Note

This option is active only if you selected the Inline parameters option.

Refer to [Default parameter behavior](#) on page 81.

Invariant signals are constant signals or signals that are directly derived from constants. Normally, they are generated as variables so that they are modifiable during the real-time simulation. However, if they are inlined, their concrete values are used in the generated C code, which makes them nonmodifiable. This typically reduces memory consumption and execution time.

Inlined signals are not available in the variable description file. You can therefore exclude individual block outputs from this optimization to include them in the variable description file.

For details on this optimization option, refer to the Simulink documentation.

Related files: [Variable Description File \(TRC File\)](#) on page 123.

To inline the invariant signals for a specific CPU, you can add the **-aInlineInvariantSignals=1** option for the CPU to the Custom TLC options edit field on the Build Options Page (CPU Options Dialog) of the [CPU Options Dialog](#) on page 64.

Enable local block outputs Lets you generate block output variables as local function variables as far as possible.

- If cleared, all block output variables are generated as global function variables (default).
- If selected, Simulink Coder attempts to generate block output variables as local function variables.

This typically reduces memory consumption or execution time. Several block output variables might not be available in the variable description file. You can therefore exclude individual block outputs from this optimization to include them in the variable description file.

Note

This option is active only if you selected the Signal storage reuse option. Refer to [Simulation Target Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80.

To enable the local block outputs for a specific CPU, you can add the **-aLocalBlockOutputs=1** option for it to the Custom TLC options edit field on the Build Options Page (CPU Options Dialog) of the [CPU Options Dialog](#) on page 64.

For details on this optimization option, refer to the user documentation by MathWorks.

Related files: [Variable Description File \(TRC File\)](#) on page 123

Reuse local block outputs Lets you reuse signal memory whenever possible.

- If selected, Simulink Coder attempts to assign a single buffer to several signals.

Reused variables are not available in the variable description file. You can therefore exclude individual signals from this optimization to make them available in the variable description file.

- If cleared, local function variables are unique (default).

Note

This option is active only if the Signal storage reuse option is set to On. Refer to [Simulation Target Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80

Eliminate superfluous temporary variables (Expression folding) Lets you optimize the computation time for intermediate results at block outputs and the storage of results stored in temporary variables.

Remove code from floating point to integer conversions with saturation that maps NaN to zero Lets you determine if code should be removed when mapping from NaN values to integer zero occurs.

- If the checkbox is selected, code is removed when NaN values are mapped to integer zero (default).
- If the checkbox is cleared, code is not removed when NaN values are mapped to integer zero.

Note

Selecting the checkbox produces results that do not match the simulation in the case of NaN values. If the checkbox is cleared, the results for the simulation and the execution of generated code match.

Use memset to initialize floats and doubles to 0.0 Lets you determine whether code is generated that explicitly initializes floating-point data to 0.0.

- If the checkbox is selected, `memset` is used to clear internal storage for floating-point data to integer bit pattern 0 (all bits 0), regardless of type (default).
- If the checkbox is cleared, code is generated that explicitly initializes storage for data of types float and double to 0.0. The generated code is slightly less efficient than code generated when you select the checkbox.

Identifiers page

Maximum identifier length Lets you specify the maximum number of characters used in functions, type definitions, and variable names in the range 31 ... 256. The default length is 31.

Use the same reserved names as Simulation Target Lets you specify whether to use the same reserved names as those defined for the simulation target. Reserved names are names of variables or functions in the generated code that match the names of variables or functions specified in custom code.

- If the checkbox is selected, the same reserved names as those defined for the simulation target are used. When selected, this parameter disables the Reserved names parameter.

- If the checkbox is cleared, the reserved names defined for the simulation target are not used (default).

Reserved names Lets you enter reserved names. This action changes the names of variables or functions in the generated code to avoid name conflicts with identifiers in custom code. Reserved names must be shorter than 256 characters.

Related topics

Basics

Building and Downloading the Model (RTI and RTI-MP Implementation Guide )
Introducing the Build and Download Process (RTI and RTI-MP Implementation Guide )

References

Code Generation Dialog (Model Configuration Parameters Dialogs).....	25
Model Parameter Configuration Dialog.....	85

Model Parameter Configuration Dialog

Access	The Configure command to open this dialog is available on the Optimization page of the Code Generation dialog, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) on page 80.
--------	--

Purpose	To declare specific parameters as tunable parameters.
---------	---

Description	You can make individual parameters tunable if the Default parameter behavior option is set to Inlined.
-------------	--

Dialog settings	This section gives only a brief summary of the dialog settings. For a detailed description of the Tunable Parameters dialog and its options, refer to the Simulink documentation.
-----------------	---

- | | | |
|-----------------------------------|-------------------------------|---|
| Global (tunable) parameters frame | New | Adds a new tunable parameter. |
| | Remove | Removes a tunable parameter. |
| | Name | Enter the parameter name here. |
| | Storage class | Select the storage class for the parameter. |
| | Storage type qualifier | Enter the storage type qualifier here. |

Source List frame

From the drop-down list, select whether you want to have all *MATLAB workspace* variables or only the *Referenced workspace variables* displayed in the source list.

The source list displays all the workspace variables that correspond to the selection of the drop-down list. The variables that are made global (tunable) parameters are displayed in bold italics.

Refresh list Updates the list from the workspace/model.

Add to table >> Adds the highlighted variable(s) to the Global (tunable) parameters list.

Related topics**References**

[Code Generation Dialog \(Model Configuration Parameters Dialogs\)..... 80](#)

Signal Properties Dialog

Access

Menu bar	None
Context menu of	Signal
Shortcut key	None
Icon	None

Purpose

To specify the properties of the selected signal.

Description

You can make a signal displayable even if certain code optimization options are selected.

In this reference you will find only the settings relevant to real-time simulation. For a detailed description of the **Signal Properties** dialog and its options, refer to the Simulink documentation.

The Simulink **Signal Properties** dialog contains the following pages:

- [Logging and accessibility page](#) on page 87
- [Code Generation page](#) on page 87
- *Documentation* page (not relevant to RTI)

Signal name Specify the name of the signal if desired.

Logging and accessibility page	Test Point	Select this checkbox if you want to make the signal global.
---------------------------------------	-------------------	---

Code Generation page	Storage class	Select the storage class for the signal if desired.
-----------------------------	----------------------	---

RTI and RTI-MP Variable Reference

Introduction

The real-time application provides a number of variables that contain information about the application. You can use, for example, ControlDesk or dSPACE XIL API .NET to read them from the real-time hardware.

Some variables show different behavior depending on where the host service is executed. Consider the **overrunQueueCount** and **state** variables of a particular task, for example:

- **overrunQueueCount** is always **0** if read in the background task. This is due to the fact that the background task has the lowest priority and is therefore executed only if no other task is queued. If **overrunQueueCount** is read from within its own task, its value is always **1** or higher because it is incremented when the task is called and decremented when the task is finished.
- **state** is always **idle** (0) if read in the background task. If **state** is read from within its own task, it is always **running** (2), and if read from any other task, it can be **running** (2), **ready** (1) or **idle** (0).

If you use one of ControlDesk's instruments to access a variable, ControlDesk automatically uses the host service that is executed in the background task.

If you trace a variable via ControlDesk instruments, you can specify the desired host service via the Measurement Configuration controlbar.

Where to go from here

Information in this section

Main group

[currentTime](#).....90

To show the current simulation time.

[errorNumber](#).....91

To show the type of the last error.

[finalTime](#).....92

To show the time when the simulation is terminated.

modelStepSize	92
To show the base sample time of the model.	
simState	97
To read or set the simulation state of the application.	
Task Info group	
overrunCheckType	93
To show the behavior of the simulation in task overrun situations.	
overrunCount	94
To help you analyze overrun situations.	
overrunQueueMax	95
To help you analyze overrun situations.	
priority	96
To show the priority of a task.	
sampleTime	97
To show the sample time of a task.	
state	98
To show the current state of a task.	
taskCallCount	99
To help you analyze task execution.	
turnaroundTime	100
To show the turnaround time of a task.	

currentTime

Purpose

To show the current simulation time.

Description

This read-only variable is incremented by the fastest timer task with the smallest sample time available in the model. Its unit is seconds.

When the application is paused, `currentTime` stops incrementing. When the application is stopped, `currentTime` is reset to zero.

Note

The dSPACE systems also provide the time-stamping feature. However, the `currentTime` variable is calculated by Simulink Coder using floating-point numbers, whereas the time stamps are calculated by the dSPACE Real-Time Kernel using integer numbers. Since the precision of floating-point numbers decreases the larger the numbers become, the `currentTime` variable and the associated time stamps might differ slightly.

Related variable [modelStepSize](#) on page 92

TRC file group Main

Related topics Basics

[Time-Stamping and Data Acquisition \(RTI and RTI-MP Implementation Guide !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)](#))

errorNumber

Purpose To show the type of the last error.

Description The value of this read-only variable indicates the type of the last error. It shows zero if no error occurred. If it has a value other than zero, you can use the Log Viewer of ControlDesk for the exact error message and the issuing component.

TRC file group Main

Related topics**Basics**

[Reaction to Run-Time Errors \(RTI and RTI-MP Implementation Guide !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)\)](#)

HowTos

[How to Send Messages to the Message Module \(RTI and RTI-MP Implementation Guide !\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\)\)](#)

finalTime

Purpose

To show the time when the simulation is terminated.

Description

The unit of this read-only variable is seconds.

The **finalTime** variable displays the stop time setting **inf** as **-1**.

Corresponding options

RTI Stop time (refer to [Solver Dialog \(Model Configuration Parameters Dialogs\)](#) on page 15)

RTI-MP Stop time (refer to [Main Page \(Multiprocessor Setup Dialog\)](#) on page 49)

TRC file group

Main

modelStepSize

Purpose

To show the base sample time of the model.

Description

The unit of this read-only variable is seconds.

Corresponding options	RTI Fixed step size (fundamental sample time) on page 16
	RTI-MP Basic step size on page 50, Step size on page 54

TRC file group	Main
-----------------------	------

overrunCheckType

Purpose	To show the behavior of the simulation in task overrun situations.
----------------	--

Description	This read-only variable displays the task's overrun check type configured in the RTI Task Configuration dialog. Possible values are:
--------------------	--

Value	Type
0	ignore overrun
1	stop simulation
2	queue task before simulation stop

Corresponding options	RTI Ignore overrun (refer to RTI Task Configuration Dialog on page 39), Queue task before simulation stop (refer to RTI Task Configuration Dialog on page 39), Stop simulation (refer to RTI Task Configuration Dialog on page 39), Max. queued task instances (refer to RTI Task Configuration Dialog on page 39)
	RTI-MP Ignore overrun (refer to RTI Task Configuration Dialog (Multiprocessor Setup Dialog) on page 59), Queue task before simulation stop (refer to RTI Task Configuration Dialog (Multiprocessor Setup Dialog) on page 59), Stop simulation (refer to RTI Task Configuration Dialog (Multiprocessor Setup Dialog) on page 59), Max. queued task instances (refer to RTI Task Configuration Dialog (Multiprocessor Setup Dialog) on page 59)

TRC file group	Task Info
-----------------------	-----------

Related topics**HowTos**

[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(eafc244b53721dd1ec133f0772f70fc7_img.jpg\)\)](#)

References

overrunCount	94
overrunQueueCount	95
overrunQueueMax	95
RTI Task Configuration Dialog	39
taskCallCount	99

overrunCount

Purpose

To help you analyze overrun situations.

Description

The `overrunCount` variable of a task counts the total number of task overruns that occurred for it.

TRC file group

Task Info

Related topics**HowTos**

[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(b538fe54c1f3a7343e37e85cc2d00497_img.jpg\)\)](#)

References

overrunCheckType	93
overrunQueueCount	95
overrunQueueMax	95
RTI Task Configuration Dialog	39
taskCallCount	99

overrunQueueCount

Purpose	To help you analyze overrun situations.
Description	The <code>overrunQueueCount</code> variable displays the number of queued task instances. The number is incremented for each queued task request and decremented each time the task finishes execution.
TRC file group	Task Info

Related topics

Basics

[Overrun Situation and Turnaround Time \(RTI and RTI-MP Implementation Guide !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)\)](#)

HowTos

[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(642aa997563f9a325b310230bb5078b7_img.jpg\)\)](#)

References

overrunCheckType	93
overrunCount	94
overrunQueueMax	95
RTI Task Configuration Dialog	39
taskCallCount	99

overrunQueueMax

Purpose	To help you analyze overrun situations.
Description	The <code>overrunQueueMax</code> variable displays the number specified for Max. queued task calls in the RTI Task Configuration dialog.
TRC file group	Task Info

Related topics**Basics**

[Overrun Situation and Turnaround Time \(RTI and RTI-MP Implementation Guide !\[\]\(3d8c13c92b853674f749aac6fa869926_img.jpg\)\)](#)

HowTos

[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)\)](#)

References

overrunCheckType.....	93
overrunCount.....	94
overrunQueueCount.....	95
RTI Task Configuration Dialog.....	39
taskCallCount.....	99

priority

Purpose

To show the priority of a task.

Description

This variable displays the task priority, defined in the RTI Task Configuration dialog.

Corresponding dialogs

RTI [RTI Task Configuration Dialog](#) on page 39

RTI-MP [RTI Task Configuration Dialog \(Multiprocessor Setup Dialog\)](#) on page 59

TRC file group

Task Info

Related topics**Basics**

[Priorities and Task-Switching Time \(RTI and RTI-MP Implementation Guide !\[\]\(08ff79f060f3543d9ed549cc693d8b98_img.jpg\)\)](#)

References

RTI Task Configuration Dialog.....	39
state.....	98

sampleTime

Purpose	To show the sample time of a task.
Description	This read-only variable is available only for timer tasks. Its unit is seconds. In the single timer task mode its value is the same as the modelStepSize; in the multiple timer task mode its value depends on the sample time of the corresponding blocks.
TRC file group	Task Info
Related topics	References <div> modelStepSize..... 92 </div>

simState

Purpose	To read or set the simulation state of the application.
Description	<p>This variable can take on the states STOP (0), PAUSE (1), or RUN (2).</p> <div> <p>Note</p> <p>The <i>PAUSE</i> simState is not supported by DS1007 and MicroLabBox (DS1202).</p> </div> <p>RTI-MP For multiprocessor systems the simulation is controlled by a single CPU: the <i>master CPU</i> (on the Main page of the Multiprocessor Setup dialog, this CPU is marked with an (M)). Therefore, the simState variable is only accessible on the master CPU.</p>
TRC file group	Main (RTI-MP: Main group of the master CPU's TRC file)

Related topics**Basics**

[Simulation Control \(RUN/STOP Mechanism\) \(RTI and RTI-MP Implementation Guide !\[\]\(99f58673407353e96a019fbca558fd72_img.jpg\)\)](#)

HowTos

[How to Set the Simulation State \(RTI and RTI-MP Implementation Guide !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\)\)](#)

References

[Multiprocessor Setup Dialog..... 48](#)

state

Purpose

To show the current state of a task.

Description

This read-only variable can take on the following task states:

State	Value	Meaning
Idle	0	The task is inactive, waiting to be triggered.
Ready	1	The task has been triggered but could not start due to a high-priority task that is currently executing. It is waiting to start the execution.
Running	2	The task has started its execution. This state is true until the task finishes its execution, regardless of whether it is suspended by a high-priority task or not.

Some changes of the **state** variable might not be observed by ControlDesk. For example, if the state of a task changes faster than the ControlDesk service code is executed, then state switches cannot be recorded because the service code is in another task. It is not possible to observe changes of the **state** variable of the task that is running the ControlDesk service code.

TRC file group

Task Info

Related topics

Basics

[Task States and Execution Order \(RTI and RTI-MP Implementation Guide !\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\)](#))

References

[priority..... 96](#)

taskCallCount

Purpose To help you analyze task execution.

Description The `taskCallCount` variable of a task is incremented whenever an interrupt triggers a task registration. It does not include the task calls that caused overrun situations.

TRC file group Task Info

Related topics

HowTos

[How to Specify Overrun Strategies for Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(0d5ec72f61334709c3fc9450209b754f_img.jpg\)](#))

References

[overrunCheckType..... 93](#)
[overrunCount..... 94](#)
[overrunQueueCount..... 95](#)
[overrunQueueMax..... 95](#)
[RTI Task Configuration Dialog..... 39](#)

turnaroundTime

Purpose

To show the turnaround time of a task.

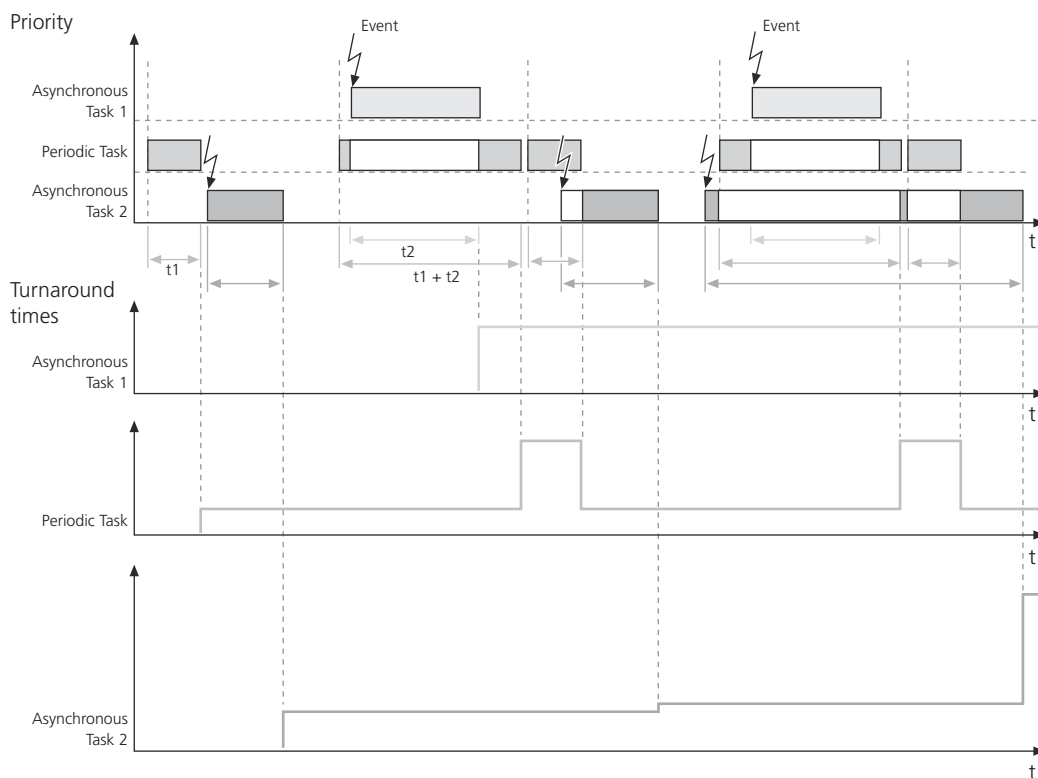
Description

This read-only variable shows the turnaround time, which consists of:

- The time it takes to execute the functional code of the task
- The period it takes to trace variables if the task contains the host service code (provided that ControlDesk is active)
- The time the task's execution is interrupted by the execution of other tasks with higher priorities
- The turnaround time does not include the task switching time.

The turnaround time is specified in seconds.

The illustration below shows the turnaround times for a number of tasks.



Note

Conditional execution paths in the model code can cause the turnaround time of a task to vary from one execution step to another. For example, if you run your model in the multiple timer task mode, the fastest task actually contains only blocks that are executed with this sample time. Nevertheless the turnaround time of this task might have periodical peaks occurring with the period of a slower task, which result from the interaction with the slower timer tasks.

- If the slower and faster tasks exchange data, the faster task contains Zero-Order Hold or Unit Delay blocks that are executed with the period of the slower task.
- To ensure that all the inputs (ADC, digital input, etc.) of the timer tasks are read almost simultaneously, parts of the code generated for RTI input blocks are always executed in the fastest task even if the RTI block is part of a task with a longer sample time.

TRC file group

Task Info

Related topics

Basics

[Overrun Situation and Turnaround Time \(RTI and RTI-MP Implementation Guide !\[\]\(0b5e7e25e8775f7e7e80906ada4f0021_img.jpg\)\)](#)

RTI and RTI-MP File Reference

Introduction

Several intermediate files are generated during or required by the build process. These are located in the \<[sub]model>_<target>\ build folder(s), where <target> is your active platform support. The current working folder contains user-supplied files (for example, <[sub]model>_usr.*) and the files that are needed for the real-time application (such as the real-time object, TRC file, map file, etc.).

Where to go from here

Information in this section

File Overview.....	104
File Details.....	112
SDF File Syntax.....	133
Syntax of the TRC File.....	138
The TRC file (variable description file) provides information on the variables of a real-time application that is required to connect variables to instruments in a ControlDesk layout, for example.	

File Overview

Introduction

Many files are created and involved in the build and download process for a model:

Where to go from here

Information in this section

Simulink Files.....	104
Files Generated by RTI and RTI-MP.....	105
Files Controlling the Build and Download Process.....	105
Executable Real-Time Object and System Description Files.....	107
Variable Access Files.....	108
User-Supplied Files.....	109
File Specifics for Model Referencing.....	110

Simulink Files

Model files

The following file required by RTI and RTI-MP is a Simulink file.

Simulink model The model file (<[sub]model>.slx) contains the Simulink model. With RTI-MP, the CPU-specific <submodel>.slx file contains the Simulink model for a submodel running on the specified CPU. The submodel is automatically extracted from the main RTI-MP model during the build process or can be provided by the customer.

Protected Simulink models have the suffix slxp.

Related topics

Basics

[Details on MP Systems \(RTI and RTI-MP Implementation Guide !\[\]\(e3f255517d37bb309a3a931ec4849e6a_img.jpg\)](#))

Files Generated by RTI and RTI-MP

Generated files

The following files are generated by Real-Time Interface.

<[sub]model>_th.c This file is generated by RTI and defines the different tasks of the (sub)model. It is used to initialize the dSPACE Real-Time Kernel with the desired tasks. With RTI-MP, it is CPU-specific.

<[sub]model>_rti.c This file is generated by RTI and contains definitions and declarations that are mainly related to the I/O functionality of the model or submodel. With RTI-MP, it is CPU-specific.

<[sub]model>_usr.c RTI generates this template file for the User-Code if it does not already exist. You can place user-written C code in the file. The code is included in the application. With RTI-MP, it is CPU-specific.

For further information, refer to [User-Code File \(USR.C File\)](#) on page 118.

<submodel>_simeng.c This CPU-specific file is generated by RTI-MP and controls the simulation for the corresponding submodel.

<submodel>_ipc.h This CPU-specific file is generated by RTI-MP and contains all the defines and macros needed for interprocessor communication (IPC).

Files Controlling the Build and Download Process

Miscellaneous files

The following files control the build and download process.

<model>_tp.m During the build process, RTI-MP maps the logical IPC channels that are specified via IPC blocks in the RTI-MP model to the physical network topology, which is described in this file. Whenever you change the Gigalink connections of a system, you have to update the file via RTI-MP's Multiprocessor Topology Setup dialog.

For further information, refer to [Multiprocessor Topology Setup Dialog](#) on page 71.

rtimp_<model>_info.html In each build process, RTI-MP stores information about the CPU IDs, the topology of the multiprocessor system and the mapping of the IPC channels for that topology. You might need this information for debugging because some messages contain only the IDs of the affected CPUs.

rti<xxxx>.tlc This is the system target file. It controls the generation of the model code and the RTI-specific files.

This system target file is valid for single-processor systems and multiprocessor systems.

rti<xxxx>.tmf This is the template makefile. It contains all commands to compile and link a model or submodel for a certain dSPACE processor board. The platform-specific template makefile (**rti<xxxx>.tmf**) provides the basis to

automatically implement a C-coded Simulink model for real-time execution. It contains all the commands to compile and link a model or submodel for a certain dSPACE system. When generating new code for a model or submodel, Simulink Coder derives the application-specific makefile `<[sub]model>.mk` from the template makefile. With RTI-MP, the application-specific makefile is CPU-specific.

For further information, refer to [<\[sub\]model>.mk](#) on page 106 and [User Makefile \(USR.MK File\)](#) on page 119.

<[sub]model>.mk Simulink Coder derives this application-specific makefile from the template makefile, and inserts model-specific macro variables, like the integration algorithm for the submodel and the names of the S-functions that are used. With RTI-MP, it is CPU-specific.

For further information, refer to [rti<xxxx>.tmf](#) on page 105 and [User Makefile \(USR.MK File\)](#) on page 119.

<[sub]model>_rti.mk RTI generates this model-specific include makefile. It is incorporated into the `<[sub]model>.mk` file.

<[sub]model>_usr.mk This is the user makefile. It lets you modify the standard build process via a set of make macros. You can define extra search paths (for S-functions or user-defined C source code), user libraries, etc.

For further information, refer to [User Makefile \(USR.MK File\)](#) on page 119.

<[sub]model>_rti.prj RTI uses this model-specific project marker file to determine whether the source files need to be recompiled.

<[sub]model>.lk The linker command file describes the assignment of memory sections to the memory banks of the corresponding processor.

For further information, refer to [Linker Command File \(LK File\)](#) on page 122.

clean.bat This RTI-MP-specific batch file deletes all the temporary files that are generated by RTI-MP on each build process. For further help on this command, enter `clean /?` in a Command Prompt window.

Note

This batch file does not consider temporary files from referenced models generated during an RTI-MP build process, because they could be required for further top-level models.

You should consider using the `rtimp_build2` command instead.

download.bat This RTI-MP-specific batch file downloads a multiprocessor application to the dSPACE hardware. For further help on this command, enter `download /?` in a Command Prompt window.

You should consider using the `rtimp_build2` command instead.

<model>.dsbuildinfo This file is used to provide additional build information. It is created or updated if a build process has been successfully finished. For RTI-MP, the entire MP application including the build results of the submodels must be available.

This file is used by other programs, for example, ControlDesk, to determine whether there is a newer application file available than the application file loaded.

Related topics

References

[rtimp_build2..... 194](#)

Executable Real-Time Object and System Description Files

Files required for working with your application

The following files are the real-time object files that are executed on your real-time hardware. In addition, there is the system description file that binds the object file(s) and variable description file(s) together for easy access, for example, in ControlDesk.

<[sub]model>.ppc RTI generates this file during the build process. It is the executable object file for the DS1104 board and MicroAutoBox II. After the build process is finished, it can be downloaded to the dSPACE real-time hardware.

When you start a build process, the existing file is deleted.

<[sub]model>.x86 RTI generates this file during the build process. It is the executable object file for the DS1006 Processor boards. After the build process is finished, the file can be downloaded to the dSPACE real-time hardware.

When you start a build process, the existing file is deleted.

<model>.rta RTI generates this file during the build process. It is the executable object file for DS1007 PPC Processor boards and MicroLabBox. After the build process is finished, the file can be downloaded to the dSPACE real-time hardware.

When you start a build process, the existing file is updated. It will also be updated, if you, for example, rename the application or rename the CPU.

<model>.sdf This file (called the system description file) is generated during each build process. It contains information about the CPU name(s), the corresponding object file(s) to be downloaded and the corresponding variable description file(s).

Note

The **<model>.sdf** file is generated during each build and download process. Additional user-specific entries, such as for loading DS230x applications, must be made in the **<model>_usr.sdf** file.

<model>_usr.sdf RTI and RTI-MP support an optional user-specific system description file (**<model>_usr.sdf**). If the file exists in the working folder, its contents are incorporated into the **<model>.sdf** file, which RTI and RTI-MP

generate for the application. You can also merge the file into the `<model>.sdf` file manually using the `rti_sdfmerge` command.

For information on the SDF file syntax, refer to [SDF File Syntax](#) on page 133.

Related topics

References

[rti_sdfmerge](#)..... 193

Variable Access Files

Generated files for variable access

The following files allow you to access variables of the real-time simulation.

<[sub]model>.trc RTI generates this variable description file, which contains the description of all signals and parameters of the model or submodel. It also contains the addresses of the variables. The signals and parameters are ordered according to the hierarchical structure of the (sub)model. The file is referenced by the SDF file of the real-time application to allow access to the signals and parameters of the application, for example, via dSPACE's ControlDesk. With RTI-MP, it is CPU-specific.

When you start a build process, the existing file is deleted.

<[sub]model>_usr.trc This is the user variable description file, which you can create yourself. If it exists, it is automatically inserted into the variable description file (`<[sub]model>.trc`) during the build process. It lets you specify additional variables such as entries for internal variables of S-functions or user-defined libraries. With RTI-MP, it is CPU-specific.

For more information, refer to [User Variable Description File \(USR.TRC File\)](#) on page 132. For a description of the file syntax, refer to [Syntax of the TRC File](#) on page 138.

For further information, refer to [Variable Description File \(TRC File\)](#) on page 123.

<[sub]model>.map The linker generates this linker map file, which describes the memory layout of an application. For example, ControlDesk, uses it to retrieve variable addresses. With RTI-MP, it is CPU-specific.

When you start a build process, the existing file is deleted.

<[sub]model>.trt This is an intermediate file for the TRC file generation.

<[sub]model>.mapx This is an intermediate file for the TRC file generation.

Related topics

References

[Syntax of the TRC File](#)..... 138

User-Supplied Files

Customizable files

The following files allow you to customize the real-time application and the build and download behavior.

<model>_tp.m During the build process, RTI-MP maps the logical IPC channels that are specified via IPC blocks in the RTI-MP model to the physical network topology, which is described in this file. Whenever you change the Gigalink connections of a system, you have to update the file via RTI-MP's Multiprocessor Topology Setup dialog.

For further information, refer to [Multiprocessor Topology Setup Dialog](#) on page 71.

<[sub]model>_usr.c RTI generates this template file for the User-Code if it does not already exist. You can place user-written C code in the file. The code is included in the application. With RTI-MP, it is CPU-specific.

For further information, refer to [User-Code File \(USR.C File\)](#) on page 118.

<[sub]model>_usr.mk This is the user makefile. It lets you modify the standard build process via a set of make macros. You can define extra search paths (for S-functions or user-defined C source code), user libraries, etc.

For further information, refer to [User Makefile \(USR.MK File\)](#) on page 119.

<model>_usr.sdf RTI and RTI-MP support an optional user-specific system description file (**<model>_usr.sdf**). If the file exists in the working folder, its contents are incorporated into the **<model>.sdf** file, which RTI and RTI-MP generate for the application. You can also merge the file into the **<model>.sdf** file manually using the `rti_sdfmerge` command.

For information on the SDF file syntax, refer to [SDF File Syntax](#) on page 133.

<[sub]model>_usr.trc This is the user variable description file, which you can create yourself. If it exists, it is automatically inserted into the variable description file (**<[sub]model>.trc**) during the build process. It lets you specify additional variables such as entries for internal variables of S-functions or user-defined libraries. With RTI-MP, it is CPU-specific.

For more information, refer to [User Variable Description File \(USR.TRC File\)](#) on page 132. For a description of the file syntax, refer to [Syntax of the TRC File](#) on page 138.

rti1xxx_template_rtimphook.m RTI-MP provides a hook function to customize the model separation and the update of submodels. To use the hook function, a specific M file must be available when applying settings in the Multiprocessor Setup dialog or during the build process. For this M file, a template is available in the Templates sublibrary of the RTI-MP blockset. There you also find an example file.

For details on the file, refer to [rti1xxx_template_rtimphook.m](#) on page 116.

Related topics

References

[rti_sdfmerge](#)..... 193

File Specifics for Model Referencing

Specific information when using model referencing

When the build process for a model is started, several files are created into the current working and build folder. This applies both to models with and without Model blocks. The file types generated for top-level models are exactly the same as the file types generated for models without Model blocks. However, for referenced models there are some file specifics which are listed in the following table:

File Type	Description
User-code file (USR.C)	<ul style="list-style-type: none"> It is supported for top-level models only. For referenced models it is not supported. If a user-code file exists for a referenced model, for example because you have converted a former RTI model to a referenced model, it is ignored.
User makefile (USR.MK file)	<ul style="list-style-type: none"> It is supported for each referenced model, except for protected models, in the same way as it is supported for a top-level model. Once a referenced model is built for which a user makefile exists, changes in the user makefile are not automatically detected. This means that they are not considered during the next build process of the top-level model except you specified the user makefile in the Model dependencies option in the Model Referencing dialog. For details, refer to Model Referencing Dialog (Model Configuration Parameters Dialogs) on page 22.
System description file (SDF file)	It is only supported for the top-level model in a model reference hierarchy, but not for referenced models.
User system description file (USR.SDF)	<ul style="list-style-type: none"> It is supported for top-level models only. For referenced models it is not supported. If a user system description file exists for a referenced model, for example because you have converted a former RTI model to a referenced model, it is ignored.
Variable description file (TRC file)	Variable descriptions applying to referenced models are supported under certain conditions. They are always generated into the variable description file of the top-level model. There are important variable description file specifics for models containing Model blocks. For details, refer to Available Variables in the Variable Description File on page 127.
User variable description file (USR.TRC file)	Each referenced model, except for protected models, can have a specific <model>_usr.trc file.

File location

The files listed above are contained in the current working folder. During the code generation phases for the top-level and referenced model all user-supplied

files are saved in the current working folder. The system description file, the variable description file and the files needed for the real-time application are generated into the current working folder during each build process of the top-level model.

Related topics

Basics

[Available Variables in the Variable Description File..... 127](#)
[Referencing Models \(RTI and RTI-MP Implementation Guide !\[\]\(9dfdaff1d86ba3c1f8353b4d1b61b8c5_img.jpg\)\)](#)

References

[File Overview..... 104](#)

File Details

Introduction

When you work with RTI and RTI-MP, you will encounter different work step-specific files types.

Where to go from here

Information in this section

Customizing the simulation environment

[startup.m](#)..... 113

To carry out user-specific commands before the dSPACE software is initialized.

[dsstartup.m](#)..... 114

To carry out user-specific commands after the dSPACE software is initialized.

[dspoststartup.m](#)..... 114

To carry out user-specific commands after the initialization phase of the dSPACE software.

[dsfinish.m](#)..... 115

To carry out user-specific commands before MATLAB is shut down.

Inserting custom C code

[User-Code File \(USR.C File\)](#)..... 118

To add hand-written C code to the real-time application.

Make process

[User Makefile \(USR.MK File\)](#)..... 119

To define extra search paths (for S-functions or user-defined C source code), user libraries, etc.

[Linker Command File \(LK File\)](#)..... 122

To provide the linker program with the necessary information on where to place the various code sections in the memory of the processor/controller board.

[User System Description File \(USR.SDF File\)](#)..... 123

To customize the download procedure.

Variable access in dSPACE software, such as ControlDesk

[Variable Description File \(TRC File\)](#)..... 123

To provide the necessary information about the signals and parameters of the application, which allows you to observe and manipulate the signals and parameters of the simulation.

Variable Description File Groups.....	125
To list different groups of signals and parameters contained in the variable description file.	
Available Variables in the Variable Description File.....	127
To show rules that define whether a variable is included in the variable description file.	
Data Type Specifications.....	131
To specify data types.	
User Variable Description File (USR.TRC File).....	132
To add variables of custom code to the variable description file, which makes them available for data capture and modification in the simulation.	

startup.m

Purpose	To carry out user-specific commands during the dSPACE software is initialized.
Result	Each time you start MATLAB or activate a different RTI platform support, this M file is executed after the dSPACE software is initialized.
Description	<p>Place this M file in the MATLAB search path, for example, in MATLAB’s working folder. In general, all valid MATLAB commands are allowed in it. Use the standard M file syntax.</p> <p>The startup.m file is a standard MATLAB feature. For further information type doc startup at the MATLAB prompt.</p>

Related topics

HowTos

[How to Customize the MATLAB Start-Up \(RTI and RTI-MP Implementation Guide 📖\)](#)

References

[dsstartup.m..... 114](#)

dsstartup.m

Purpose	To carry out user-specific commands after the dSPACE software is initialized.
Result	Each time you start MATLAB or switch to a different platform, this M-file is executed after the dSPACE software is initialized.
Description	Create a new M file and place it in the MATLAB search path, for example, in MATLAB's working folder. In general, all valid MATLAB commands are allowed in it, including the dSPACE-specific commands. Use the standard M file syntax.
Example	<p>Suppose you want MATLAB to automatically change to your working folder <code>d:\work</code> and open the Simulink model <code>my_model.slx</code>. Create the <code>dsstartup.m</code> file and write the following commands to it:</p> <pre>cd d:\work my_model</pre>

Related topics

HowTos

[How to Customize the MATLAB Start-Up \(RTI and RTI-MP Implementation Guide !\[\]\(339a16584d5da0f0a3ca4e9ec17bf6a1_img.jpg\)\)](#)

References

dsfinish.m	115
dspoststartup.m	114
startup.m	113

dspoststartup.m

Purpose	To carry out user-specific commands after the initialization phase of the dSPACE software.
Result	Each time you start MATLAB or switch to a different platform, this M-file is executed after the initialization phase of the dSPACE software.

Description

If you create the M file representing this optional script, you can add generally all valid MATLAB commands and dSPACE-specific commands in it. The file must be placed in the MATLAB search path. It is executed directly in the MATLAB base workspace, which is necessary for executing some specific MATLAB functions.

The script is also executed if the initialization of the dSPACE software failed. It is executed after `dsstartup.m`.

Related topics

HowTos

[How to Customize the MATLAB Start-Up \(RTI and RTI-MP Implementation Guide !\[\]\(b93c3e1add16fe46100bba7a6da1e82f_img.jpg\)](#))

References

[dsstartup.m..... 114](#)

dsfinish.m

Purpose

To carry out user-specific commands before MATLAB is shut down.

Result

Each time you exit MATLAB, this M file is executed.

Description

Create a new M file and place it in the MATLAB search path, for example, in MATLAB's working folder. All valid MATLAB commands are allowed in it, including the dSPACE-specific commands. Use the standard M file syntax.

Note

- You can specify several `dsfinish.m` files. All `dsfinish.m` files located in the MATLAB search path are executed before MATLAB is shutdown.
- For compatibility reasons available user-specific `finish.m` files are executed. This behavior can, however, change with future MATLAB versions. It is therefore recommended to use `dsfinish.m` files.

Related topics**HowTos**

[How to Customize the MATLAB Start-Up \(RTI and RTI-MP Implementation Guide !\[\]\(dfbd6b3763a6d1d9afaa974f64e2e4b5_img.jpg\)\)](#)

References

[dsstartup.m](#)..... 114
[startup.m](#)..... 113

rti1xxx_template_rtimphook.m

Purpose

To provide a hook function to customize the model separation and the update of submodels in an RTI-MP model.

Result

The hook function is executed according to its configured methods when you apply settings in the Multiprocessor Setup dialog or during the build process.

Description

You find the template file in the Templates sublibrary of the RTI-MP blockset. An example file is also available. When double-clicking the RTI-MP Hook Template block, you are asked to copy the file to your current MATLAB working directory. This must be the directory in which the RTI-MP model you want to customize is stored.

Rename the file to

`rti<PlatformIdentifier>_<CustomIdentifier>_rtimphook.m`.

The following platforms support RTI-MP and can be set as the platform identifier:

- 1006
- 1007
- 1202

Syntax

```
function [errorCode, errorMsg, varargout] =
    rti1xxx_template_rtimphook(callmethod, rtimphookStruct, varargin)
```

Change the function name to the specified file name. You must not modify the function input arguments.

Argument	Description
callmethod	This parameter is passed by RTI-MP. The following hook methods are available: <ul style="list-style-type: none"> ▪ PreUpdate

Argument	Description
	<p>This function is called before the update of a submodel configuration and can be used to set configuration parameters.</p> <ul style="list-style-type: none"> ▪ PostUpdate This function is called after the update of a submodel configuration and can be used to set parameters that have been added to the configuration during the update. ▪ PreSeparation This function is called before the model separation and can be used to set configuration parameters of the RTI-MP model. ▪ PostSeparation This function is called after the model separation and can be used to set parameters of blocks that have been created during the separation, i.e., IPC and IPI blocks.
rtimpHookStruct	This parameter is passed by RTI-MP and can be read in the custom code to get the model name and submodel name.

The function output arguments provide the error code and an error message.

Argument	Description
errorCode	<p>Provides an integer value:</p> <ul style="list-style-type: none"> ▪ 0: No error ▪ 1: An error occurred
errorMsg	Provides a string with a description of the error.

The file provides four sections for the supported hook methods. The lines in which you are allowed to add custom code are marked by **BEGIN_USER_CODE** and **END_USER_CODE**.

Only one *_rtimphook.m file is allowed in your model directory. You can specify the custom code for all four hook methods in the same file.

Example

You find an example file in the **Templates** sublibrary of the RTI-MP blockset. By double-clicking the **RTI-MP Hook Example** block, you are asked to copy the file to your current MATLAB working directory.

The example shows you how to set the Simulink block priority of the IPC blocks in the **PostSeparation** hook.

Related topics

References

User-Supplied Files..... 109

User-Code File (USR.C File)

Purpose

To add hand-written C code to the real-time application.

Description

The User-Code file `<[sub]model>_usr.c` is included in both the initialization part and the evaluation sequence of the timer task that is executed at the base sample rate. Therefore, you can incorporate access to I/O devices into a real-time program via the User-Code file. For information on how to write a User-Code file and transfer data from and to the Simulink model, refer to [Implementing User-Code \(RTI and RTI-MP Implementation Guide\)](#).

RTI generates a model-specific template User-Code file during the first build process for the corresponding Simulink model. If this file already exists, no new template file is generated. The template User-Code file contains the following C functions, which are automatically called by RTI:

usr_initialize() The user-specific code that you want to be executed during program initialization, for example, user-specific initialization of I/O devices, must be placed within this function. It is called after the model initialization function `MdlStart()`, which originates from the Simulink model.

usr_sample_input() The user-specific code that you want to use for sampling input devices (start converters and read digital input devices) must be placed within this function. In the RUN mode it is executed before the inputs of the RTI input blocks are sampled.

usr_input() This function also holds function calls for reading input devices. In the RUN mode it is executed after the inputs of the RTI input blocks are sampled but before the model code of the Simulink model is executed. It can also be used for special purposes such as overwriting values already read by RTI.

usr_output() This function holds function calls for writing to output devices. In the RUN mode it is executed after the model code of the Simulink model.

usr_terminate() The code to be executed when the real-time application terminates must be placed within this function, for example, to write particular termination values to the output devices whenever the simulation is stopped. In the real-time application, it is executed whenever the `simState` variable is changed to STOP mode. The function is executed after the model termination function `MdlTerminate()`, which originates from the Simulink model.

usr_background() If your real-time application has to execute some code in the background task, place the code within this function.

Related topics

Basics

[Execution Order of S-Functions and User-Code \(RTI and RTI-MP Implementation Guide !\[\]\(feabb98897b440bc8695a03336a6e2df_img.jpg\)\)](#)
[Implementing User-Code \(RTI and RTI-MP Implementation Guide !\[\]\(c7f935293d8062fa748ed86b74d28761_img.jpg\)\)](#)

References

[simState.....97](#)

User Makefile (USR.MK File)

Purpose To define extra search paths (for S-functions or user-defined C/C++ source code), user libraries, etc.

Description The user makefile lets you modify the standard build process by defining additional paths, files and dependencies.

Tip

As an alternative to the user makefile you can use the custom code settings on the Code Generation dialog (Model Configuration Parameters dialog). They do not offer all the features of the RTI user makefile, but the most frequently used ones. Unlike the user makefile, the custom code settings are part of the model configuration and are not stored in a separate file. This makes model sharing easier.

The user makefile provides a set of macros for additional settings. The following paragraphs list the different options, which you can find in the <[sub]model>_usr.mk file under the headline **Macros for user-specific settings**.

Note

- You can use the backslash character \ as a line-continuation character.
- Do not indent the options via spaces or tabs. However, you may indent the parameters of an option if they are continued in a new line. For example:

```
SFCN_DIR = \project1\sfcns \
           \project2\sfcns
```

- If path names contain whitespaces they need to be set in double quotes. For example:

```
SFCN_DIR = "\project one\sfcns" \
           "\project two\sfcns"
```

- Whitespaces and double quotes are not supported with file names like "my source.c".
- Path and file names containing "#" characters are not supported. The "#" character is a reserved character of the make utility to mark comments.
- Path and file names containing "\$" characters can lead to unexpected results. The "\$" character is a reserved character of the make utility.

Tip

You can use environment variables or make macros when specifying file or path names, for example, in the user makefile you could write

```
USER_LIBS = $(PROJECTPATH)\libraries\mylib.lib
```

If you do not want to make PROJECTPATH an environment variable, you can specify it as a make macro, for example:

- RTI : make_rti PROJECTPATH="C:\project1"
- RTI-MP : Make options: PROJECTPATH="C:\project1"

USER_BUILD_CPP_APPL The C++ support is enabled by setting the make macro to ON. The C++ support must be enabled for each model separately, also for referenced models.

```
# Enable C++ support
USER_BUILD_CPP_APPL = ON
```

For more information, refer to [Using C++ Code in an RTI Application \(RTI and RTI-MP Implementation Guide !\[\]\(e3f255517d37bb309a3a931ec4849e6a_img.jpg\)](#)).

SFCN_DIR These folders are used as the search path for the source code of S-functions. As a result you can collect the source code of frequently used S-functions in these common folders and do not need to keep it in the current working folder. For example:


```
# Directories where S-Function C source files are stored.
SFCN_DIR = \project1\sfcns \
          "\project two\sfcns"
```

Note

If you also want to place the corresponding MEX DLL files in the S-function folder, you have to add it to the MATLAB search path as well.

USER_SRCS The additional C/C++ source code files are compiled and linked. These files need to have the extension `.c/.cpp`, which marks them as C/C++ source code files. They have to be located in the working folder of your model or on the search path that is specified via the `USER_SRCS_DIR = <...>` make macro. For example:

```
# Additional C/C++ source files to be compiled
USER_SRCS = mysource1.c mysource2.cpp
```

Note

Only the following file extensions are supported:

- `.c`
- `.cpp`
- `.h`

Tip

You do not need to specify S-function source files that are used in the Simulink model because their names are incorporated automatically in the list of source files to be compiled.

USER_ASM_SRCS The additional assembler source files are assembled and linked. They need to have the extension `.asm`, which marks them as assembler source code files. They have to be located in the working folder of your model or on the search path that is specified via the `USER_SRCS_DIR = <...>` make macro. For example:

```
# Add. assembler source files to be compiled (file ext .asm).
USER_ASM_SRCS = mysource1.asm mysource2.asm
```

USER_SRCS_DIR These folders are used as the search path for the C and assembler source code files that are declared via the `USER_SRCS = <...>` and `USER_ASM_SRCS = <...>` make macros. For example:

```
# Directories where add. C and assembler source files are
stored.
USER_SRCS_DIR = \project1\usr \
                "\project two\usr"
```

USER_INCLUDES_PATH These folders are used as the search path for include files that are used in the source code of S-functions or User-Code. For example:

```
# Path names for user include files.
USER_INCLUDES_PATH = \project1\sfcns \
                    "\project two\usr"
```

USER_OBJS The precompiled object files are used to supply functions that you can use in the source code of S-functions and User-Code. For example:

```
# Additional user object files to be linked
USR_OBJS = module1.obj module2.cppo50
```

The file name extension can vary and depends on the platform (for example, o40 or o05).

USER_LIBS These libraries are used to supply functions that you can include in the source code of S-functions and User-Code. For example:

```
# Additional user libraries to be linked
USR_LIBS = \project1\libs\usr1.lib \
          "\project two\cpplibs\usr2.lib"
```

Dependencies It is quite common for user source files to depend on other files; for example, a `source.c` file needs to be recompiled if it includes a `header.h` file via the `#include` directive and that header file is modified. You can include such dependencies (even to several files) in the user makefile, for example:

```
source.c : header.h
source2.c : header1.h header2.h
```

Ensure you include blanks before and after each colon.

Related files

- [rti<xxxx>.tmf](#) on page 105
- [<\[sub\]model>.mk](#) on page 106

Related topics

Basics

[Tips and Tricks for Custom C Code \(RTI and RTI-MP Implementation Guide !\[\]\(f60b7a900783ac3fd531bfd9c111be6d_img.jpg\)\)](#)
[Using C++ Code in an RTI Application \(RTI and RTI-MP Implementation Guide !\[\]\(fe5cf1978663f480c504f8fc2019fe62_img.jpg\)\)](#)

Linker Command File (LK File)

Purpose

To provide the linker program with the necessary information on where to place the various code sections in the memory of the processor/controller board.

Description

The linker command file is specially adapted to the needs of Simulink Coder/RTI-built programs so that the placement of the different sections and the sizing of the program heap are optimal for a wide range of different applications. For some applications it is desirable to work with a modified local

copy of the original linker command file. As a linker command file is tightly coupled to a specific application, the naming of a local copy must be `<model>.lk` or `<submodel>.lk`. Depending on your processor/controller board, you can use one of the following files as a starting-point for your application-specific linker command file:

Processor / Controller Board	Corresponding Linker Command File
DS1006	<code><RCP_HIL_InstallationPath>\... ...\DS1006\RTLib\ds1006.lk</code>
DS1007	<code>...\DS1007\Lib\LinkerScript.ld</code>
DS1104	<code>...\DS1104\RTLib\ds1104.lk</code>
MicroLabBox	<code>...\DS1202\Lib\LinkerScript.ld</code>
MicroAutoBox II	<code>...\DS1401\RTLib\ds1401.lk</code>

User System Description File (USR.SDF File)

Purpose

To customize the download procedure.

Description

The user system description file allows you to include in the download process object files that are not connected to the main application running on a dSPACE processor board, for example, DS230x applications. Whenever you build the model, RTI and RTI-MP incorporate the contents of the user system description file in the generated system description file (`<model>.sdf`). You can also merge the file into the `<model>.sdf` file manually using the `rti_sdfmerge` command.

For information on the SDF file syntax, refer to [SDF File Syntax](#) on page 133.

Related topics

References

[rti_sdfmerge](#)..... 193

Variable Description File (TRC File)

Purpose

To provide the necessary information about the signals and parameters of the application, which allows you to observe and manipulate the signals and parameters of the simulation, for example, by using ControlDesk.

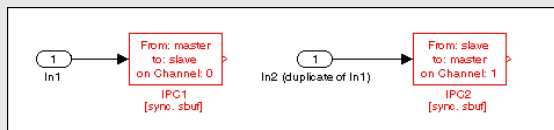
Description

The structure of the variable description file reflects the hierarchy of the underlying Simulink model. The file can contain signals, parameters, simulation control variables and task information variables.

When you build a model, RTI and RTI-MP automatically generate the necessary variable description file(s) <[sub]model>.trc. You can directly alter the generation of this file by various options. For details, refer to [RTI variable description file options page](#) on page 35 (RTI) or [Variable Description File Options Page \(CPU Options Dialog\)](#) on page 67 (RTI-MP).

Note

- RTI and RTI-MP also provide a system description file (SDF) that bundles all the TRC files and additional information for the application. You should use the SDF file to access the signals and parameters of the application, for example, via ControlDesk.
- The variables of custom C code are not automatically included in the variable description file. To access them via ControlDesk you have to add them manually to the user variable description file (see [User Variable Description File \(USR.TRC File\)](#) on page 132 for details).
- For S-functions, TRC files do not support variables other than outputs and parameters. Use the USR.TRC file instead (see [User Variable Description File \(USR.TRC File\)](#) on page 132).
- For variables of fixed-point data type, the TRC file contains scaling information.
- Signal generators and viewers defined with the Signals & Scope Manager are not generated into the TRC file.
- In the TRC file, duplicate input ports are displayed as common input ports. RTI-MP checks if the duplicate input ports are modeled correctly. For example, specifying the same input signal with duplicate input ports as coming from different CPUs is not permitted.

**Aspects of the variable description file**

The following aspects are important for using the variable description file:


- The variable description file contains various groups that serve different purposes (see [Variable Description File Groups](#) on page 125).
- Whether a variable is available in the variable description file depends on certain conditions, for example on the use of model referencing (see [Available Variables in the Variable Description File](#) on page 127).
- Depending on the hardware used, the data types in the variable description file might differ from those defined in the Simulink model (see [Data Type Specifications](#) on page 131).

- For information on how to load a variable description file in ControlDesk, see [How to Add a Variable Description to a Platform/Device \(ControlDesk Variable Management !\[\]\(3da2b303d29c1ea489bbe26a3f5ac664_img.jpg\)](#)).
- For details on the options that control the generation of TRC files, see [RTI variable description file options page](#) on page 35 (RTI) or [Variable Description File Options Page \(CPU Options Dialog\)](#) on page 67 (RTI-MP).


Related files [User Variable Description File \(USR.TRC File\)](#) on page 132

Related topics

Basics

Excluding Subsystems from the TRC File (RTI and RTI-MP Implementation Guide 

HowTos

How to Exclude a Subsystem from the TRC File (RTI and RTI-MP Implementation Guide 

References

TRC Exclusion Block.....

254

User Variable Description File (USR.TRC File).....

132

Variable Description File Options Page (CPU Options Dialog).....

67

Variable Description File Groups

Groups in the variable description file

The groups of the variable description file contain various signals and parameters. The following paragraphs list the different groups of the variable description file together with a short description.

Main This group has the name of the (sub)model. It contains the simulation control variables, which allow online observation and manipulation of the simulation. The following simulation control variables are available:

- [currentTime](#) on page 90
- [errorNumber](#) on page 91
- [finalTime](#) on page 92
- [modelStepSize](#) on page 92
- [simState](#) on page 97 (on MP systems only for the master CPU)

Note

When using model referencing, the simulation control variables appear only for the top-level model. These variables do not exist for referenced models.

Model Root Contains the block and subsystem groups and the labeled signals for the top level of the Simulink model. Variables from the contained blocks and subsystems are located in lower hierarchical levels.

- Each subsystem group contains the labels defined in the subsystem, and the block groups for the contained blocks.
- Each block group contains the variables of the block, for example, outputs, parameters and states.
- Block groups for Stateflow® charts contain the outputs to Simulink, Stateflow test points and parameters.

All Stateflow charts together form the state machine, which can have global data. This is available via the State Machine Data group.

Labels Contains all labeled signals and appears only if labeled signals are found in the Simulink model. In this group, labels are treated as globals, and the model hierarchy is not taken into account to give quick access to important signals. The labeled signals are also available in the subsystem groups from which they originate.

If a label occurs more than once in a model or a subsystem, no entries are generated into the variable description. To activate TRC file entries for multiple labels, refer to [ds_trc_multiplelabeloccurrence](#) on page 184.

User Variables from <[sub]model>_usr.trc Is filled with the user-specific contents of the user variable description file <[sub]model>_usr.trc whenever code is generated from the Simulink model. It is created only if a user variable description file is available. See also [User Variable Description File \(USR.TRC File\)](#) on page 132.

Task Info Contains a separate subgroup for each task in the model. The subgroups are named after the corresponding task or hardware/software interrupt block, for example, Timer Task 1, CrankEvent. Each subgroup contains the following variables:

- [overrunCheckType](#) on page 93
- [overrunCount](#) on page 94
- [overrunQueueCount](#) on page 95
- [overrunQueueMax](#) on page 95
- [priority](#) on page 96
- [sampleTime](#) on page 97
- [state](#) on page 98
- [taskCallCount](#) on page 99
- [turnaroundTime](#) on page 100

Note

When using model referencing, the task info variables appear only for the top-level model. These variables do not exist for referenced models.

State Machine Data All the Stateflow charts of a model form the state machine. This group contains data objects of scope Exported defined at the State Machine level. The group is only created if the model contains Stateflow charts. Data objects of scopes Output, Parameter or Local defined for individual Stateflow charts are available via the Model Root group. Depending on the MATLAB release used, Data objects of scopes Local and Output are only generated into the variable description file if the value attribute Test point is set.

Tunable Parameters Collects the global parameters of the model, e.g., MATLAB workspace variables used as block parameters. It is also created if there is no global parameter specified in the model. Refer to Model Parameters Configuration Dialog (RTI) and Model Parameters Configuration Dialog (RTI-MP). See also *Rules for Simulink.Signal and Simulink.Parameter* in [Available Variables in the Variable Description File](#) on page 127.

Related topics

Basics	
Available Variables in the Variable Description File.....	127
References	
Model Parameter Configuration Dialog.....	38
RTI and RTI-MP Variable Reference.....	89

Available Variables in the Variable Description File

Introduction

Not all variables of a Simulink model are always available in the variable description file. The following rules define whether a variable is included.

Rules for dSPACE blocks

- The following rules apply to the dSPACE blocks:
- Since the parameters of most I/O blocks cannot be altered during the simulation, the groups for non-tunable I/O blocks are empty.
 - For performance reasons, RTI usually generates no code for unused I/O block channels. These are channels that are unconnected or connected only to Ground, Terminator, Inport, Outport, Goto or From blocks (in Simulink terminology, all these blocks are *virtual*). As a result, these channels are not available in the variable description file or are marked with the description *No data (unused channel)*.

Rules for virtual Simulink blocks

By default, the virtual blocks are not available in the variable description file. If you want to include them, you can set the Include virtual blocks option. Refer to *Include virtual blocks* in the [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25 (RTI) or on the [Variable Description File Options Page \(CPU Options Dialog\)](#) on page 67 (RTI-MP).

- To have the outputs of a virtual block in the variable description, you have to add a test point to it.
 - Virtual signals are not generated into the variable description.
 - Structured parameters, bus arrays, and non-virtual Simulink buses are generated into the variable description and can be directly accessed.
-

Rules for Simulink.Signal and Simulink.Parameter objects

`Simulink.Parameter` and `Simulink.Signal` objects can be used to specify the characteristics of tunable parameters and labeled signals in a model.

`Simulink.Parameter` and `Simulink.Signal` objects are represented in the variable description file with the following properties:

- Description
- DocUnits
 - Is displayed as *Unit* in your experimentation software.
- Min and Max (represented by a range in the TRC file)

Note the following preconditions and restrictions:

- The properties are considered for labels in the Labels group and the subsystem groups, if the labeled signal references a `Simulink.Signal` object of global storage class (`ExportedGlobal`, `ImportedExtern`, `ImportedExternPointer`). The properties are not considered for block output signals and Sink Inputs within the block groups.
 - The properties are considered for parameters in the Tunable Parameters group, if the parameters are specified by `Simulink.Parameter` objects of global storage class (`ExportedGlobal`, `ImportedExtern`, `ImportedExternPointer`).
 - The range for Min and Max is generated only, if both values are given.
 - The range for Min and Max is not generated for fixed-point data.
-

Rules for boolean value in Simulink

Signals and parameters of boolean data type in Simulink are recognized as boolean value by your experimentation software. The TRC file entry describes a value of boolean data type as an `UInt8` data type with a range of 0:1.

Rules for nonvirtual bus signals

A nonvirtual bus is generated as a structured variable into the variable description file. The name of a structure field matches the element name in the bus.

Rules for n-dimensional look-up tables

For the Lookup Table (n-D), Direct Lookup Table (n-D), and Interpolation (n-D) using PreLookup blocks, RTI generates one or more `LookUpTableData` entries into the variable description file:

- $n = 1$: one `LookUpTableData` vector
- $n = 2$: one 2-dimensional `LookUpTableData` array
- $n > 2$: one 1-dimensional `LookUpTableData` array

Rules for optimization methods

There are a number of code optimization methods that usually make signals and parameters unavailable in the variable description file:

- With the Signal storage reuse option selected, block output variables may be reused by several blocks. Refer to *Signal storage reuse* in the [Simulation Target Dialog \(Model Configuration Parameters Dialogs\)](#) on page 24 (RTI) or in the [Simulation Target Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80 (RTI-MP).
- With local block outputs enabled, output variables may be created as local function variables, and not as global variables. Refer to [Enable local block outputs](#) on page 83.
- With inlining of parameters enabled, block parameters are not available as variables in the code. Refer to *Inline parameters* in the [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25 (RTI) or in the [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80 (RTI-MP).
- With inlining of invariant signals enabled, constants or signals that depend only on constants are not available as variables in the code. Refer to [Inline invariant signals](#) on page 82.

Rules for complex variables

Code, generated with Simulink Coder, stores the real and imaginary parts of complex variables separately. Therefore, the display of the variable dimensions in ControlDesk is different from the one displayed in MATLAB or Simulink: Twice as many rows or columns are displayed in comparison to the Simulink model.

Rules for model referencing

The following list shows the specifics regarding model referencing:

- The TRC file contains the variables of the top-level model and of all the referenced models. The variables for the referenced models are stored in groups in a similar way to variables of subsystems.
- Parameters defined in the model workspace of referenced models can be accessed via the `Model Parameters` group of the referenced model. Block-level references to these parameters can be accessed via the referenced model group of the respective model.
- The TRC file information for a referenced model is rebuilt only if new code is generated for the model. Incremental code generation thus also includes incremental TRC file generation.

- Whether variables of a referenced model are written to the TRC file depends on the setting of the **Total number of instances per top model** option in the **Model Referencing** dialog. If the total number of instances is set to
 - "One", the contents of the referenced model appear in the TRC file.
 - "Multiple" (default), the contents of the referenced model do not appear in the TRC file. This means that such a model can be used to build real-time applications, but you cannot access internal variables for the model via the TRC file.

For details, refer to the Simulink online help by MathWorks.

- Variables of Inport blocks located on the root level of a referenced model are never generated into the TRC file. This also applies to the outputs of virtual blocks (Mux, Demux, Goto, From, subsystems), which are directly connected to such an Inport block.
- It can happen that block outputs of blocks which drive an Outport block located on the root level of a referenced model are not generated. To solve this problem, a test point can be set for the respective signal.
- Like subsystem outputs, Model block outputs are also contained in the TRC file.
- The following applies to the entries for block parameters of referenced models:

If the **Default parameter behavior** of the referenced model is set to **Tunable**, the tunable parameters are available:

- In the **Tunable Parameters** group, if they are defined globally.
- In the **Model Parameters** group, if they are defined in the model workspace.
- Directly at the block, if they are defined locally.

If the **Default parameter behavior** of the referenced model is set to **Inlined**, parameters are available in the TRC file, which are specified as tunable via *Simulink.Parameter* objects.

- Consider the following regarding the use of options on the **RTI variable description file options** page:
 - The **Include states** and **Include Derivatives** options are not supported for model referencing, neither for top-level models nor for referenced ones. If you select these options you are informed during the build process that they are ignored.
 - All the other options on the **RTI variable description file options** page are supported for top-level as well as for referenced models.
For details on the above-mentioned options, refer to [RTI variable description file options page](#) on page 35.
 - The setting for the **Include only Simulink.Parameter** and **Simulink.Signal** objects with **global storage class** option must be the identical for all models in a Model Referencing hierarchy.

When you use protected models, note the following points:

- The protected model must be generated with the same MATLAB version and dSPACE product version, with which it will later be used.

- The protected model must contain the generated code for the RTI platform, with which it will later be used.


Rules for monitoring variables for electrical error simulation (EESPort)

When you simulate electrical errors by using an EESPort implementation via the ControlDesk user interface or via a dSPACE XIL API application, you are able to monitor the switching behavior of your electrical error simulation hardware.

The generated variable description file provides the following five variables in the XIL API/EESPort group:


- Active ErrorSet
- Error Activated
- Error Switching
- Flags (for internal use)
- Trigger (for internal use)

The variables are available for each platform except for DS1104. For multiprocessor models, the XIL API group is generated into each submodel section separately.

For more information, refer to [Monitoring the Switching Behavior of Electrical Error Simulation Hardware](#) (dSPACE XIL API Implementation Guide ).

Related topics

Basics

Referencing Models (RTI and RTI-MP Implementation Guide )

References

Code Generation Dialog (Model Configuration Parameters Dialogs).....

File Specifics for Model Referencing.....

Variable Description File Options Page (CPU Options Dialog).....

25

110

67

Data Type Specifications

Data types

Integer data types on the real-time hardware have the lengths specified in the Simulink model. **boolean** is a uint8, **double** is a 64-bit floating-point type and **single** is a 32-bit floating-point variable.


User Variable Description File (USR.TRC File)

Purpose

To add variables of custom code to the variable description file, which makes them available for data capture and modification in the simulation.

Description

When writing a user variable description file, follow the syntax defined for the variable description file (refer to [Syntax of the TRC File](#) on page 138), and name it `<[sub]model>_usr.trc`. During the build process, the file is inserted into the variable description file. It must be created before the build process is started. It has to be located in the working folder.

For instructions on how to create a variable description file, refer to [How to Make Custom Variables Accessible](#) (RTI and RTI-MP Implementation Guide .

Note

You can only access variables, for example, in ControlDesk, that are declared as global and non-static in the C source code.

Related topics

Basics

[Limitations for Custom Code](#) (RTI and RTI-MP Implementation Guide )

HowTos

[How to Make Custom Variables Accessible](#) (RTI and RTI-MP Implementation Guide )

References

Syntax of the TRC File	138
Variable Description File (TRC File)	123

SDF File Syntax

Where to go from here

Information in this section

SDF File Syntax: Sections and Keys.....	133
Example SDF Files.....	137

SDF File Syntax: Sections and Keys

SDF files

A system description (SDF) file is a variable description file that describes the files to be loaded to the individual processing units of a dSPACE simulation platform.

SDF file generation An SDF file is generated automatically when the application is built, for example, with RTI.

However, you have to generate the SDF file yourself for a handcoded application.

Principle SDF file structure and syntax An SDF file is an ASCII file that is composed of *sections*, each of which contains pairs of *keys* and key values. The order of keys in a section and the order of sections in a file is irrelevant. The SDF file syntax is similar to the syntax of INI files.

Note

The values of sections and keys are not case-sensitive.

Example SDF files Refer to [Example SDF Files](#) on page 137.

[System] section

The [System] section contains the general settings of the system and the related application. An SDF file contains exactly one [System] section.

The table below shows the possible keys of the [System] section:

Key ¹⁾	Description
Version (1)	Specifies the version of the SDF file format.
SystemType (1)	Specifies the type of the system. Possible values: <ul style="list-style-type: none"> ▪ SingleProcessorSystem ▪ MultiProcessorSystem
Status ²⁾ (0 ... 1)	Specifies the state of the system after loading the SDF file to the simulation platform. If no Status key is specified, the Start value is used as the default Status.

Key ¹⁾	Description
RTP (1 ... *) Board (0 ... *) File (0 ... 1)	<p>Possible values:</p> <ul style="list-style-type: none"> ▪ Load (the application is loaded but not started) ▪ Start (the application is loaded and started) <p>Specifies a reference to an [<RTP_name>] section.</p> <p>Specifies a reference to a [<Board_name>] section (only DS230x).</p> <p>Specifies the name of the application that can be loaded to and executed by the system.</p> <div> Note The File key in the [System] section and in the [<RTP_name>] sections are exclusive: If the [System] section contains a File key, the [<RTP_name>] sections must not contain File keys, and vice versa. </div>
DAQService (1 ... *) DsDAQBackground (0 ... 1)	<p>Specifies the measurement service(s) supported by the real-time application.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ▪ DsDAQ ▪ HostService <p>If the real-time application supports the DsDAQ measurement service, the DsDAQBackground key specifies whether the DsDAQ background service is available in the real-time application.</p> <p>If the background service is available, set the DsDAQBackground to 1.</p> <div> Note To use the DsDAQBackground key, the [System] section must contain the DAQService=DsDAQ key. </div>

¹⁾ In brackets: parameter multiplicity

²⁾ Some dSPACE products currently do not evaluate the value of this key.

[<RTP_name>] section

An SDF file contains one or more [<RTP_name>] sections, each of which contains information on a processor in a single-processor or multiprocessor system.

The table below shows the possible keys of an [<RTP_name>] section:

Key ¹⁾	Description
Type (1)	<p>Specifies the type of the processor.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ▪ DS1006 (for a DS1006 Processor Board) ▪ DS1007 (for a DS1007 PPC Processor Board) ▪ DS1104 (for a DS1104 R&D Controller Board) ▪ DS1202 (for a MicroLabBox)

Key ¹⁾	Description
ServiceID (1) BoardName (0 ... 1)	<ul style="list-style-type: none"> ▪ DS1401 (for a MicroAutoBox II) <p>Specifies the application number which is used to identify the processors in a multiprocessor system. For a single-processor system, set the ServiceID value to 1.</p> <p>Specifies the name of the board.</p> <div> Tip If you do not know the board name, you can specify unknown as the BoardName value. If no BoardName key is specified, the unknown value is used as the default BoardName. </div>
File (0 ... 1)	<p>Specifies the name of the application file that can be loaded to and executed by the processor. The application file name can be specified with or without file extension. Possible values for the path to the application file:</p> <ul style="list-style-type: none"> ▪ <i>No path</i>: If no path is specified, the application file must be located in the same folder as the SDF file. ▪ <i>Relative path</i>: If a relative path is specified, it starts from the folder in which the SDF file is located. Example: <code>..\obj\ThrottleControl1.ppc</code> ▪ <i>Absolute path</i>: You can specify an absolute path. Example: <code>c:\dSPACE\Work\Applications\ThrottleControl1.ppc</code> <div> Note The File key in the [System] section and in the [<RTP_name>] sections are exclusive: If the [System] section contains a File key, the [<RTP_name>] sections must not contain File keys, and vice versa. </div>
Peripheral (0 ... *) State (0 ... 1)	<p>Specifies the type of I/O board connected to the current processor board. Example: Peripheral=DS2211</p> <p>Specifies whether the processor is ready to execute an application.</p> <div> Note Use this key only for the processors of a multiprocessor system. The key allows you to model dynamic multiprocessor system topologies. If no State key is specified, the Enabled value is used as the default State. </div> <p>Possible values:</p> <ul style="list-style-type: none"> ▪ Enabled: An application can be loaded to the processor. ▪ Disabled: When an application is loaded to the multiprocessor system, the selected processor is ignored, i.e., no application is loaded to it.
TraceFile (0 ... 1)	<p>Specifies the name of the TRC files of the application running on the processor.</p>

Key ¹⁾	Description
ID (0 ... 1)	Specifies the ID of the application running on the processor. The ID is updated after each build process and allows the experiment software to check whether the currently loaded application matches the binary application file.

¹⁾ In brackets: parameter multiplicity

[<Board_name>] section (only DS230x)

An SDF file can contain one or more [<Board_name>] sections, each of which contains information on a DS230x board.

The table below shows the possible keys of a [<Board_name>] section:

Key ¹⁾	Description
Type (1)	Specifies the type of the board.
ISRRegister (1)	Specifies the ISR register content of the DS230x.
OSRRegister (1)	Specifies the OSR register content of the DS230x.
IOError (1)	Must be set to 0x00.

¹⁾ In brackets: parameter multiplicity

[<Board_Channel>] section (only DS230x)

An SDF file can contain one or more [<Board_Channel>] sections, each of which contains information on a single channel of a DS230x board.

The section name is derived from the related BoardName value and the channel-specific suffix _A ... _F.

The table below shows the possible keys of a [<Board_Channel>] section:

Key ¹⁾	Description
BoardName (1)	Specifies the DS230x board to which the channel belongs.
File (1)	Specifies the object file of the channel.
IOModule (1)	Specifies an I/O module of the channel. Example: DAC_1

¹⁾ In brackets: parameter multiplicity

Related topics

Examples

[Example SDF Files.....](#) 137

Example SDF Files

DS1006-based multiprocessor system

The example SDF file below describes a multiprocessor system based on two DS1006 boards.

```
[System]
Version=2.2
Status=Start
SystemType=MultiProcessorSystem
DAQService=DsDAQ
DsDAQBackground=1
DAQService=HostService
RTP=MASTER
RTP=SLAVE

[MASTER]
Type=DS1006
ServiceId=1
BoardName=unknown
File=pi_1006_sl.x86
State=Enabled
ID=<Unique_ID for master>

[SLAVE]
Type=DS1006
ServiceId=2
BoardName=unknown
File=pt1_1006_sl.x86
State=Enabled
ID=<Unique_ID for slave>
```

Related topics

Basics

[SDF File Syntax: Sections and Keys.....](#) 133

Syntax of the TRC File

Introduction

The TRC file provides information on the variables of a real-time application that is required to connect variables to instruments in a ControlDesk layout, for example. It is an ASCII file that can either be generated automatically by RTI, or written manually.

Note

If you write a TRC file manually, you must adhere to the syntax of the TRC file. Then, you can easily switch from a simulation on the Simulink platform to an application running on a dSPACE real-time board.

TRC file syntax

To structure variables, for example, in the Variable Browser of ControlDesk, you can divide all model variables into hierarchical levels of subgroups. This feature is called *grouping*, see [Grouping](#) on page 139.

Refer to the following sections for information on the syntax elements of a TRC file:

- [Keywords](#) on page 144
- [Variable Names](#) on page 141
- [Variable and Group Properties](#) on page 154
- [Comments](#) on page 142

Example

For examples of TRC files, refer to

- [Example of Accessing Custom Variables in ControlDesk](#) on page 172
- [Example of a TRC File Generated by RTI](#) on page 175

Error file

If you write your own TRC file incorrectly, an Error file is generated when you download the corresponding application: see [Error File](#) on page 142. Use this file to correct your own TRC file.

Where to go from here

Information in this section

Principles of the TRC File	139
Keywords	144
Variable and Group Properties	154
Examples	172

Principles of the TRC File

Where to go from here	Information in this section
	Grouping..... 139
	Variable Names..... 141
	Comments..... 142
	Error File..... 142

Grouping

Defining groups

For large real-time applications with numerous variables, it is useful to arrange these variables into several groups. To define a group, enclose the corresponding variables in the keywords **group** and **endgroup**. Nesting **group** – **endgroup** statements allows you to create multilevel tree structures. An **endgroup** statement always belongs to the most recent **group** statement. Variables that are declared between these statements belong to this group and will be listed in the Variable List of the corresponding browser node.

Naming of groups

The keyword **group** must be followed by a name enclosed in quotation marks ("..."). If quotation marks are used in the string, they must appear twice. The name must be of the same format as described in [alias](#) on page 158. If two successive slashes occur in a name (//), they are transformed into a single one.

Example

```
group  "Model"
group  "Group-Name  ""A"""
```

Note

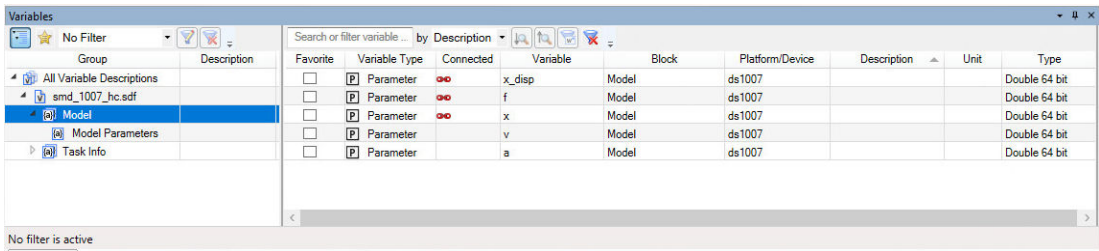
- In a TRC file, a **group** statement must always have a matching **endgroup** statement.
- Always insert an empty line between the closing brace and the **endgroup** statement.

Example

The following extract is taken from the `smd_xxxx_hc.trc` file found in the ControlDesk demo project, that is available as backup file in `<RCP_HIL_InstallationPath>\Demos\dsxxxx\GettingStarted\HandCode`.

```
group "Model"
  x_disp  flt
  f       flt
  x       flt
  v       flt
  a       flt
  group "Model Parameters"
    d      flt
    c      flt
    m      flt
  endgroup
endgroup
```

In ControlDesk's Variables controlbar, these variables will look like this:



Note

At the end of the TRC file an empty line has to be inserted to avoid an error message caused by the TRC file parser.

Appearance of groups in ControlDesk's Variables controlbar

In ControlDesk's Variables controlbar, for example, a group will appear as a node (unless the node has the flag `HIDDEN`, see [flags](#) on page 161).

Related topics

Basics

Keywords.....	144
Variable and Group Properties.....	154
Variable Names.....	141

Examples

Example of a TRC File Generated by RTI.....	175
---	-----

Example of Accessing Custom Variables in ControlDesk.....	172
References	
Comments.....	142

Variable Names

Variable names

The name of the variable can be a scalar or an array and is limited to a maximum of 4096 characters. The name (or its alias) will appear in ControlDesk's Variables controlbar, for example. The name of the variable must be identical to the name of the corresponding global variable of the real-time program. Variables declared as **static** cannot be accessed, for example by ControlDesk, unless their address is explicitly given in the TRC file because such variables do not appear in the MAP file. If a variable is not defined in `<model>.c`, the line in the TRC file is accepted only if the absolute address is given.

Note

You must assign a datatype to each variable.

Example

```
X[0]
{
  type: flt
}
```

Related topics

Basics	
Grouping.....	139
Keywords.....	144
Variable and Group Properties.....	154
Examples	
Example of a TRC File Generated by RTI.....	175
Example of Accessing Custom Variables in ControlDesk.....	172
References	
alias.....	158

Comments

Syntax of a comment

TRC files may contain comments. Initial double minus characters `--` declare a line in the TRC file as a comment.

Example

```
-- this is a comment
```

Note

The length of a comment is limited to 4096 characters.

Related topics

Basics

Grouping.....	139
Keywords.....	144
Variable and Group Properties.....	154
Variable Names.....	141

Examples

Example of a TRC File Generated by RTI.....	175
Example of Accessing Custom Variables in ControlDesk.....	172

Error File

Error messages

The experiment software parses the TRC file together with the Linker MAP file. If you write your own TRC file incorrectly, an error message will be displayed, for example, when you download the corresponding application.

Error messages are listed in the `<model>.err` or in the `<model>_user.err` file. Some of the possible error messages are:

Error	Description
Syntax error	
identifier expected	A line must begin with a name or a keyword; a group instruction must be followed by a name.
type, [or expected	An identifier can only be followed by the given symbols.
type expected	Array declarations must be followed by a type.
number expected	A type can only be followed by an address or a comment.
illegal numeric format	Illegal syntax used for a numeric value (decimal or hex with a leading 0x or a trailing h).
float number expected	A floating-point number is expected, either in absolute or exponential format.
extra characters	Superfluous characters given; maybe a comment without -- .
] expected	A right bracket is expected.
string exceeds end of line (" expected)	The terminating quotes of a string could not be found; multi-line strings are not allowed.
endgroup missing	Each group statement requires a matching endgroup statement.
illegal endgroup	There is no matching group statement for the endgroup statement.
groupname must not be empty	The matching group statement must be followed by a group name in " " , or the description block must contain an alias statement.
filename is empty	The keyword _application must be followed by a string constant that contains a file name.
keyword _application must not occur multiple	The keyword _application may occur only once in the TRC file.
illegal data size	The data size can only be 32-bit or 64-bit (TI floating-point data format can only be 32-bit).
illegal data format	The data format can only be TI or IEEE .
illegal index or array declaration	An array must be defined in one of the following formats: 1. [2] 2. [4.6]
unexpected symbol	A symbol does not fit the TRC file structure.
illegal use of keyword	A keyword was not expected to be on its position.
string constant expected	The keyword _application must be followed by a string constant.
Semantic error	
invalid index range	The first index of an array declaration is higher than the last one.
group already defined	A group name must not occur multiple times in the same subgroup.

Switching between short and verbose error file formats

Error files are by default short. They contain only the ordinal numbers of the erroneous lines which are followed by the error messages. Error files can also be verbose. A verbose error file contains the complete context of the TRC file. The error messages are inserted behind the erroneous TRC file lines.

To switch between short and verbose error files, ControlDesk provides the Trace file parser error output option. Refer to [Variables Page \(ControlDesk Variable Management !\[\]\(d263118e0bfd47dc6bc704167d936b83_img.jpg\)](#)).

Keywords

Introduction

In TRC files different keywords are used to store information on the TRC file and structure the contents.

Rules for keywords

Each keyword is optional and is followed by a string containing the corresponding value. If a keyword definition appears more than once in a TRC file, the latest definition will be applied.

Note

- A keyword must not be used as variable name in the real-time model.
- All of the keywords are reserved words. You cannot use them for global variables, such as a Simulink.Parameter with *ExportedGlobal* as the storage class.
- In a structured data type, the field names can be set to keywords, such as *value* or *default*.
- Except for **group** and **endgroup**, all keywords are case sensitive.

Where to go from here

Information in this section

_author.....	145
To indicate the name of the author creating the model.	
_description.....	146
To give additional information on the model.	
_floating_point_type().....	146
To set a new default size for floating-point variables (flt, float).	
_gendate.....	147
To indicate the date and time when the TRC file was created.	
_genname.....	147
To indicate the name of the tool generating the TRC file.	
_genversion.....	147
To indicate the version of the tool generating the TRC file.	

_integer_type()	148
To set a new default size for integer variables (int) and unsigned integer variables (uint).	
_model	148
To indicate the name of the model.	
endgroup	149
To indicate the end of a subgroup.	
endstruct	149
To indicate the end of a structure data type.	
enum	150
To define enumeration values.	
group	150
To define a group.	
sampling_period[host_service_index]	151
To specify a host service for data capturing.	
struct	153
To define a structure data type.	
typedef	154
To define a new customized datatype.	

[_author](#)

Syntax	<code>_author "name"</code>
Purpose	To indicate the name of the author creating the model.
Description	This keyword is used to indicate the name of the model's author. The keyword is case sensitive. The entire name must be enclosed in quotation marks ("...").
Example	<code>_author "RTI1104 7.5 (02-November-2015)"</code>

`_description`

Syntax	<code>_description "model description"</code>
Purpose	To give additional information on the model.
Description	This keyword can be used to describe the model more precisely or to add further information. The keyword is case sensitive. The entire value must be enclosed in quotation marks ("...").
Example	<code>_description "Add a clear description if possible"</code>

`_floating_point_type()`

Syntax	<code>_floating_point_type(size, format)</code>
Purpose	To set a new default size for floating-point variables (flt, float).
Description	<p>By default all types are evaluated as 32-bit variables, and floating-point values are supposed to be defined in the Texas Instruments format.</p> <p>The default size of floating-point variables (<code>flt</code> or <code>float</code>, <code>flt*</code> or <code>float*</code>) can be changed using the keyword <code>_floating_point_type</code>. This keyword expects two parameters, the size (32-bit or 64-bit) and the internal format of the floating-point value, TI or IEEE.</p> <p>The scope of <code>_floating_point_type</code> ranges from its current position within the TRC file until the end of file or until another <code>_floating_point_type</code> occurs.</p> <p>The keyword is case sensitive.</p>
Example	<code>_floating_point_type(64, IEEE)</code>

Note

The combination of 64-bit with the TI format for floating-point values is not supported and leads to an error. Variables using data types that are not allowed are removed while the MAP file is parsed.

_gendate

Syntax	<code>_gendate "date and time"</code>
Purpose	To indicate the date and time when the TRC file was created.
Description	This keyword is used to indicate the date and time when the TRC file was created. The keyword is case sensitive. The entire value must be enclosed in quotation marks ("...").
Example	<code>_gendate "05/04/2015 10:49:39"</code>

_genname

Syntax	<code>_genname "name"</code>
Purpose	To indicate the name of the tool generating the TRC file.
Description	This information is useful when the format of any of the blocks used in the real-time application has to be ascertained. The keyword is case sensitive. The entire value must be enclosed in quotation marks ("...").
Example	<code>_genname "RTI"</code>

_genversion

Syntax	<code>_genversion "number"</code>
Purpose	To indicate the version of the tool generating the TRC file.

Description	If the TRC file is generated automatically, this keyword indicates the version number of the generating tool. The keyword is case sensitive. The entire value must be enclosed in quotation marks ("...").
Example	<code>_genversion "1.2"</code>

`_integer_type()`

Syntax	<code>_integer_type(size)</code>
Purpose	To set a new default size for integer variables (int) and unsigned integer variables (uint).
Description	<p>The keyword <code>_integer_type</code> changes the default size (32-bit) of all variables defined as <code>int</code>, <code>int*</code> or <code>uint</code>, <code>uint*</code>. The size can be set to 8-bit, 16-bit, 32-bit or 64-bit. This value follows the keyword enclosed in parentheses.</p> <p>The scope of <code>_integer_type</code> ranges from its current position within the TRC file until the end of file or until another <code>_integer_type</code> occurs.</p> <p>The keyword is case sensitive.</p>
Example	<code>_integer_type(64)</code>

`_model`

Syntax	<code>_model "name"</code>
Purpose	To indicate the name of the model.
Description	This keyword is used to indicate the name of the model and is case sensitive. The entire value must be enclosed in quotation marks ("...").
Example	<code>_model "smd_1104_s1"</code>

endgroup

Syntax	<code>endgroup</code>
Purpose	To indicate the end of a subgroup.
Description	<p>The keyword <code>endgroup</code> is used to close a group. Refer to Grouping on page 139.</p> <p>This keyword is not case sensitive.</p>
Example	<pre>... endgroup</pre>
Related topics	<p>References</p> <p>group..... 150</p>

endstruct

Syntax	<code>endstruct</code>
Purpose	To indicate the end of a structure data type.
Description	<p>The keyword <code>endstruct</code> is used to close a structure definition in a TRC file.</p> <p>This keyword is case sensitive.</p>
Example	<pre>... endstruct</pre>
Related topics	<p>References</p> <p>struct..... 153</p>

enum

Syntax

```
enum <enumName>
{
    type: int32
    enums
    {
        <enumNumber>: <enumString>
        ...
    }
}
```

Purpose

To define enumeration values.

Description

Enums specified in MATLAB/Simulink are generated into the Variable Description File and can be used for experimentation.

Example

The definition of the enumeration values for an LED state:

```
enum LEDState
{
    type:int(32)
    enums
    {
        0: "Off"
        1: "Blinking"
        2: "On"
    }
}
```

The definition of the `myLED` variable using the enum data type:

```
myLED
{
    type: enum LEDState
    alias: "myLED"
    flags: PARAM
}
```

The definition of a type definition using an enum:

```
typedef LEDStateArray enum LEDState[2]
```

group

Syntax

```
group    "name"
```

Purpose	To define a group.
Description	<p>The keyword group is used as an initialization command of a group in a TRC file. For example, in ControlDesk's Variables controlbar, this group will appear as a node. For further information please refer to Grouping on page 139.</p> <p>This keyword is not case sensitive.</p>
Example	<pre>group "group_name" { desc: ...</pre>
Related topics	<p>References</p> <p>endgroup..... 149</p>

sampling_period[host_service_index]

Syntax	<pre>sampling_period[host_service_index] { value: alias: }</pre>
Purpose	To specify a host service for data capturing.
Description	<p>A real-time application can have up to 31 host services for data capturing (host service 1 ... 31; 29 ... 31: reserved). For each host service one sampling_period[host_service_index] entry must be located in the TRC file. The host_service_index entry represents the host service number minus 1. The alias specifies the name of the Data Capture that is, for example, displayed in ControlDesk. The value can be:</p> <ul style="list-style-type: none"> ▪ The sample time of the host service (the host service is located in a timer-driven task) ▪ 0.0 (the host service is located in an event-driven task) <p>The value must be given as a floating-point number. In RTI data captures are specified by the Data Capture block. The host_service_index corresponds to</p>

the host service number of the block minus 1. The “service name” of the data capture block properties is written to the **alias** field, the sample time to the **value** field.

If you have implemented the `host_service(3,0)` the following entry for the sampling rate must be specified: `sampling period[2]`.

Note

- The keyword `sampling_period[host_service_index]` is case sensitive.
- Each index is assigned to a separate host service. The corresponding host service number is always one integer greater than the index of the `sampling_period`. Indices used twice are not permitted and will cause an error.

Multiprocessor systems

In multiprocessor systems, such as a DS1006 multiprocessor system, the alias of a host service has a more complex meaning:

In multiprocessor systems, host services with equal names on different processors belong together and are displayed as one multiprocessor host service, for example, in ControlDesk.

For host services in timer-driven tasks it is sufficient that the services exist on more than one processor to be offered as multiprocessor host service. When the service is missing on one processor and the user nevertheless wants to capture data with it, the host service in the fastest timer task is used instead.

Host services in event-driven tasks that are not common to all processors are not offered as multiprocessor host service.

Example

In a TRC file generated by RTI, the entry for the sampling period for a host service defined via RTI’s Data Capture block may look like this:

```
sampling_period[0]
{
  value:      0.0001
  alias:      "HostService"
}
```

Note

The keywords `increment` and `unit` that may also appear in TRC files generated by RTI are not used. They are provided for future use.

struct

Syntax	<code>struct <structName></code>
Purpose	To define a structured data type.
Description	<p>The keyword <code>struct</code> is used as an initialization command of a structure in a TRC file. It is followed by the type definition and must be closed with the <code>endstruct</code> keyword.</p> <p>This keyword is case sensitive.</p>

Example	<pre>struct MyStruct { desc: ... array-incr: -1 } X { type: int ofs: -1 } CustomNameForY { alias: "Y" type: int ofs: -1 } endstruct</pre>
---------	---

Note

If you manually create a User TRC file, you must set the values for `ofs` and `array-incr` to -1. During the TRC file generation, these values are replaced according to the variable addresses.

Related topics

References

endstruct.....	149
--------------------------------	-----

typedef

Syntax

```
typedef typename type[size]
```

Purpose

To define a new customized datatype.

Description

The keyword is followed by the new datatype name, the datatype being used and, enclosed in brackets, the number of elements being created. The keyword is case sensitive. The following example creates a 5 x 5 matrix.

Example

```
typedef Seq1D flt[5][5]
```

Note

Variables using datatypes that are not allowed are removed while the MAP file is parsed. For example, on DSP base hardware, the data types Int8 and Int16 are not supported and therefore not allowed. Defining C code structures by means of this keyword is not possible.

Variable and Group Properties

Introduction

You can assign properties and attributes to variables or groups of variables.

Variable and group properties

For information on the naming of variables and groups, refer to [Variable Names](#) on page 141.

Note

- A property must not be used as variable name in the real-time model.
- You must assign a datatype to each variable. See [type \(Data Type, Data Format and Type Definition\)](#) on page 167.
- Except for the datatype, all other properties are optional.
- Enclose the properties belonging to a variable or a group of variables in braces ({...}).

In a TRC file each variable is declared in a separate line that is followed by a block containing all properties such as the **type**, the (physical) **unit** or the **alias** of the block.

Example

```
X[0]
{
  type:    flt(32, IEEE)
  alias:   "rpm"
  addr:    0x805000
  unit:    "s"
  flags:   READONLY
}
```

Several property blocks

For each signal, several property blocks can be defined. Make sure that the **alias** names used for these blocks are unambiguous. Defining several property blocks is useful whenever a signal should be observed with different data types.

Example The following example shows how the signal **myUnion** can be made accessible both as an integer value and as a float value for experiment software.

```
myUnion
{
  alias: "Output as int"
  type:  int
  ...
}

myUnion
{
  alias: "Output as float"
  type:  float
  ...
}
```

Where to go from here

Information in this section

addr.....	156
To specify the memory address of a variable that is not accessible via the MAP file.	
alias.....	158
To define a more intuitive name for a variable.	
array-incr.....	159
To specify the memory size in bytes of the related structure.	
bitmask.....	159
To mask bits of the signal value.	
block.....	160
To describe the blocktype of a Simulink block.	
default.....	160
To specify the default value for a signal.	

desc	161
To describe a signal or a group.	
flags	161
To describe special properties of a signal.	
increment	161
To specify the unit increment for a task.	
offs	162
To specify the offset of a field definition in a structure in bytes.	
origin	163
To specify the entire path of signals, parameters and blocks.	
range	163
To define the valid range for the signal value variation.	
refelem	163
To specify a field in a structure or an element in an array that is used as reference.	
refgroup	165
To specify the group name of a variable reference.	
refvar	166
To specify the variable name of a reference.	
scale	166
To convert the signal value.	
scaleback	167
To reverse the scale function.	
type (Data Type, Data Format and Type Definition)	167
To specify the type, format and size of a variable and to define look-up tables.	
unit	171
To set the physical unit for a signal value.	
value	171
To specify the current value for a signal.	

addr

Syntax

```
addr:    address
```

Purpose

To specify the memory address of a variable that is not accessible via the MAP file.

Description

If a variable is allocated to an absolute address outside the scope of the linker (for example, in dual-port memory) this variable does not appear in the MAP file. However, it can be made accessible, for example, for ControlDesk, if the base address of this variable is known. Therefore, this address in the real-time processor's memory has to be entered.

Note

The `addr` property is not available for references.

Addresses may be declared as:

- Absolute addresses in hexadecimal notation starting with `0x`
- Absolute addresses in hexadecimal notation with at least one leading digit and a trailing `h` character
- Absolute addresses in decimal notation

For example, the address can be written as follows:

Variable Name	Data Type	Address Notation
<code>X[0]</code>	<code>flt</code>	<code>0x805000</code>
<code>X[0]</code>	<code>flt</code>	<code>805000h</code>
<code>X[0]</code>	<code>flt</code>	<code>8409088</code>

Note

Although the given variable is declared with an address it is not a pointer variable. Its type remains `float`. This is in contrast to the `float *` type, which means that a pointer to a float is located at the address.

Arbitrary array subranges can be referenced in TRC files as shown below. Each array element is treated, for example, by ControlDesk, as a separate variable. Array indices in TRC files as well as in C programs always start at zero. The variable `rtB[3]` is equal to the fourth element of that array.

Although only the base address has to be given, the offset of the variable's address will be calculated automatically. For example:

```
rtB[3]
{
  type: flt
  alias: "Element with index 3"
  addr: 0x00000010
}
```

The following type of declaration must be used to access arrays that were allocated during run time by function calls to `malloc()` or `calloc()`. For example:

```
x_dot[0..3]
{
  type: float *
  alias: "Array access"
}
```

Note

It is not possible to declare an array of pointers in the TRC file. An array with a pointer datatype (`float *`, `int *` or `uint *`) means that there is a pointer variable in the C program pointing to an array of `float`, `int` or `uint` values. The subsequent array indices in the TRC file are used to access the respective array elements.

alias

Syntax

```
alias: "customized variable name"
```

Purpose

To define a more intuitive name for a variable.

Description

This property can be used to set the `alias` name of a variable (array element or scalar variable) that has already been defined in order to provide the variable with a more intuitive name. Alias names can have two formats: either a standard C variable name or a string. The name of a variable is formed by an underline or a letter ('_', 'a-z' or 'A-Z') as the first character, followed by a sequence of alphanumeric characters ('_', 'a-z', 'A-Z', or '0-9'). A string begins and ends with quotation marks (").

Example

```
X[0]
{
  alias: "rpm"
  addr: 0x805000
  ...
}
```

If two successive slashes occur in a name (`//`), they are transformed into a single one. If quotation marks are used in the string, they must appear twice.

Example

```
"This is block ""a""".
```

Note

- The **alias** name must be defined within the braces of the corresponding variable. The variable cannot be renamed in another property block.
- Alias names are limited to a maximum of 128 characters.

array-incr

Syntax

```
array-incr:  number
```

Purpose

To specify the memory size in bytes of the related structure.

Description

In a user TRC file, the array increment is to be specified by -1. During the build, the value is replaced according to the platform-specific variable addresses. This property is mandatory and valid only for structure definitions.

Example

```
struct MyStruct
{
  desc: ...
  array-incr: -1
}
X
{
  type: int
  ofs: -1
}
CustomNameForY
{
  alias: "Y"
  type: int
  ofs: -1
}
endstruct
```

bitmask

Syntax

- **bitmask:** hexnumber
- **bitmask:** startbit : endbit

Purpose	To mask bits of the signal value.
Description	This property provides bit access to the signal value. The least significant bit is defined as bit number 0.
Example	<ul style="list-style-type: none">▪ <code>bitmask: 0xF012</code>▪ <code>bitmask: 8:11</code>

block

Syntax	<code>block: "blocktype"</code>
Purpose	To describe the blocktype of a Simulink block.
Description	This property can only be assigned to nodes. For example, it stores the type of a Simulink block that is represented by this node.
Example	<code>block: "Gain"</code>

default

Syntax	<code>default: value</code>
Purpose	To specify the default value for a signal.
Description	This property specifies the default value for a signal, which can automatically be displayed, for example, in a ControlDesk instrument. The permissible values depend on the type of the signal. String values must be enclosed in quotation marks ("").
Example	<code>default: 75.2</code>

desc

Syntax

```
desc:  "text"
```

Purpose

To describe a signal or a group.

Description

This field contains text describing a signal or a group.

Example

```
desc:  "Current_Speed"
```

flags

Syntax

```
flags:  flag [ | flag ]
```

Purpose

To describe special properties of a signal.

Description

This field contains flags describing special properties of the signal. Flags can be combined and also be set to variables or blocks.

Example

```
flags:  HIDDEN | PARAM
```

The following table lists all available flags alphabetically:

Flag	Purpose
HIDDEN	To hide a node in ControlDesk's Variables controlbar.
OUTPUT	To mark RTI block outputs.
PARAM	To mark a variable as a parameter.
READONLY	To make a variable read-only. The variable cannot be written.
DEPRECATED	To mark an item as deprecated.

increment

Syntax

```
increment:  time_in_seconds
```

Purpose	To specify the unit increment for a task.
----------------	---

Description	<p>RTI uses this property to provide details on the sampling period of a task in a Simulink model:</p> <p>If the Simulink model contains no Data Capture blocks, the <code>increment</code> value corresponds to the sampling period of the model.</p>
--------------------	--

Example	<code>increment: 0.01</code>
----------------	------------------------------

offs

Syntax	<code>offs: no_of_bytes</code>
---------------	--------------------------------

Purpose	To specify the offset of a field definition in a structure in bytes.
----------------	--

Description	In a user TRC file, the offset is to be specified by -1. During the build, the value is replaced according to the platform-specific variable addresses. This property is mandatory and valid only for struct elements.
--------------------	--

Example	<pre> struct MyStruct { desc: ... array-incr: -1 } X { type: int offs: -1 } CustomNameForY { alias: "Y" type: int offs: -1 } endstruct </pre>
----------------	---

origin

Syntax	<code>origin: "model/subsystem/.../block/signal"</code>
Purpose	To specify the entire path of signals, parameters and blocks.
Description	In TRC files generated by RTI, this property is used for signal labels to indicate the path of the corresponding signal in the Simulink model.
Example	<pre>origin: "smd_1007_sl/Integrator 1/Out1" flags: LABEL READONLY</pre>

range

Syntax	<code>range: <min; max></code>
Purpose	To define the valid range for the signal value variation.
Description	Integer, floating point and exponential numbers are possible for min and max. Use the keyword <code>inf</code> to define an infinite limiting value.
Example	<ul style="list-style-type: none"> ▪ <code>range: <-5; 5></code> ▪ <code>range: <-5; inf></code>

refelem

Syntax	<code>refelem: "elementname"</code>
Purpose	To specify a field in a structure or an element in an array that is used as reference.

Description

If the **refvar** property references a structure or an array, the **refelem** property specifies the concrete element to be used as a reference.

The element name is specified as a string.

A nested structure is described with dots as path delimiters. The path must then also start with a dot.

For an array element, only the index of the element is given in square brackets. The name of the array is specified in the related **refvar** property.

When using arrays in structures or arrays of structures, the notations for referencing a structure field and for referencing an array element can be combined, e.g., **refelem**: `".myStructField[2].myArray[5]"`.

Example

Example of a structure element

```
struct MyStruct
{
  desc: ...
  array-incr: -1
}
X
{
  type: int
  offs: -1
}
CustomNameForY
{
  alias: "Y"
  type: int
  offs: -1
}
endstruct
```

```
pointStructVar
{
  type: struct pointStruct
  alias: "MyPointStructVar"
  flags: PARAM
}

ref2FieldVar
{
  alias: "MyFieldVar"
  refgroup: "."
  refvar: "MyPointStructVar"
  refelem: ".X"
}
```

Example of an array

```
typedef IntArray int[5]

intArrayVar
{
  type: IntArray
  alias: "MyIntArray"
  flags: PARAM
}

ref2IntArrayElem
{
  alias: "MyArrayElem"
  refgroup: "."
  refvar: "MyIntArray"
  refelem: "[2]"
}
```

refgroup

Syntax	<code>refgroup: "groupname"</code>
Purpose	To specify the group name of a variable reference.
Description	<p>The <code>refgroup</code> property specifies in which group the referenced variable is declared. The group name is specified as a string and contains either an absolute or a relative path to the group.</p> <p>The following elements are supported in the path definition:</p> <ul style="list-style-type: none"> ▪ <code>"/</code> as the path delimiter ▪ <code>".."</code> to specify the parent element ▪ <code>"."</code> to specify the current element <p>Absolute paths have to begin with a slash (<code>/</code>), relative paths can begin with the name of a subgroup, with a single dot (<code>.</code>), or with two dots (<code>..</code>).</p> <p>The <code>refgroup</code> property is mandatory for a reference element. The definition of the referenced group can be placed before or after the definition of the referenced variable in the variable description file.</p>
Example	<p>Example for an absolute path:</p> <pre>refgroup: "/Tunable Parameters"</pre>

Example for a relative path:

```
refgroup: "../MySubGroup/MyNestedSubGroup"
```

refvar

Syntax

```
refvar: "variablename"
```

Purpose

To specify the variable name of a reference.

Description

The **refvar** property is used within a **reference** element and requires at least a related **refgroup** property. The variable name is specified as a string. If an **alias** is specified, the alias name is used, otherwise the name of the referenced variable is used.

Example

Example of specifying the **refvar** property with an alias defined for the variable.

```
typedef IntArray int[5]

intArrayVar
{
    type: IntArray
    alias: "MyIntArray"
    flags: PARAM
}

ref2IntArrayElem
{
    refgroup: "."
    refvar: "MyIntArray"
    refelem: "[2]"
}
```

scale

Syntax

```
scale: [numerator polynomial] / [denominator polynomial]
```

Purpose

To convert the signal value.

Description	<p>When you read the signal from a data source, the signal value is converted by using the scale function. The value conversion is an option and not performed automatically.</p> <p>The denominator polynomial is optional. Possible coefficients are integer, floating point and exponential numbers.</p>
Example	<ul style="list-style-type: none"> ▪ <code>scale: [2 0 3] / [2 4]</code> ▪ <code>scale: [2, 0, 3] / [2, 4]</code> ▪ <code>scale: [2, 1.345, 2^-11]</code>

scaleback

Syntax	<code>scaleback: [numerator polynomial] / [denominator polynomial]</code>
Purpose	To reverse the scale function.
Description	<p>When you write the value to a data source, the value is converted to the signal value by using the scaleback function. The value conversion is an option and not performed automatically.</p> <p>The denominator polynomial is optional. Possible coefficients are integer, floating point and exponential numbers.</p>
Example	<ul style="list-style-type: none"> ▪ <code>scaleback: [2 4] / [2 0 3]</code> ▪ <code>scaleback: [2, 4] / [2, 0, 3]</code> ▪ <code>scaleback: [2, 1.345, 2^-11]</code>

type (Data Type, Data Format and Type Definition)

Syntax	<code>type: type (size,format)</code>
Purpose	To specify the type, format and size of a variable and to define look-up tables.

Description

- The size has to be set according to the real-time hardware. The following table displays the permissible data types and sizes:

Data Type	Description
<code>int (8)</code>	8-bit integer value
<code>int (8) *</code>	pointer to an 8-bit integer value
<code>int (16)</code>	16-bit integer value
<code>int (16) *</code>	pointer to a 16-bit integer value
<code>int (32)</code>	32-bit integer value
<code>int (32) *</code>	pointer to a 32-bit integer value
<code>int (64)</code>	64-bit integer value
<code>int (64) *</code>	pointer to a 64-bit integer value
<code>uint (8)</code>	8-bit unsigned integer value
<code>uint (8) *</code>	pointer to an 8-bit unsigned integer value
<code>uint (16)</code>	16-bit unsigned integer value
<code>uint (16) *</code>	pointer to a 16-bit unsigned integer value
<code>uint (32)</code>	32-bit unsigned integer value
<code>uint (32) *</code>	pointer to a 32-bit unsigned integer value
<code>uint (64)</code>	64-bit unsigned integer value
<code>uint (64) *</code>	pointer to a 64-bit unsigned integer value
<code>flt (32, IEEE)</code>	32-bit floating-point value
<code>flt (32, IEEE) *</code>	pointer to a 32-bit floating-point value
<code>flt (64, IEEE)</code>	64-bit floating-point value
<code>flt (64, IEEE) *</code>	pointer to a 64-bit floating-point value

- The format for floating-point values can only be IEEE standard. If you specify a variable of integer type, you do not need to define the format.
- You can additionally specify a variable of array, enumeration or struct type:
 - [Arrays](#) on page 168
 - [Enumerations](#) on page 169
 - [Structs](#) on page 170

Note

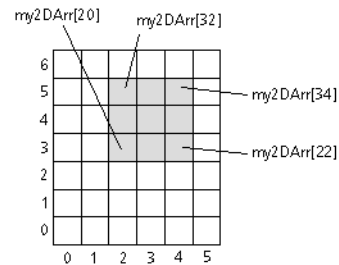
The `type` property used in structure elements does not support pointer types.

Arrays**Description**

- If you defined a 6x7 matrix, you can refer to specific elements of this two-dimensional array, for example:
`my2DArr[2..4][3..5] float (32, IEEE)`

This line refers to the following nine elements of `my2DArr`: `my2DArr[20]` ... `my2DArr[22]`, `my2DArr[26]` ... `my2DArr[28]` and `my2DArr[32]` ... `my2DArr[34]`

The following illustration shows the 6x7 matrix `my2DArr`. The selected matrix elements `my2DArr[2..4][3..5]` are highlighted.



Note

The **alias** name(s) are also added with index information in order to be unambiguous. Always start with zero when you count the elements of an array.

- You can also create an n-dimensional look-up table. Insert **lookup** between the **typename** and the **type** within a **typedef** statement.

Syntax `typedef typename lookup type`

Example `typedef Lookup2D lookup flt[6][4]`
`MyLookupTable Lookup2D`

For information on how to define new datatypes, refer to [typedef](#) on page 154.

Enumerations

Description Enums specified in MATLAB/Simulink are generated into the Variable Description File and can be used for experimentation.

Syntax

```
enum <enumName>
{
  type: <Integer-DataType>
  enums
  {
    <enumNumber>: <enumString>
    ...
  }
}
```

Example The definition of the enumeration values for an LED state:

```
enum LEDState
{
    type:int(32)
    enums
    {
        0: "Off"
        1: "Blinking"
        2: "On"
    }
}
```

The definition of the `LEDState` variable using the enum data type:

```
LEDState
{
    type: enum LEDState
    alias: "LEDState"
    value: 2
    unit: "-"
```

The definition of a type definition using an enum:

```
typedef LEDStateArray enum LEDState[2]
```

Structs

Description Structs specified in MATLAB/Simulink are generated into the Variable Description File and can be used for experimentation.

Syntax

```
struct <structname>

{
    desc: <String>
    array-incr: <Integer-DataType>
}

    <StructElement>
    {
        type: <String-DataType>
        offs: <Integer-DataType>
    }
    ...
endstruct
```

Example The definition of the struct elements:

```
struct MyStruct
{
  array-incr: -1
}
structField0
{
  alias: "element1"
  type: flt(64,IEEE)
  offs: -1
  unit: "mph"
  range; < 0.0 ; 225.0 >
  desc: "SPeed of the vehicle."
}
structField1
{
  alias: "element2"
  type: int(8)
  offs: -1
}
endstruct
```

Example of a type definition using a struct:


```
typedef PointStructArrayType struct MyStruct[4]
```

unit

Syntax	<code>unit: "physical_unit"</code>
Purpose	To set the physical unit for a signal value.
Description	This property gives information about the physical unit of the signal value. This text can automatically be displayed, for example, in the caption of an instrument.
Example	<code>unit: "mph"</code>

value

Syntax	<code>value: value</code>
---------------	---------------------------

Purpose	To specify the initial value of a parameter.
Description	In ControlDesk, this property is mainly used by the data set management. The specified value is used when initial data sets are generated. For details on data sets in ControlDesk, refer to Data Sets and their Relation to Memory Pages (ControlDesk Calibration and Data Set Management ).
Example	<code>value: 75.2</code>

Examples

Where to go from here	Information in this section
	Example of Accessing Custom Variables in ControlDesk..... 172 Example of a TRC File Generated by RTI..... 175

Example of Accessing Custom Variables in ControlDesk

Introduction	To make different variables of user-written code, User-Code or an S-function accessible for ControlDesk, you have to prepare the C code and provide a user-specific TRC file.
---------------------	---

Preparing the C code	In the following code segment of an S-function, the variables a ... f are defined as volatile to make them accessible for ControlDesk. The variables b and e are pointers, the variable c is a one-dimensional array, and the variable f is a two-dimensional array.
-----------------------------	---

```
/* global variables */
volatile real_T a = 1.23;
volatile real_T* b = &a;
volatile real_T c[3] = {4.4, 5.5, 6.6};
volatile int_T d = 1;
volatile int_T* e = &d;
volatile int_T f[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

The variables to be accessed have to be specified in the user-specific TRC file.

Specifying custom variables

The variables `a ... f` are specified in the user-specific TRC file. The variables `c[0]`, `c[1]` and `c[2]` are defined to access each element of `c` individually. The variable `f[1..1][1..2]` is defined to access the elements named `f[3]` and `f[5]` of `f`.

```
group "Variables in "sfunc.c""

  group "Scalars"
    a
    {
      type: flt(64,IEEE)
      alias: "a real"
      flags: READONLY
    }
    b
    {
      type: flt(64,IEEE)*
      alias: "b pointer to a"
      flags: READONLY
    }
    d
    {
      type: int(32)
      alias: "d integer"
      flags: PARAM
    }
    e
    {
      type: int(32)*
      alias: "e pointer to d"
      flags: PARAM
    }
  }
endgroup
```

```

group "Vectors"
c[0..2]
{
    type: flt(64,IEEE)
    alias: "c all"
    flags: READONLY
}
c[0]
{
    type: flt(64,IEEE)
    alias: "c first element"
    flags: READONLY
}
c[1]
{
    type: flt(64,IEEE)
    alias: "c second element"
    flags: READONLY
}
c[2]
{
    type: flt(64,IEEE)
    alias: "c third element"
    flags: READONLY
}
f[0..1][0..2]
{
    type: int(32)
    alias: "f all"
    flags: READONLY
}
f[1..1][1..2]
{
    type: int(32)
    alias: "f selection"
    flags: READONLY
}
endgroup
endgroup

```

User variables in ControlDesk's Variables controlbar

In ControlDesk's Variables controlbar, the tree window provides the custom variables in the User Variables - Variables in "sfunc.c" node.

Related topics

Basics

[Inserting Custom C/C++ Code \(RTI and RTI-MP Implementation Guide !\[\]\(feabb98897b440bc8695a03336a6e2df_img.jpg\)\)](#)

HowTos

[How to Make Custom Variables Accessible \(RTI and RTI-MP Implementation Guide !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)\)](#)

Examples

[Example of a TRC File Generated by RTI..... 175](#)

References

[type \(Data Type, Data Format and Type Definition\)..... 167](#)

Example of a TRC File Generated by RTI

Example

The following example is an extract from the TRC file `smd_1007_sl.trc` generated by RTI when you build the `smd_1007_sl` demo model. You can find the demo model in `<RCP_HIL_InstallationPath>\Demos\DS1007\GettingStarted\Simulink`.

Note

This example does not contain the generated variable addresses.

```

-- *****
-- Trace file: smd_1007_sl.trc
-- RTI1007 7.14 (02-May-2020)
--
-- Copyright 2020, dSPACE GmbH. All rights reserved.
-- Used options:
--   TRCGenerateVirtualBlocks      = 1
--   TRCGenerateLabels             = 1
--   TRCGenerateStates             = 0
--   TRCGenerateDerivatives        = 0
--   TRCGenerateParamValues        = 0
--   TRCGenerateApplicationKeyword = 0
--   TRCOnlyGlobals                = 0
--   TRCIsOmitVdOn                 = 0
-- Trace file format: 3
-- *****
-- ***** Keywords*****
-- _application "smd_1007_sl.map"
-- _genname     "RTI"
-- _genversion  "7.14"
-- _gendate     "09/07/2018 09:49:03"
-- _description ""
-- _author      "RTI1007 7.14 (02-May-2020)"
-- _model       "smd_1007_sl"
-- Default data type formats
-- _floating_point_type(64,IEEE)
-- _integer_type(32)
-- No Data Capture blocks within model: use base sample time as sampling period
sampling_period[0]
{
  value:      0.0001
  alias:      "HostService"
  increment:  0.0001
  unit:       "s"
}
-- ***** Simulation control variables*****
finalTime
{
  type: flt(64,IEEE)*
  alias: "finalTime"
  desc: "Simulation stop time. When reached, simState switches to STOP."
  flags: READONLY
  unit: "s"
}
currentTime
{
  type: flt(64,IEEE)*
  alias: "currentTime"
  desc: "Current simulation time. Increments with execution of Timer Task 1."
  flags: READONLY
  unit: "s"
}
modelStepSize
{
  type: flt(64,IEEE)
  alias: "modelStepSize"
  desc: "Fixed step size of the model, sample time of Timer Task 1."
  flags: READONLY
  unit: "s"
}

```



```
simState
{
  type: int(32)
  alias: "simState"
  desc: "Simulation state: STOP=0 RUN=2"
  unit: "-"
}
p_msg_last_error_no
{
  type: uint(32)
  alias: "errorNumber"
  desc: "Error number of last error message (zero if no error).\"
  unit: "-"
  flags: READONLY
}
p_dsts_sum_of_reset_time
{
  type: flt(64,IEEE)*
  alias: "sumOfResetTime"
  desc: "Internal variable for summing up reset time.\"
  unit: "s"
  flags: READONLY|HIDDEN
}
```

```
-- ***** Task Information variables*****
group "Task Info"

    group "Timer Task 1"
    pRti_TIMERA_STime
    {
        type: flt(64,IEEE)*
        alias: "sampleTime"
        flags: READONLY
    }
    pRti_TIMERA_TTime
    {
        type: flt(64,IEEE)*
        alias: "turnaroundTime"
        flags: READONLY
    }
    pRti_TIMERA_TState
    {
        type: int(32)*
        alias: "state"
        flags: READONLY
    }
    Rti_TIMERA_OType
    {
        type: int(32)*
        alias: "overflowCheckType"
        flags: READONLY
    }
    pRti_TIMERA_OMax
    {
        type: int(32)*
        alias: "overflowQueueMax"
        flags: READONLY
    }
    Rti_TIMERA_ORpt
    {
        type: int(32)*
        alias: "overflowQueueCount"
        flags: READONLY
    }
    pRti_TIMERA_OCnt
    {
        type: int(32)*
        alias: "overflowCount"
        flags: READONLY
    }
    pRti_TIMERA_TCnt
    {
        type: flt(64,IEEE)*
        alias: "taskCallCount"
        flags: READONLY
    }
    pRti_TIMERA_Prio
    {
        type: int(32)*
        alias: "priority"
        flags: READONLY
    }
endgroup
endgroup
```

```
-- ***** Model variables *****
group "Model Root"

  group "Integrator 1" -- block-group
  {
    block: "Integrator"
  }

    p_0_smd_1007_sl_real_T_0[4]
    {
      type:   flt(64,IEEE)*
      alias:  "Out1"
      flags:  OUTPUT|READONLY
    }
    p_1_smd_1007_sl_real_T_0[6]
    {
      type:   flt(64,IEEE)*
      alias:  "InitialCondition"
      flags:  PARAM
    }
  }
endgroup -- block-group "Integrator 1"

...
p_0_smd_1007_sl_real_T_0[7]
{
  type:   flt(64,IEEE)*
  alias:  "a"
  origin: "smd_1007_sl/Equation Block/a"
  flags:  READONLY
}
p_0_smd_1007_sl_real_T_0[4]
{
  type:   flt(64,IEEE)*
  alias:  "v"
  origin: "smd_1007_sl/Integrator 1/Out1"
  flags:  READONLY
}
p_0_smd_1007_sl_real_T_0[0]
{
  type:   flt(64,IEEE)*
  alias:  "x"
  origin: "smd_1007_sl/Integrator 2/Out1"
  flags:  READONLY
}
p_0_smd_1007_sl_real_T_0[1]
{
  type:   flt(64,IEEE)*
  alias:  "x disp"
  origin: "smd_1007_sl/x disp/Out1"
  flags:  READONLY
}
-- ***** Tunable Parameters*****
group "Tunable Parameters"

endgroup

-- ***** State Machine Data *****
-- No Stateflow chart within the model.
```

```
-- ***** Labels*****
group "Labels"

    p_0_smd_1007_sl_real_T_0[7]
    {
        type:    flt(64,IEEE)*
        alias:   "a"
        origin:  "smd_1007_sl/Equation Block/a"
        flags:   READONLY
    }
    p_0_smd_1007_sl_real_T_0[4]
    {
        type:    flt(64,IEEE)*
        alias:   "v"
        origin:  "smd_1007_sl/Integrator 1/Out1"
        flags:   READONLY
    }
    p_0_smd_1007_sl_real_T_0[0]
    {
        type:    flt(64,IEEE)*
        alias:   "x"
        origin:  "smd_1007_sl/Integrator 2/Out1"
        flags:   READONLY
    }
    p_0_smd_1007_sl_real_T_0[1]
    {
        type:    flt(64,IEEE)*
        alias:   "x disp"
        origin:  "smd_1007_sl/x disp/Out1"
        flags:   READONLY
    }
endgroup
-- ***** RTT Dynamic Variables *****
-- Generation of RTT Dynamic Variables turned off with EnableRealTimeTesting option.
-- ***** Blockset variables *****
-- ***** User variables from model_usr.trc**
-- RTI_USR_TRC_BEGIN
-- No user file smd_1007_sl_usr.trc found.
-- RTI_USR_TRC_END

-- ***** EESPort States Variables *****
group "XIL API"
    group "EESPort"

        p_xilapi_eesport_activeerrorset_uint32_T
        {
            type: uint(32)*
            alias: "Active ErrorSet"
            desc: "Index of active ErrorSet or 0 if no ErrorSet is active."
            flags: OUTPUT|READONLY
            range: < 0 ; 4294967295 >
        }
        ...
    endgroup -- "EESPort"
endgroup -- "XILAPI"

-- ***** [EOF] *****
```

Related topics

Examples

Example of Accessing Custom Variables in ControlDesk..... 172

RTI and RTI-MP Command Reference

Introduction

RTI and RTI-MP provide some commands that allow you to manage the RTI environment and your simulation models. You can use these commands from the MATLAB Command Window or in your M files.

Where to go from here

Information in this section

ds_trc_multiplelabeloccurrence	184
To activate TRC file entries for multiple labels.	
rti1xxx	185
To activate a different RTI platform support and open the corresponding RTI block library.	
rti_build2	185
To start the RTI build procedure for a given application.	
rti_mdcleanup	189
To configure the simulation parameters of a model for use with a MATLAB/Simulink environment without RTI installation.	
rti_optionget	189
To get the values of options supported by the RTI options pages of the Code Generation dialog.	
rti_optionset	191
To set the values of options supported by the RTI options pages of the Code Generation dialog.	
rti_sdfmerge	193
To update an SDF file with the contents of the <code><model>_usr.sdf</code> file of the model without rebuilding the entire model.	
rtimp_blktargetcpunameget	193
To return the name of the CPU which a given block belongs to.	
rtimp_build2	194
To start the RTI-MP build procedure for a given application.	

rtimp_targetswitch.....	197
To switch target-specific settings of an RTI-MP model.	
rtiver.....	198
To display the currently activated RTI platform support and the currently installed RTI version.	
set_rti.....	198
To configure the simulation parameters of a model for use with RTI.	

ds_trc_multiplelabeloccurrence

Syntax

```
ds_trc_multiplelabeloccurrence('set', <values>)
```

Purpose

To activate TRC file entries for multiple labels.

Description

As of dSPACE Release 2019-B, multiple labels with the same name no longer have an entry in the TRC file by default. If necessary, you can temporarily activate TRC file entries for multiple labels.

Note

- We strongly recommend not to activate this feature, because in the worst case it may cause you to measure wrong signals in the experiment software.
Using multiple labels with the same name in your variable description is on your own risk.
- The option to activate this feature will be discontinued in a future release. It will be available only temporarily to allow migration of existing models or scripts handling multiple labels.

Value	Description
0	Deactivates the generation of TRC file entries for multiple labels. This is the default setting for the TRC file generation.
1	Activates the generation of TRC file entries for multiple labels. The labels are then numbered by an index. The numbering may change after a rebuild of your model.

You have to note the following points:

- The activation must be set for each MATLAB session.
- Changing the setting does not modify the model. If there is no other change in the model, a rebuild does not start the code generation and the TRC file is not updated.

Examples

- To activate the feature:

```
ds_trc_multiplelabeloccurrence('set', 1)
```

- To deactivate the feature:

```
ds_trc_multiplelabeloccurrence('set', 0)
```

rti1xxx

Syntax

```
rti<XXXX>
```

Purpose

To activate a different RTI platform support and open the corresponding RTI block library.

Example

Type **rti1104** in the MATLAB Command Window to change to RTI1104 (DS1104 platform).

Related topics

HowTos

[How to Activate a Specific Platform Support \(RTI and RTI-MP Implementation Guide !\[\]\(56549452e01ca28bdf2500ced9653143_img.jpg\)](#))

rti_build2

Syntax

```
[errorFlag, errorMsg] = rti_build2 mdlName, 'Command', ...  
    <CommandValue>, <Parameter1>, <Value1>, ...  
    <Parameter2>, <Value2>, ...)
```

Note

Optional parameters are in bold format.

Purpose

To start the RTI build procedure for a given application.

Description

If you only specify the model name, the build procedure is started. By specifying the command value, you can use the command also to make a backup or to cleanup your build results.

If the build procedure stops with unspecific errors, you should firstly execute the cleanup procedure with the **CleanUpAll** value and then restart the build procedure. For example, the cleanup procedure is required after modifying the RCP and HIL environment by activating another installation or installing a software patch.

The command has the following outputs:

errorFlag 0 if no error occurred, otherwise a positive integer value.

errorMsg Empty string if no error occurred, otherwise a string containing information about the error.

The command has the following inputs:

mdlName (String) The name of the model. The model to be built must be open.

<CommandValue> (String) The action to be carried out. Depending on this command an automated build, backup or cleanup procedure is carried out. Each of the possible commands has specific optional parameter-value pairs. See below for details on the commands and the related parameter-value pairs.

<Parameter>, <Value> (String) Optional options for the given command, which are specified by parameter-value pairs. See below for details on the commands and the related parameter-value pairs.

Commands for the build procedure

If you do not specify the 'Command', **<CommandValue>** setting, **rti_build2** defaults to **CodeGen&Make&Load** of the build procedure.

The build procedure is carried out in three phases: code generation, make and load. The following values or combination of values are valid for the 'Command' to start the build procedure:

Value	Description
'CodeGen' or 'C'	Generates the real-time code for the model.
'CodeGen&Make' or 'CM'	Combines CodeGen and Make to compile and link the real-time code to the real-time application.
'CodeGen&Make&Load' or 'CML'	Combines CodeGen , Make and Load .
'Load' or 'L'	Loads the real-time application to the hardware. CodeGen or CodeGen&Make must be invoked beforehand.

Tip

Example:

```
[errorFlag, errorMsg] = rti_build2('MyModel', 'Command',
    'CodeGen&Make');
```

The following parameter-value pairs are valid for the build procedure:

- **SkipTaskConfiguration** To select whether the Task Configuration dialog is opened interactively to let you confirm any changed tasks of the model, for example, newly added interrupt blocks:

Value	Description
'on'	Skips the interactive Task Configuration during the build process (default).
'off'	Opens the Task Configuration dialog if necessary and stops the build process.

Note

When you add a new task to your model, the task automatically has a priority. However, this priority might not comply with your needs. Therefore, to avoid an unsuitable real-time application, you should check the task configuration of the model before you start the automated build procedure, especially if you select the **SkipTaskConfiguration**, 'on' setting.

- **LogOutput** To select whether the output from the MATLAB Command Window is saved to a text file:

Value	Description
'on'	Saves the output from the MATLAB Command Window to the <model>_rti<xxx>_log.txt file.
'off'	Does not save the output from the MATLAB Command window (default).

Tip

Example for the entire build procedure:

```
[errorFlag, errorMsg] = rti_build2('MyModel', 'Command',
    'CodeGen&Make', 'SkipTaskConfiguration', 'off', 'LogOutput',
    'on');
```

Commands for the backup procedure

The following value is valid for the 'Command' parameter to start the backup procedure:

Value	Description
'BackUp'	Creates a backup copy of the generated real-time application files that are necessary to load the application with ControlDesk.

Note

If you use model referencing, the backup procedure is applied only to the top-level model.

The following parameter-value pair is valid for the backup procedure:

- **BackUpFolder** To specify the location where the backup is saved. Make sure that the specified location exists before you start the backup procedure:

Value	Description
'<location>'	Saves the backup copy of the generated real-time application to this folder.

Commands for the cleanup procedure

The following values are valid for the 'Command' parameter to start the cleanup procedure:

Value	Description
'CleanUp'	Removes all the temporary files and folders from the current MATLAB folder, this includes a submodel's build directory with all its contents. As of dSPACE Release 2019-B, the command also removes the Simulink cache files (.SLXC).
'CleanUpAll'	Removes not only the temporary files and the Simulink cache files but also the generated real-time application. Only the Simulink model and the user files are kept.

Note

If you use model referencing, the cleanup procedure is applied only to top-level models. Temporary files located in the Simulink project directory are not changed since they might be relevant for other top-level models.

Related topics**HowTos**

[How to Automate the Build Process \(RTI and RTI-MP Implementation Guide !\[\]\(899d8b7697d64725bf017d3296cfcf1b_img.jpg\)\)](#)

References

[rtimp_build2..... 194](#)

rti_mdldcleanup

Syntax

```
rti_mdldcleanup
rti_mdldcleanup(<modelName>)
```

Purpose

To configure the simulation parameters of a model for use with a MATLAB/Simulink environment without RTI installation.

Result

rti_mdldcleanup configures the simulation parameters of the current model (or the model specified by <modelName>) to settings suitable for a MATLAB/Simulink environment without RTI installation.

Description

When a model is configured for use with RTI, it contains not only RTI-specific blocks but also block-independent, RTI-specific settings. These remain in the model even if you remove all the RTI-specific blocks. If you open such a model with a non-RTI environment, MATLAB issues appropriate warnings. Therefore, you should use the **rti_mdldcleanup** function to remove all the RTI-specific settings from the model and avoid the warnings.

To use the **rti_mdldcleanup** function, the model must not contain any RTI blocks.

Related topics

HowTos

[How to Remove the RTI-Specific Settings from a Model \(RTI and RTI-MP Implementation Guide\)](#)

References

[set_rti](#)..... 198

rti_optionget

Syntax

```
rti_optionget(modelHandle,<parameters>)
```

Purpose

To get the values of options supported by the RTI options pages of the Code Generation dialog. For information, refer to [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25.

Description

The following table lists the supported parameters and their values:

Parameter Name	Possible Values
InitialSimState	<ul style="list-style-type: none"> ▪ RUN ▪ PAUSE¹⁾ ▪ STOP
ExecutionMode	<ul style="list-style-type: none"> ▪ real-time ▪ time-scaled ▪ as fast as possible
TimeScaleFactor	Any numeric value > 0
AssertionMode	<ul style="list-style-type: none"> ▪ OFF ▪ WARN ▪ STOP
EnableRealTimeTesting	<ul style="list-style-type: none"> ▪ 0 ▪ 1
CCompilerCommonOpts	Any string
CCompilerOptimizationOptsPopup	<ul style="list-style-type: none"> ▪ Default ▪ None ▪ User-Defined
CCompilerOptimizationOpts	Any string ²⁾
EnableDataSetStorage	<ul style="list-style-type: none"> ▪ 0 ▪ 1³⁾
LoadAppl	<ul style="list-style-type: none"> ▪ ON ▪ OFF ▪ FLASH
PlatformSelectionPopup	<ul style="list-style-type: none"> ▪ Platform Name ▪ Network Client⁴⁾
BoardName	Any string ⁵⁾
NetworkClient	Any string ⁶⁾
TRCGenerateLabels	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateVirtualBlocks	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateStates	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateDerivates	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateParamValues	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCOnlyGlobals	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCIsOmitVdOn	<ul style="list-style-type: none"> ▪ 0 ▪ 1

¹⁾ The **PAUSE** simState is not supported by DS1007 and MicroLabBox (DS1202).

Parameter Name	Possible Values
²⁾ Only active if CCompilerOptimizationOptsPopup is set to User-Defined . ³⁾ Not supported for RTI1006 and RTI1104. For RTI1007 and RTI1202, the data set storage feature is always active and cannot be disabled. ⁴⁾ Network Client is only available for RTI1007, RTI1202 and RTI1401. ⁵⁾ For RTI1007, RTI1202 and RTI1401: Only active if PlatformSelectionPopup is set to Platform Name . ⁶⁾ Only active if PlatformSelectionPopup is set to Network Client . This option is only available for RTI1007, RTI1202 and RTI1401.	

Note

The settings of the following parameter groups are interdependent:

- CCompilerOptimizationOptsPopup and CCompilerOptimizationOpts
- PlatformSelectionPopup, BoardName (only for RTI1007, RTI1202, and RTI1401), and NetworkClient
- ExecutionMode and TimeScaleFactor

For example, the time-scaled simulation mode is only active, if the TimeScaleFactor parameter has a value > 0 *and* the ExecutionMode parameter is set to **time-scaled**.

Examples

```
optionValue = rti_optionget(bdroot,'TRCGenerateLabels')
optionValues = ...
rti_optionget(bdroot,{'TRCGenerateLabels','LoadAppl'})
```

rti_optionset

Syntax

```
rti_optionset(modelHandle,<parameters>,<values>)
```

Purpose

To set the values of options supported by the RTI options pages of the Code Generation dialog. For information, refer to [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 25.

Description

The following table lists the supported parameters and their values:

Parameter Name	Possible Values
InitialSimState	<ul style="list-style-type: none"> ▪ RUN ▪ PAUSE¹⁾ ▪ STOP

Parameter Name	Possible Values
ExecutionMode	<ul style="list-style-type: none"> ▪ real-time ▪ time-scaled ▪ as fast as possible
TimeScaleFactor	Any numeric value > 0
AssertionMode	<ul style="list-style-type: none"> ▪ OFF ▪ WARN ▪ STOP
EnableRealTimeTesting	<ul style="list-style-type: none"> ▪ 0 ▪ 1
CCompilerCommonOpts	Any string
CCompilerOptimizationOptsPopup	<ul style="list-style-type: none"> ▪ Default ▪ None ▪ User-Defined
CCompilerOptimizationOpts	Any string ²⁾
EnableDataSetStorage	<ul style="list-style-type: none"> ▪ 0 ▪ 1³⁾
LoadAppl	<ul style="list-style-type: none"> ▪ ON ▪ OFF ▪ FLASH
PlatformSelectionPopup	<ul style="list-style-type: none"> ▪ Platform Name ▪ Network Client⁴⁾
BoardName	Any string ⁵⁾
NetworkClient	Any string ⁶⁾
TRCGenerateLabels	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateVirtualBlocks	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateStates	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateDerivates	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCGenerateParamValues	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCOnlyGlobals	<ul style="list-style-type: none"> ▪ 0 ▪ 1
TRCIsOmitVdOn	<ul style="list-style-type: none"> ▪ 0 ▪ 1

¹⁾ The **PAUSE** simState is not supported by DS1007 and MicroLabBox (DS1202).

²⁾ Only active if CCompilerOptimizationOptsPopup is set to **User-Defined**.

³⁾ Not supported for RTI1006 and RTI1104. For RTI1007 and RTI1202, the data set storage feature is always active and cannot be disabled.

⁴⁾ Network Client is only available for RTI1007, RTI1202 and RTI1401.

⁵⁾ For RTI1007, RTI1202 and RTI1401: Only active if PlatformSelectionPopup is set to **Platform Name**.

Parameter Name	Possible Values
⁶⁾ Only active if PlatformSelectionPopup is set to Network Client . This option is only available for RTI1007, RTI1202 and RTI1401.	

Note

The settings of the following parameter groups are interdependent:

- CCompilerOptimizationOptsPopup and CCompilerOptimizationOpts
- PlatformSelectionPopup, BoardName (only for RTI1007, RTI1202, and RTI1401), and NetworkClient
- ExecutionMode and TimeScaleFactor

For example, the time-scaled simulation mode is only active, if the TimeScaleFactor parameter has a value > 0 *and* the ExecutionMode parameter is set to **time-scaled**.

Examples

```
rti_optionset(bdroot, 'TRCGenerateLabels', 1)
rti_optionset(bdroot, {'TRCGenerateLabels', 'LoadAppl'}, ...
{1, 'OFF'})
```

rti_sdfmerge

Syntax

```
rti_sdfmerge <model>.mdl
```

Purpose

To update an SDF file with the contents of the <model>_usr.sdf file of the model without rebuilding the entire model.

rtimp_blktargetcpunameget

Syntax

```
rtimp_blktargetcpunameget
rtimp_blktargetcpunameget(<mdlName>, <blockName>)
```

Purpose

To return the name of the CPU which a given block belongs to.

Result

If no arguments are specified, the name of the CPU is returned to which the currently selected block in the currently active model belongs.

If **<mdlName>** and **<blockName>** are specified, the name of the CPU to which the block belongs is returned.

Description

The function returns a struct. You will find the name of the CPU in the **targetcpu** field. If a block is assigned to multiple CPUs, the **targetcpu** field contains a list of CPU names. In addition, the **invalidtargetcpu** field is **1** only if the block is assigned to a single CPU, otherwise it is zero.

Valid for

RTI-MP

rtimp_build2

Syntax

```
[errorFlag, errorMsg] = rtimp_build2(mdlName, 'Command', ...
<CommandValue>, <Parameter1>, <Value1>, ...
<Parameter2>, <Value2>, ...)
```

Note

Optional parameters are in bold format.

Purpose

To start the RTI-MP build procedure for a given application.

Description

If you only specify the model name, the build procedure is started. By specifying the command value, you can use the command also to make a backup or to cleanup your build results.

If the build procedure stops with unspecific errors, you should firstly execute the cleanup procedure with the **CleanUpAll** value and then restart the build procedure. For example, the cleanup procedure is required after modifying the RCP and HIL environment by activating another installation or installing a software patch.

The command has the following outputs:

errorFlag 0 if no error occurred, otherwise a positive integer value.

errorMsg Empty string if no error occurred, otherwise a string containing information about the error.

The command has the following inputs:

mdlName (String) The name of the model. The model to be built must be open.

<CommandValue> (String) The action to be carried out. Depending on this command an automated build, backup or cleanup procedure is carried out. Each of the possible commands has specific optional parameter-value pairs. See below for details on the commands and the related parameter-value pairs.

<Parameter>, <Value> (String) Optional options for the given command, which are specified by parameter-value pairs. See below for details on the commands and the related parameter-value pairs.

Commands for the build procedure

If you do not specify the 'Command', **<commands>** setting, **rtimp_build2** defaults to **CodeGen&Make** of the build procedure.

The build procedure is carried out in three phases: code generation, make and load. The following values or combination of values are valid for the 'Command' to start the build procedure:

Value	Description
'CodeGen' or 'C'	Generates the real-time code for the model.
'CodeGen&Make' or 'CM'	Combines CodeGen and Make to compile and link the real-time code to the real-time application.
'CodeGen&Make&Load' or 'CML'	Combines CodeGen , Make and Load .
'Load' or 'L'	Loads the real-time application to the hardware. CodeGen or CodeGen&Make must be invoked beforehand.

The following parameter-value pairs are valid for the build procedure:

- **AppNames** To specify the desired applications of the multiprocessor model:

Value	Description
{ '-all' }	Processes all applications (default).
{ '<app1>', '<app2>', ... }	Processes only the specified applications.

- **KeepWorkingOnError** To select whether an error in one application stops the overall build procedure:

Value	Description
'on'	On an error occurring in one application, continue building the remaining applications (default).
'off'	On an error occurring, stop the overall build procedure.

- **SkipTaskConfiguration** To select whether the Task Configuration dialog is opened interactively to let you confirm any changed tasks of the model, for example, newly added interrupt blocks:

Value	Description
'on'	Skips the interactive Task Configuration during the build process (default).

Value	Description
'off'	Opens the Task Configuration dialog if necessary and stops the build process.

Note

When you add a new task to your model, the task automatically has a priority. However, this priority might not comply with your needs. Therefore, to avoid an unsuitable real-time application, you should check the task configuration of the model before you start the automated build procedure, especially if you select the `SkipTaskConfiguration`, 'on' setting.

- **LogOutput** To select whether the output from the MATLAB Command Window is saved to a text file:

Value	Description
'on'	Saves the output from the MATLAB Command Window to the <code><model>_rtimp_log.txt</code> file.
'off'	Does not save the output from the MATLAB Command window (default).

Command for the backup procedure

The following value is valid for the 'Command' parameter to start the backup procedure:

Value	Description
'BackUp'	Creates a backup copy of the generated real-time application files that are necessary to load the application with ControlDesk.

Note

If you use model referencing, the backup procedure is applied only to the top-level model.

The following parameter-value pair is valid for the backup procedure:

- **BackUpFolder** To specify the location where the backup is saved. Make sure that the specified location exists before you start the backup procedure:

Value	Description
'<location>'	Saves the backup copy of the generated real-time application to this folder.

Commands for the cleanup procedure

The following values are valid for the 'Command' parameter to start the cleanup procedure:

Value	Description
'CleanUp'	Removes all the temporary files and folders from the current MATLAB folder, this includes a submodel's build directory with all its contents. As of dSPACE Release 2019-B, the command also removes the Simulink cache files (.SLXC).
'CleanUpAll'	Removes not only the temporary files and the Simulink cache files but also the generated real-time application. Only the Simulink model and the user files are kept.

Note

If you use model referencing, the cleanup procedure is applied only to top-level models. Temporary files located in the Simulink project directory are not changed since they might be relevant for other top-level models.

The following parameter-value pair is valid for the cleanup procedure:

- **AppNames** To specify the desired applications of the multiprocessor model:

Value	Description
{ '-all' }	Processes all applications (default).
{ '<app1>', '<app2>', ... }	Processes only the specified applications.

Related topics

HowTos

[How to Automate the Build Process \(RTI and RTI-MP Implementation Guide !\[\]\(47734e4656765d20df4fdbd5b7aff048_img.jpg\)\)](#)

References

[rti_build2..... 185](#)

rtimp_targetswitch

Syntax

```
rtimp_targetswitch(<ModelName>)
```

Purpose

To switch target-specific settings of an RTI-MP model.

Result

This function switches target-specific settings for an RTI-MP model according to the currently active RTI platform.

Description

Examples of target-specific settings are the RTI-MP model's system target file and the selected platform type in the **Multiprocessor Setup** dialog. You can use this function, for example, before starting the build process with RTI-MP to ensure that target-specific settings are up-to-date regarding the current RTI platform.

Supported RTI platforms for this function are: RTI1006, RTI1007, and RTI1202.

rtiver

Syntax

```
rtiver
```

Purpose

To display the currently activated RTI platform support and the currently installed RTI version.

Tip

- You can use the `rti1xxx` command to activate a different RTI platform support. Refer to [rti1xxx](#) on page 185.
- You can use the `ver` command to display version information on MathWorks® products. For details, refer to the MATLAB Function Reference.

Example

The following example shows a typical output of the `rtiver` command:

```
>> rtiver
RTI1202 7.13 (02-Nov-2019)
```

set_rti

Syntax

```
set_rti
set_rti('-quiet')
set_rti(<ModelName>)
set_rti(<ModelName>,'-quiet')
```

Purpose

To configure the simulation parameters of a model for use with RTI.

Result

This function configures the simulation parameters of the current model to settings suitable for code generation with the currently active platform, for example, the system target file, template makefile, make command, etc.

If you do not want to reconfigure the current model, you can also specify a model by `<modelName>`.

This function makes use of modal dialogs which can be suppressed by using the optional `'-quiet'` flag.

Related topics**References**

rti_mdcleanup.....	189
------------------------------------	-----

RTI and RTI-MP Task Configuration API Reference

Introduction

RTI and RTI-MP provide an API for managing the task configuration by script instead of using the RTI Task Configuration dialog. You can use these API commands in the MATLAB Command Window or in M files.

Template files

Your RTI and RTI-MP installation provides template files with some basic instructions for task configuration.

To open the files, type the following command in the MATLAB Command Window:

- When you use RTI: `edit rti_task_api_template.m`
- When you use RTI-MP: `edit rtimp_task_api_template.m`

Note

Save the open template file in your working directory before you start modifying it.

Where to go from here

Information in this section

Task Configuration Classes.....	202
Task Configuration Methods.....	207

Information in other sections

[How to Configure Tasks via Scripts \(RTI and RTI-MP Implementation Guide !\[\]\(b4eeff342f60cc7bcd67d869b4fedca2_img.jpg\)\)](#)

Task Configuration Classes

Introduction

The Task Configuration object provides different classes for RTI models and RTI-MP models.

Where to go from here

Information in this section

TaskManager	202
To instantiate a task configuration object for an RTI model.	
MainModelTaskManager	203
To instantiate a task configuration object for an RTI-MP main model.	
SubModelTaskManager	205
To instantiate a task configuration object for an RTI-MP submodel.	

TaskManager

Syntax

```
myTaskManager = RTI.TaskConfiguration.TaskManager(ModelName)
```

Purpose

To instantiate a task configuration object for an RTI model.

Parameters

The following parameters are used:

Parameter	Type	Description
ModelName	string	Lets you specify the name of the model whose task configuration you want to access.

Return value

This function returns a **TaskManager** object.

Methods

The **TaskManager** object supports the following methods:

- [GetAllTasks](#) on page 209
- [GetGroupTasks](#) on page 210
- [GetModelTasks](#) on page 211
- [GetTaskGroupNames](#) on page 213
- [GetTaskGroupPriority](#) on page 214
- [GetTaskNamesByGroup](#) on page 215

- [GetTaskOverrunBehavior](#) on page 217
- [GetTaskOverrunQueueLength](#) on page 218
- [GetTaskPriority](#) on page 219
- [GetTaskSampleTime](#) on page 220
- [IsTaskConfigurationChanged](#) on page 221
- [SetOverrunQueueLength](#) on page 222
- [SetTaskPriority](#) on page 224
- [SetTaskGroupPriority](#) on page 225
- [SetTaskOverrunBehavior](#) on page 226
- [TurnInfoMessageOff](#) on page 228
- [TurnInfoMessageOn](#) on page 228
- [Commit](#) on page 208

Exceptions

An exception is generated in the following cases:

- The method was called without any parameters.
- The **modelName** parameter is not a string.
- The specified model is not on the MATLAB search path.
- The specified model does not exist.
- The specified model is not a Simulink model.
- The specified model is not open.
- The specified platform is not an RTI platform.
- The specified RTI platform differs from the currently active RTI platform.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example

The following example shows you how to create a **TaskManager** object for configuring the tasks in the *myModelName* model.

```
myTaskManager = RTI.TaskConfiguration.TaskManager('myModelName');
```

MainModelTaskManager

Syntax

```
myTaskManager =  
    RTIMP.TaskConfiguration.MainModelTaskManager(modelName)
```

Purpose

To instantiate a task configuration object for an RTI-MP main model. The RTI-MP main model has to contain the **Multiprocessor Setup** block.

Parameters The following parameter is used:

Parameter	Type	Description
ModelName	string	Lets you specify the name of the RTI-MP main model for which you want to access its task configuration.

Return value This function returns a `MainModelTaskManager` object.

Methods The `MainModelTaskManager` object supports the following methods:

- [GetSubModelNames](#) on page 212
- [GetSubModelTaskManager](#) on page 213
- [Commit](#) on page 208

Exceptions An exception is generated in the following cases:

- The method was called without any parameters.
- The `ModelName` parameter is not a string.
- The specified model is not on the MATLAB search path.
- The specified model does not exist.
- The specified model is not a Simulink model.
- The specified model is not open.
- The specified platform is not an RTI platform.
- The specified RTI platform differs from the currently active RTI platform.
- The specified model does not contain a `Multiprocessor Setup` block.

The identifier of the exception is: `RTI_MAIN_MODEL_TASK_MANAGER:BadInput`

Example The following example shows you how to create a `MainModelTaskManager` object for configuring the tasks in the *myMainModelName* RTI-MP main model.

```
myMainModelTaskManager =  
RTIMP.TaskConfiguration.MainModelTaskManager('myMainModelName');
```

Related topics

References

[SubModelTaskManager..... 205](#)

SubModelTaskManager

Syntax

```
mySubModelTaskManager =  
myMainModelTaskManager.GetSubModelTaskManager(SubmodelName)
```

Purpose

To instantiate a task configuration object for an RTI-MP submodel.

Description

An object of `SubModelTaskManager` class can be instantiated only by using the `GetSubModelTaskManager` method of a `MainModelTaskManager` object. You must not directly instantiate a `SubModelTaskManager` object.

Parameters

The following parameters are used:

Parameter	Type	Description
SubmodelName	string	Lets you specify the name of the RTI-MP submodel whose task configuration you want to access.

Return value

This function returns a `SubModelTaskManager` object.

Methods

The `SubModelTaskManager` object supports the following methods:

- [GetAllTasks](#) on page 209
- [GetGroupTasks](#) on page 210
- [GetModelTasks](#) on page 211
- [GetTaskGroupNames](#) on page 213
- [GetTaskGroupPriority](#) on page 214
- [GetTaskNamesByGroup](#) on page 215
- [GetTaskOverrunBehavior](#) on page 217
- [GetTaskOverrunQueueLength](#) on page 218
- [GetTaskPriority](#) on page 219
- [GetTaskSampleTime](#) on page 220
- [IsTaskConfigurationChanged](#) on page 221
- [SetOverrunQueueLength](#) on page 222
- [SetTaskPriority](#) on page 224
- [SetTaskGroupPriority](#) on page 225
- [SetTaskOverrunBehavior](#) on page 226
- [TurnInfoMessageOff](#) on page 228
- [TurnInfoMessageOn](#) on page 228

Exceptions

An exception is generated in the following cases:

- The method was called without any parameters.
- The **SubModel1Name** parameter is not a string.
- The specified submodel is not on the MATLAB search path.
- The specified submodel does not exist.
- The specified submodel is not a Simulink model.
- The specified submodel is not open.
- The specified platform is not an RTI platform.
- The specified RTI platform differs from the currently active RTI platform.

The identifier of the exception is: **RTI_SUB_MODEL_TASK_MANAGER:BadInput**

Example

The following example shows you how to create **SubModelTaskManager** objects for an RTI-MP model with two submodels.

```
myMainModelTaskManager =
RTIMP.TaskConfiguration.MainModelTaskManager('myMainModelName');
mySubModel1TaskManager =
myMainModelTaskManager.GetSubModelTaskManager('mySubModel1Name');
mySubModel2TaskManager =
myMainModelTaskManager.GetSubModelTaskManager('mySubModel2Name');
```

Related topics**References**

[MainModelTaskManager](#)..... 203

Task Configuration Methods

Introduction

The Task Configuration API provides methods for getting and setting relevant task attributes.

Where to go from here

Information in this section

Commit.....	208
To apply the configured changes in the model.	
GetAllTasks.....	209
To get all task names available in the model.	
GetGroupTasks.....	210
To get the task names of all tasks in a task group.	
GetModelTasks.....	211
To get the names of those tasks that you can configure.	
GetSubModelNames.....	212
To get the names of the submodels in your RTI-MP model.	
GetSubModelTaskManager.....	213
To get the class instance of the <code>SubModelTaskManager</code> of the specified submodel in your RTI-MP main model.	
GetTaskGroupNames.....	213
To get the names of the available task groups.	
GetTaskGroupPriority.....	214
To get the task priorities of the specified task groups.	
GetTaskNamesByGroup.....	215
To get the names of the tasks in the specified task group.	
GetTaskOverrunBehavior.....	217
To get the task overrun strategies of the specified tasks.	
GetTaskOverrunQueueLength.....	218
To get the task overrun queue lengths of the specified tasks.	
GetTaskPriority.....	219
To get the task priorities of the specified tasks.	
GetTaskSampleTime.....	220
To get the sample times of the specified tasks.	
IsTaskConfigurationChanged.....	221
To get the modification state of the task configuration.	
SetOverrunQueueLength.....	222
To set the queue length of the overrun task instances.	

SetTaskPriority.....	224
To set the task priority.	
SetTaskGroupPriority.....	225
To set the task group priority.	
SetTaskOverrunBehavior.....	226
To set the task overrun strategy.	
TurnInfoMessageOff.....	228
To turn off the display of information messages for all class instances.	
TurnInfoMessageOn.....	228
To turn off the display of information messages for all class instances.	

Commit

Syntax	<code>Obj.Commit();</code>
Purpose	<p>To apply the configured changes in the model.</p> <p>For RTI-MP models, the changes of the task configuration in a sub model are applied if you call the Commit method of the MainModelTaskManager object.</p>
Parameters	None
Related TaskConfiguration classes	<p>This method can be accessed by the following TaskConfiguration classes:</p> <ul style="list-style-type: none"> ▪ TaskManager on page 202 ▪ MainModelTaskManager on page 203
Exceptions	<p>An exception is generated in the following cases:</p> <ul style="list-style-type: none"> ▪ The analyzed model is no longer on the MATLAB search path. ▪ The model file of the analyzed model does not exist. ▪ The analyzed model is not a Simulink model. ▪ The analyzed model is not open. ▪ The analyzed model is configured for a not supported platform. ▪ The configured RTI platform differs from the currently active RTI platform. ▪ For a MainModelTaskManager instance: <ul style="list-style-type: none"> The specified main model does not have a Multiprocessor Setup block.

The identifier of the exception depends on the related class instance:

Class	Exception
TaskManager	RTI_TASK_MANAGER:BadInput
MainModelTaskManager	RTI_MAIN_MODEL_TASK_MANAGER:BadInput

Example

The following example shows you how to commit a modified task priority for an RTI model.

```
open_system('myModelName');
try
    myTaskManager = RTI.TaskConfiguration.TaskManager('myModelName');
    myTaskManager.SetTaskPriority('Timer Task 1', 1);
    myTaskManager.Commit();
    save_system('myModelName');
catch ME
    disp('TaskManager has generated an exception.');
```

The following example shows you how to commit a modified task priority for an RTI-MP model.

```
open_system('myMainModelName');
try

myMainTaskManager = RTIMP.TaskConfiguration.MainModelTaskManager('myMainModelName
');
    mySubTaskManager = myMainTaskManager.GetSubModelTaskManager('mySubModelName');
    mySubTaskManager.SetTaskPriority('Timer Task 1', 1);
    myMainTaskManager.Commit();
    save_system('myMainModelName');
catch ME
    disp('SubModelTaskManager or MainModelTaskManager has generated an exception.');
```

GetAllTasks

Syntax

```
Value = Obj.GetAllTasks();
```

Purpose

To get all task names available in the model.

Parameters

None

Return value

The following value is returned:

Type	Description
cellstring	Returns the names of all the available tasks in the model.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

None

Example

The following example shows you how to get all the task names which are available in the model.

```
myTasks = myTaskManager.GetAllTasks();
```

Related topics**References**

[GetModelTasks..... 211](#)

GetGroupTasks

Syntax

```
Value = Obj.GetGroupTasks();
```

Purpose

To get the task names of all tasks in a task group.

Parameters

None

Return value

The following value is returned:

Type	Description
cellstring	Returns the names of all the tasks available in task groups.

Related TaskConfiguration classes	<p>This method can be accessed by the following TaskConfiguration classes:</p> <ul style="list-style-type: none"> ▪ TaskManager on page 202 ▪ SubModelTaskManager on page 205 				
Exceptions	None				
Example	<p>The following example shows you how to get all the task names that are available in the task groups.</p> <pre>myTasks = myTaskManager.GetGroupTasks();</pre>				
Related topics	<p>References</p> <table> <tr> <td>GetTaskGroupNames.....</td> <td>213</td> </tr> <tr> <td>GetTaskNamesByGroup.....</td> <td>215</td> </tr> </table>	GetTaskGroupNames.....	213	GetTaskNamesByGroup.....	215
GetTaskGroupNames.....	213				
GetTaskNamesByGroup.....	215				

GetModelTasks

Syntax	<code>Value = Obj.GetModelTasks();</code>				
Purpose	To get the names of those tasks that you can configure.				
Parameters	None				
Return value	<p>The following value is returned:</p> <table> <tr> <th>Type</th><th>Description</th></tr> <tr> <td>cellstring</td><td>Returns the names of all the tasks in the model that you can configure.</td></tr> </table>	Type	Description	cellstring	Returns the names of all the tasks in the model that you can configure.
Type	Description				
cellstring	Returns the names of all the tasks in the model that you can configure.				
Related TaskConfiguration classes	<p>This method can be accessed by the following TaskConfiguration classes:</p> <ul style="list-style-type: none"> ▪ TaskManager on page 202 ▪ SubModelTaskManager on page 205 				
Exceptions	None				

Example

The following example shows you how to get all the task names that are available in the model.

```
myTasks = myTaskManager.GetModelTasks();
```

Related topics**References**

[GetAllTasks..... 209](#)

GetSubModelNames

Syntax

```
Value = Obj.GetSubModelNames();
```

Purpose

To get the names of the submodels in your RTI-MP model.

Parameters

None

Return value

The following value is returned:

Type	Description
cellstring	Returns the names of the submodels in the currently opened RTI-MP model.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [MainModelTaskManager](#) on page 203

Exceptions

None

Example

The following example shows you how to get the names of the submodels in your RTI-MP model.

```
mySubModelNames = myMainTaskManager.GetSubModelNames();
```

The following example shows you how to use the returned submodel names to get the **SubModelTaskManager** instance of the first submodel in your RTI-MP model.

```
mySubModelTaskManager =
myMainTaskManager.GetSubModelTaskManager(mySubModelNames{1});
```

Related topics**References**

[GetSubModelTaskManager.....](#) 213

GetSubModelTaskManager

Purpose

To get the class instance of the **SubModelTaskManager** of the specified submodel in your RTI-MP main model.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [MainModelTaskManager](#) on page 203

For more information on the instantiated **SubModelTaskManager** class, refer to [SubModelTaskManager](#) on page 205.

Related topics**References**

[GetSubModelNames.....](#) 212
[SubModelTaskManager.....](#) 205

GetTaskGroupNames

Syntax

```
Value = Obj.GetTaskGroupNames();
```

Purpose

To get the names of the available task groups.

Parameters

None

Return value

The following value is returned:

Type	Description
cellstring	Returns the names of the available task groups.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

None

Example

The following example shows you how to get the task group names that are available in the model.

```
myTasks = myTaskManager.GetTaskGroupNames();
```

Related topics**References**

GetGroupTasks.....	210
GetTaskGroupPriority.....	214
GetTaskNamesByGroup.....	215

GetTaskGroupPriority

Syntax

```
Value = Obj.GetTaskGroupPriority(TaskGroupNames);
```

Purpose

To get the task priorities of the specified task groups.

Parameters

The following parameters are used:

Parameter	Type	Description
TaskGroupNames	cellstring	Lets you specify the task groups for which to get the task group priorities.

Return value

The following value is returned:

Type	Description
vector	Returns the task group priorities of the specified task groups.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskGroupNames** parameter is not a string or a cell array of strings (vector cell array).
- The **TaskGroupNames** parameter contains a task group name that does not exist in the analyzed model.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example

The following example shows you how to get the priorities of the second, third and fourth task groups in the model.

```
myTaskGroups = myTaskManager.GetTaskGroupNames();
myTaskPriorities = myTaskManager.GetTaskGroupPriority(myTaskGroups(2:4));
```

The following example shows you how to get the priority of the *RTILINMM TX* task group.

```
myTaskGroupPriority = myTaskManager.GetTaskGroupPriority('RTILINMM TX');
```

Related topics**References**

GetGroupTasks.....	210
GetTaskGroupNames.....	213
GetTaskNamesByGroup.....	215
SetTaskGroupPriority.....	225

GetTaskNamesByGroup

Syntax

```
Value = Obj.GetTaskNamesByGroup(TaskGroupName);
```

Purpose To get the names of the tasks in the specified task group.

Parameters The following parameter is used:

Parameter	Type	Description
TaskGroupName	string	Lets you specify the task group for which to get the task names.

Return value The following value is returned:

Type	Description
cellstring	Returns the names of the tasks in the specified task group.

Related TaskConfiguration classes This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskGroupName** parameter is not a string.
- The **TaskGroupName** parameter contains a task group name that does not exist in the analyzed model.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example The following example shows you how to get the names of the tasks which are available in the *RTILINMM TX* task group to set the overrun queue length of the second task in the task group.

```
myGroupTasks = myTaskManager.GetTaskNamesByGroup('RTILINMM TX');
myTaskManager.SetOverrunQueueLength(myGroupTasks{2}, 3);
```

Related topics

References

[GetGroupTasks](#)..... 210

[GetTaskGroupNames](#)..... 213

GetTaskOverrunBehavior

Syntax

```
Value = Obj.GetTaskOverrunBehavior(TaskNames);
```

Purpose

To get the task overrun strategies of the specified tasks.

Parameters

The following parameter is used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the tasks for which you will get the task overrun strategies.

Return value

The following value is returned:

Type	Description
vector	<p>Returns the task overrun strategies of the specified tasks.</p> <ul style="list-style-type: none"> ▪ 1: Stop simulation ▪ 2: Queue tasks ▪ 3: Ignore overrun

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskNames** parameter is not a string or a cell array of strings (vector cell array).
- The **TaskNames** parameter contains a task name that does not exist in the analyzed model.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example

The following example shows you how to get the task overrun strategies of the second, third and fourth tasks in the model.

```
myTasks = myTaskManager.GetModelTasks();
myTaskOvrBehavior = myTaskManager.GetTaskOverrunBehavior(myTasks(2:4));
```

The following example shows you how to get the overrun strategy of the *Timer Task 1* task.

```
myTimerTaskOvrBehavior = myTaskManager.GetTaskOverrunBehavior('Timer Task 1');
```

Related topics

References

GetTaskOverrunQueueLength.....	218
SetTaskOverrunBehavior.....	226

GetTaskOverrunQueueLength

Syntax

```
Value = Obj.GetTaskOverrunQueueLength(TaskNames);
```

Purpose

To get the task overrun queue lengths of the specified tasks.

Parameters

The following parameter is used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the tasks for which to get the task overrun queue lengths.

Return value

The following value is returned:

Type	Description
vector	Returns the task overrun queue length of the specified tasks.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskNames** parameter is not a string or a cell array of strings (vector cell array).
- The **TaskNames** parameter contains a task name that does not exist in the analyzed model.

The identifier of the exception is: `RTI_TASK_MANAGER:BadInput`

Example

The following example shows you how to get the task overrun queue lengths of the second, third and fourth tasks in the model.

```
myTasks = myTaskManager.GetModelTasks();
myTaskOvrQueueLength = myTaskManager.GetTaskOverrunQueueLength(myTasks(2:4));
```

The following example shows you how to get the overrun strategy of the *Timer Task 1* task.

```
myTimerTaskOvrQueueLength =
myTaskManager.GetTaskOverrunQueueLength('Timer Task 1');
```

Related topics

References

[GetTaskOverrunBehavior.....](#) 217
[SetOverrunQueueLength.....](#) 222

GetTaskPriority

Syntax

```
Value = Obj.GetTaskPriority(TaskNames);
```

Purpose

To get the task priorities of the specified tasks.

Parameters

The following parameter is used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the tasks for which to get the task priorities.

Return value

The following value is returned:

Type	Description
vector	Returns the task priorities of the specified tasks.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskNames** parameter is not a string or a cell array of strings (vector cell array).
- The **TaskNames** parameter contains a task name that does not exist in the analyzed model.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example

The following example shows you how to get the priorities of the second, third and fourth task in the model.

```
myTasks = myTaskManager.GetModelTasks();
myTaskPriorities = myTaskManager.GetTaskPriority(myTasks(2:4));
```

The following example shows you how to get the priority of the *Timer Task 1* task.

```
myTimerTaskPriority = myTaskManager.GetTaskPriority('Timer Task 1');
```

Related topics**References**

GetModelTasks.....	211
GetTaskGroupPriority.....	214
SetTaskPriority.....	224

GetTaskSampleTime

Syntax

```
Value = Obj.GetTaskSampleTime(TaskNames);
```

Purpose

To get the sample time of one task or multiple tasks specified by the **TaskNames** parameter.

Parameters

The following parameter is used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the tasks for which to get the sample time.

Return value

The following value is returned:

Type	Description
vector	Returns the sample time values of the specified tasks. The return value for a sample time is set to -1, if the task is asynchronous, i.e., it does not have a discrete sample time.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskNames** parameter is not a string or a cell array of strings (vector cell array).
- The **TaskNames** parameter contains a task name that does not exist in the analyzed model.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example

The following example shows you how to get the sample time of the second, third and fourth task in the model.

```
myTasks = myTaskManager.GetModelTasks();
mySampleTimes = myTaskManager.GetTaskSampleTime(myTasks(2:4));
```

The following example shows you how to get the sample time of the *Timer Task 1* task.

```
mySampleTime = myTaskManager.GetTaskSampleTime('Timer Task 1');
```

Related topics**References**

[GetModelTasks.....](#) 211

IsTaskConfigurationChanged

Syntax

```
Value = Obj.IsTaskConfigurationChanged();
```

Purpose

To get the modification state of the task configuration.

Description Any model modification that has an affect on the task handling, such as, adding a task to the model or deleting a task from the model, leads to the modified state of the task configuration. The modified state shows that the task configuration that was loaded when the TaskManager object was instantiated differs from the task configuration that is stored in the model.

Parameters This method has no parameters.

Return value The following value is returned:

Type	Description
FALSE (0)	There is no model modification that has an affect on the task configuration.
TRUE (1)	There is a model modification that has an affect on the task configuration.

Related TaskConfiguration classes This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions None

Example The following example shows you how to use this method.

```
if myTaskManager.IsTaskConfigurationChanged()
    ...
end;
```

Related topics References

[GetModelTasks..... 211](#)

SetOverrunQueueLength

Syntax `Obj.SetOverrunQueueLength(TaskNames, Value)`

Purpose To set the queue length of the overrun task instances.

Parameters The following parameters are used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the overrun tasks whose queue lengths you want to set.
Value	vector	Lets you specify the queue length for the task specified in the TaskNames parameter in the range 0 ... $2^{31}-1$.

Return value None

Related TaskConfiguration classes This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskNames** parameter is not a string or cell array of strings (vector cell array).
- The **TaskNames** parameter contains a task group name that does not exist in the analyzed model.
- The **Value** parameter is not a double or double vector array.
- The **Value** parameter is out of range.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example The following example shows you how to set task 2 to the queue length of 5, task 3 to the queue length of 10, and task 4 to the queue length of 20.

```
myTasks = myTaskManager.GetModelTasks;
myTaskManager.SetOverrunQueueLength(myTasks(2:4),[5 10 20]);
```

The following example shows you how to set the overrun task with the name *Timer Task 1* to the queue length of 3.

```
myTaskManager.SetOverrunQueueLength('Timer Task 1',3);
```

Related topics

References

GetTaskOverrunQueueLength.....	218
SetTaskOverrunBehavior.....	226

SetTaskPriority

Syntax

```
Obj.SetTaskPriority(TaskNames, Value)
```

Purpose

To set the task priority.

Parameters

The following parameters are used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the tasks whose priorities you want to set. If you called GetModelTasks beforehand, you can use the determined list of task names to specify the tasks by their position. If you want to set the priority for only one task, you can specify it directly by its name. For backward compatibility, it is allowed to specify the priorities of task groups by specifying task group names. However, it is recommended to use the SetTaskGroupPriority method to specify the priorities of task groups.
Value	vector	Lets you specify the priority for the tasks specified in the TaskNames parameter in the range 1 ... 128. The highest priority is 1.

Note

The specified priorities are automatically adapted to continuously increasing values. For example, if you specify the priorities [3 8 1], the vector is changed to [2 3 1].

If you have specified further task priorities, they have an effect on the adapted priority values.

Return value

None

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The **TaskNames** parameter is not a string or cell array of strings (vector cell array).
- The **TaskNames** parameter contains a task name or task group name that does not exist in the analyzed model.

- The **Value** parameter is not a double or double vector array.
- The **Value** parameter is out of range.

The identifier of the exception is: **RTI_TASK_MANAGER:BadInput**

Example

The following example shows you how to set task 2 to priority 3, task 3 to priority 5, and task 4 to priority 1.

```
myTasks = myTaskManager.GetModelTasks;
myTaskManager.SetTaskPriority(myTasks(2:4),[3 5 1]);
```

The priorities might be changed to [2 3 1] to get continuously increasing priorities.

The following example shows you how to set the task with the name *Timer Task 1* to priority 1.

```
myTaskManager.SetTaskPriority('Timer Task 1',1);
```

Related topics

References

GetTaskPriority.....	219
SetTaskGroupPriority.....	225

SetTaskGroupPriority

Syntax

```
Obj.SetTaskGroupPriority(
    TaskGroupNames:cellstring,
    Value:vector)
```

Purpose

To set the task group priority.

Parameters

The following parameters are used:

Parameter	Type	Description
TaskGroupNames	cell string	Lets you specify the task groups whose priorities you want to set. If you called GetGroupTasks beforehand, you can use the determined list of task group names to specify the task groups by their position. If you want to set the priority for only one task group, you can specify it directly by its name.
Value	vector	Lets you specify the priority for the task groups specified in the TaskGroupNames parameter in the range 1 ... 127. The highest priority is 1.

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [MainModelTaskManager](#) on page 203
- [SubModelTaskManager](#) on page 205

Exceptions

The identifier of the exception is: `RTI_TASK_MANAGER:BadInput`

The exception is generated in the following cases:

- The method was called with insufficient parameters.
- The `TaskGroupNames` parameter is not a string or cell array of strings (vector cell array).
- The `TaskGroupNames` parameter contains a task group name that does not exist in the analyzed model.
- The `Value` parameter is not a double or double vector array.
- The `Value` parameter is out of range.

Example

The following example shows you how to set task group 1 to priority 2 and task group 2 to priority 1.

```
myTaskGroups = myTaskManager.GetGroupTasks;
myTaskManager.SetTaskGroupPriority(myTaskGroups(1:2),[2,1]);
```

The following example shows you how to set the task group with the name *RTILINMM TX* to priority 6.

```
myTaskManager.SetTaskGroupPriority('RTILINMM TX',6);
```

Related topics**References**

[GetTaskGroupPriority..... 214](#)

SetTaskOverrunBehavior

Syntax

```
Obj.SetTaskOverrunBehavior(TaskNames, Value)
```

Purpose

To set the task overrun strategy.

Parameters

The following parameters are used:

Parameter	Type	Description
TaskNames	cellstring	Lets you specify the task whose overrun strategies you want to set. If you called <code>GetModelTasks</code> beforehand, you can use the determined list of task names to specify the task by their position. If you want to set the overrun behavior for only one task, you can specify it directly by its name.
Value	vector	Lets you specify the overrun strategy for the task specified in the <code>TaskNames</code> parameter in the range 1 ... 3. <ul style="list-style-type: none"> ▪ 1: Stop simulation ▪ 2: Queue tasks ▪ 3: Ignore overrun

Return value

None

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

An exception is generated in the following cases:

- The method was called with insufficient parameters.
- The `TaskNames` parameter is not a string or cell array of strings (vector cell array).
- The `TaskNames` parameter contains a task name that does not exist in the analyzed model.
- The `Value` parameter is not a double or double vector array.
- The `Value` parameter is out of range.

The identifier of the exception is: `RTI_TASK_MANAGER:BadInput`

Example

The following example shows you how to set task 2 and task 3 to overrun strategy 1, and task 4 to overrun strategy 3.

```
myTasks = myTaskManager.GetModelTasks;
myTaskManager.SetTaskOverrunBehavior(myTasks(2:4), [1 1 3]);
```

The following example shows you how to set the task with the name *Timer Task 1* to overrun strategy 1.

```
myTaskManager.SetTaskOverrunBehavior('Timer Task 1',1);
```

Related topics**References**

GetTaskOverrunBehavior	217
SetOverrunQueueLength	222

TurnInfoMessageOff

Syntax

```
RTI.TaskConfiguration.TaskManager.TurnInfoMessageOff();
```

or

```
RTIMP.TaskConfiguration.SubModelTaskManager.TurnInfoMessageOff();
```

Purpose

To turn off the display of information messages for all class instances.

Parameters

None

Return value

None

Related TaskConfiguration classes

This method can be accessed by the following TaskConfiguration classes:

- [TaskManager](#) on page 202
- [SubModelTaskManager](#) on page 205

Exceptions

None

Related topics**References**

TurnInfoMessageOn	228
---	-----

TurnInfoMessageOn

Syntax

```
RTI.TaskConfiguration.TaskManager.TurnInfoMessageOn();
```

or

```
RTIMP.TaskConfiguration.SubModelTaskManager.TurnInfoMessageOn();
```

Purpose	To turn on the display of information messages for all class instances.
----------------	---

Parameters	None
-------------------	------

Return value	None
---------------------	------

Related TaskConfiguration classes	<p>This method can be accessed by the following TaskConfiguration classes:</p> <ul style="list-style-type: none">▪ TaskManager on page 202▪ SubModelTaskManager on page 205
--	--

Exceptions	None
-------------------	------

Related topics	References
-----------------------	-------------------

TurnInfoMessageOff.....	228
---	-----

General RTI and RTI-MP Libraries

Introduction	RTI and RTI-MP provide a number of general sublibraries, which contain blocks that are common to all dSPACE systems.
--------------	--

Where to go from here	Information in this section
	TaskLib Block Reference..... 232
	Extras Block Reference..... 248
	RTI-MP Blockset Reference..... 256
	RTI Gigalink Blockset Reference..... 272

TaskLib Block Reference

Where to go from here

Information in this section

Overview

[Overview of the Task Configuration Blockset.....](#) 233

Task configuration

[Function-Call Subsystem Block.....](#) 237

To insert a ready-to-use function-call subsystem triggered by a software or hardware interrupt.

[Timer Interrupt Block.....](#) 239

To make the timers of the platform available as interrupt sources.

[Hardware Interrupt Block.....](#) 235

To implement a custom solution for an I/O board where no hardware interrupt block is available.

[Software Interrupt Block.....](#) 238

To make software-generated interrupts available as trigger sources. Software interrupts can be used to build your own subscheduling of tasks.

[Timer Task Assignment Block.....](#) 246

To assign the connected hardware interrupt block, Time-Triggered Task block, or Timetable Task block as the trigger source for the model's timer task(s).

[Background Block.....](#) 234

To execute a function-call subsystem in the background task of the real-time application.

Time-triggered tasks

[Time-Trigger Set Block.....](#) 242

To set one or more task trigger delay times after which the corresponding time-triggered task is scheduled.

[Time-Triggered Task Block.....](#) 243

To execute the connected function-call subsystem whenever the task trigger delay time set by the corresponding Time-Trigger Set block is reached.

[Timetable Start Block.....](#) 244

To specify the timetable start delay time and start the timetable consisting of the associated Timetable Task blocks.

Timetable Task Block.....245

To specify the task trigger delay time and execute the connected function-call subsystem when the task trigger delay time (plus the optional timetable start delay time of the Timetable Start block) is reached.

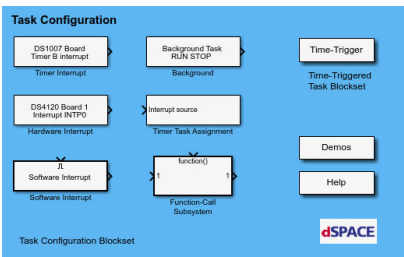
Information in other sections

Extras Block Reference.....248

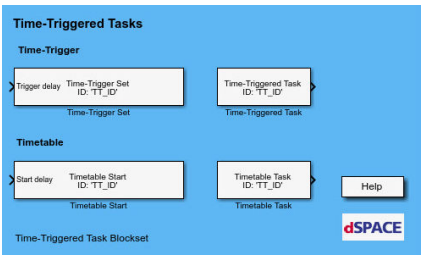
Overview of the Task Configuration Blockset

Introduction The Task Configuration Blockset provides blocks for creating tasks driven by various interrupts.

Overview of the blockset After you double-click the TaskLib button in the RTI library, the `rtitasklib` library opens.



The `rtitasklib` library contains the Time-Triggered Task Blockset, which is shown below:

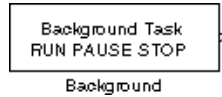


Note

The Time-Triggered Task Blockset is not available for RTI1104.

Background Block

Block



Purpose

To execute a function-call subsystem in the background task of the real-time application.

Description

Use this block to specify which parts of the Simulink model are executed in the background task. A background task is executed, whenever there is no real-time task currently executed in the model, for example, a timer task or a task driven by a hardware interrupt block. You must specify at least one simulation state for which the function-call subsystem is executed: for the RUN, PAUSE or STOP simulation states, or any combination of the three.

Dialog settings

Execute subsystem for simState RUN Select this checkbox if you want the function-call subsystem to be executed when the simulation state is set to RUN.

Execute subsystem for simState PAUSE Select this checkbox if you want the function-call subsystem to be executed when the simulation state is set to PAUSE.

Note

This option is not available for DS1007 and MicroLabBox (DS1202).

Execute subsystem for simState STOP Select this checkbox if you want the function-call subsystem to be executed when the simulation state is set to STOP.

Note

- Depending on the computational load of the foreground tasks and the fixed step size, the calculations performed in the background task can be either distributed over several base-sampling steps of the model or executed several times between two base-sampling steps.
- The service code for ControlDesk is also executed in the background task (see `host_service` in your board-specific *RTLib Reference*). If you make extensive use of Background blocks, this might slow down the updates of instruments in ControlDesk layouts and the parameter download to the real-time processor.

Related topics**Basics**

[Assigning Parts of A Model to The Background Task \(RTI and RTI-MP Implementation Guide !\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\)](#))

HowTos

[How to Assign Parts of Models to Background Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(a03a7eb2f4046e1d3c76772003e549ea_img.jpg\)](#))

References

[Function-Call Subsystem Block..... 237](#)
[host_service \(MicroAutoBox II RTLib Reference !\[\]\(e474458956c9a37fbf9586ddb60a7fa1_img.jpg\)](#))
[host_service \(DS1104 RTLib Reference !\[\]\(4d1d3f2547aeece54bb6babd23f4121b_img.jpg\)](#))
[host_service \(DS1006 RTLib Reference !\[\]\(ec45aa71601db5755c5e2662ad427708_img.jpg\)](#))

Hardware Interrupt Block

Block**Note**

This block is only for use with the PHS-bus-based systems and is intended for custom solutions only.

Purpose

To implement a custom solution for an I/O board where no hardware interrupt block is available.

Description

Several dSPACE I/O boards provide board-specific hardware interrupts. To make them available as trigger sources in a Simulink model, use the hardware interrupt blocks of the Controller board or I/O board you are addressing (see *Interrupts* and *Slave DSP Interrupts* in the *DS<xxx> RTI Reference* that corresponds to your board).

For some I/O boards you might want to implement a custom solution that is not available via their RTI blocks. In that case, you can use the Hardware Interrupt block found in the TaskLib library of PHS-bus-based systems.

Dialog settings

Board type Enter an I/O board in this edit field to access its hardware interrupt, for example, **DS4121**.

Board number Enter the board number in the range 1 ... 16.

Interrupt number Enter the interrupt number in the range INTP0 ... INTP7. This number must correspond to the interrupt number of the selected I/O board. You can find it in the interrupt table for the I/O board, see *DS<xxxx> RTI Reference*.

Perform sub-interrupt handling Select this checkbox if sub-interrupt handling is required.

Sub-interrupt number Enter a positive integer value or 0.
This edit field is enabled only if the **Perform sub-interrupt handling** checkbox is selected.

Use specific sub-interrupt handler Select this checkbox if a board-specific sub-interrupt handler is needed.
This checkbox is enabled only if the **Perform sub-interrupt handling** checkbox is selected.

Sub-interrupt handler function Specify the name of the board-specific sub-interrupt handler function. This function must have a prototype that matches *dSPACE Real-Time Kernel's* definition for an interrupt handler.
This edit field is enabled only if the **Use specific sub-interrupt handler** checkbox is selected.

Use custom code-generation file Select this checkbox if additional code must be generated for the selected hardware interrupt block.

TLC file name Specify the name of the TLC file that is used to generate additional code for the selected interrupt block.
This edit field is enabled only if the **Use custom code-generation file** checkbox is selected.

Note

- The TLC file you specify must be written according to the syntax of TLC files used for RTI's hardware interrupt blocks. For details, please contact <mailto:support@dspace.de>.
- Do not specify the extension of the TLC file.

Related topics

HowTos

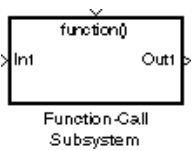
- [How to Combine Several Interrupt Sources \(RTI and RTI-MP Implementation Guide !\[\]\(86b7331e04fe40a56bcff2e9c065738b_img.jpg\)\)](#)
- [How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(92f87f30b7499b35d0173f4346c498d6_img.jpg\)\)](#)
- [How to Subschedule Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(497b6684f704c0aa6fbea9f0fd4d56c7_img.jpg\)\)](#)
- [How to Use Hardware Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(4320279ad715106747262028f44bd102_img.jpg\)\)](#)

References

Function-Call Subsystem Block.....	237
Timer Task Assignment Block.....	246
Timetable Start Block.....	244
Time-Trigger Set Block.....	242

Function-Call Subsystem Block

Block



Purpose

To insert a ready-to-use function-call subsystem triggered by a software or hardware interrupt.

Dialog settings

- Trigger type** The type of event that triggers the execution of the subsystem. Select *function-call*.
- Show output port** If this checkbox is selected, Simulink draws the Trigger block output and outputs the trigger signal.

Related topics**Basics**

[Assigning Parts of A Model to The Background Task \(RTI and RTI-MP Implementation Guide !\[\]\(eafc244b53721dd1ec133f0772f70fc7_img.jpg\)\)](#)
[Tasks Driven by Interrupt Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(cb741e910ae1fce3b15fcd4605753ff5_img.jpg\)\)](#)

HowTos

[How to Combine Several Interrupt Sources \(RTI and RTI-MP Implementation Guide !\[\]\(950a62bbddad88d64435fd35607dfc42_img.jpg\)\)](#)
[How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(80ae2b64037a63e4dd106d2cfb4205ab_img.jpg\)\)](#)
[How to Subschedule Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(9e6b464392878bce7cea642e72141689_img.jpg\)\)](#)
[How to Use Hardware Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(f5a23b4dd22b63e9bd2a86f3cac27ff1_img.jpg\)\)](#)
[How to Use Software Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(875f9de3b5d02ac3c4a42c2d3b9123e0_img.jpg\)\)](#)

References

Background Block.....	234
Hardware Interrupt Block.....	235
Software Interrupt Block.....	238
Timer Interrupt Block.....	239
Timetable Task Block.....	245
Time-Triggered Task Block.....	243

Software Interrupt Block

Block**Purpose**

To make software-generated interrupts available as trigger sources. Software interrupts can be used to build your own subscheduling of tasks.

Description

To trigger a Simulink subsystem by means of a software interrupt, connect the output of the Software Interrupt block to the trigger port of a function-call subsystem (see [Function-Call Subsystem Block](#) on page 237). The Software Interrupt block executes according to the settings of the trigger and enable ports. For example, if you use the default setting for the Software Interrupt block (Trigger port: *none*, Enable port: *yes*), the block triggers the corresponding function-call subsystem whenever the control signal has a positive value.

You can use any Simulink signal to trigger the execution of the Software Interrupt block, and thus to generate a software interrupt.

For further information on trigger and enable ports, refer to the Simulink documentation.

Dialog settings

Trigger port Lets you specify the signal type of the trigger port:

- none (default)
- rising
- falling
- either
- function-call

Enable port Lets you specify the activation of the enable port:

- none
- yes (default)

Note

Using the Software Interrupt block in the background task is not supported. That means you are not allowed to place the Software Interrupt block in a subsystem that is

- Connected to a Background Task block or
- Triggered from within a background task subsystem.

Related topics

HowTos

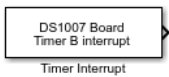
- [How to Combine Several Interrupt Sources \(RTI and RTI-MP Implementation Guide !\[\]\(1f101ad452ef9a3f01bb1e89af34fc34_img.jpg\)\)](#)
- [How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(30cdfe4eafd101fab5ecfaf690363fad_img.jpg\)\)](#)
- [How to Subschedule Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(4dcb2e0a5dd4ebc9597cee4f5b07c053_img.jpg\)\)](#)
- [How to Use Software Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(8cc998b11b9258f275abe61ed68f70ec_img.jpg\)\)](#)

References

Function-Call Subsystem Block.....	237
Timetable Start Block.....	244
Time-Trigger Set Block.....	242

Timer Interrupt Block

Block



Purpose

To make the timers of the platform available as interrupt sources.

As an example, the illustration above shows the Timer Interrupt block for the DS1007.

Description

Each platform provides various timers that are driven by the platform's bus clock. The Timer Interrupt block lets you use these timers as interrupt sources in a Simulink model. For example, if you specify *0.01* in the Period edit field of the block dialog, the selected timer generates one interrupt every 0.01 seconds.

The Source drop-down list of the Timer Interrupt block displays platform-independent names for the timer devices across all platforms. These names can differ from the names of the actual timer devices as shown in the table below. For example, if you specify *Timer B* of a DS1007, you actually select the board's Decrementer. The RTLib, feature and hardware references refer to the names of the timer devices.

The following table displays the bus clock frequencies that drive the corresponding timers. The bus clock frequency is referred to as BCLK.

Name in the Timer Interrupt Block	Hardware Device	Driving Frequency	Resolution	Count Range
DS1006 Timing Devices (BCLK = 133.16 MHz)				
Timer A interrupt	Timer A	BCLK/2	15.02 ns	32 bit
Timer B interrupt	Timer D	BCLK/2	30.04 ns	32 bit
Timer C interrupt	Timer B	BCLK/4	30.04 ns	32 bit
n/a	Time Stamp Counter	CPUCLK	0.455 ns	64 bit
DS1007 Timing Devices (BCLK = 100 MHz)				
Timer A interrupt	Timer A	BCLK/2	20 ns	32 bit
Timer B interrupt	Timer D	BCLK/2	20 ns	32 bit
Timer C interrupt	Timer B	BCLK/4	40 ns	32 bit
n/a	Time Stamp Counter	25 MHz	40 ns	64 bit
DS1104 Timing Devices (BCLK = 100 MHz)¹⁾				
Timer A interrupt	Timer 0	BCLK/8	80 ns	32 bit
Timer B interrupt	Timer 1	BCLK/8	80 ns	32 bit
n/a	Timer 2	BCLK/8	80 ns	32 bit
n/a	Timer 3	BCLK/8	80 ns	32 bit
Timer C interrupt	Decrementer	BCLK/4	40 ns	32 bit
n/a	Time Base Counter	BCLK/4	40 ns	64 bit
MicroAutoBox II Timing Devices (BCLK = 100 MHz)				
Timer A interrupt	Decrementer	BCLK/4	40 ns	32 bit
Timer B interrupt	Timer	BCLK/4 ²⁾	40 ns	32 bit
n/a	Time Base Counter	BCLK/4	40 ns	64 bit

Name in the Timer Interrupt Block	Hardware Device	Driving Frequency	Resolution	Count Range
MicroLabBox Timing Devices (BCLK = 100 MHz)				
Timer A interrupt	Timer A	BCLK/2	20 ns	32 bit
Timer B interrupt	Timer D	BCLK/2	20 ns	32 bit
Timer C interrupt	Timer B	BCLK/4	40 ns	32 bit
n/a	Time Stamp Counter	25 MHz	40 ns	64 bit

¹⁾ The bus clock of the DS1104 depends on the PCI bus clock of the host PC.

²⁾ Timer B interrupt of the DS1401 is scalable from 1/128 to 1/4 BCLK. However, RTKernel (and therefore RTI) always uses the highest resolution at 1/4 BCLK.

The minimum timer period is limited by the platform's bus clock frequency. For example, if the bus clock frequency of the DS1007 is 100 MHz, the minimum timer period of Timer A is 20 ns.

The maximum timer period is limited by the platform's bus clock and the resolution of the corresponding counter, which is 32 bits for all counters. For example, if the bus clock frequency of the DS1007 is 100 MHz, the maximum timer period of Timer A is 85.9 s ($= 2^{32} * 20 \text{ ns}$).

Tip

You can find the value of the platform's bus clock in ControlDesk's Properties dialog, refer to [Board Details Properties \(ControlDesk Platform Management\)](#).

Note

The minimum timer period a DS1007 or MicroLabBox (DS1202) can be used with is 3 μ s. A lower timer period might lead to unpredictable behavior.

Unit page

Source From this drop-down list, select a timer that serves as the interrupt source.

Note

- Each timer displayed in the Source edit field can only be used once as an interrupt source in a Simulink model.
- If your model contains timer tasks, Timer A of your platform provides the necessary timer interrupts for the timer task(s). Therefore, you cannot use Timer A as an interrupt source for such a model. However, you can use Timer A if the timer tasks of your model are driven by another hardware interrupt via the Timer Task Assignment block (refer to [Timer Task Assignment Block](#) on page 246).

Period Enter the timer period in this edit field. The default period is 0.01 sec.

Sample time Enter the sample time in this edit field. The default sample time is -1. The value determines the sample time of the Timer Interrupt block for Simulink simulation. It can differ from the defined timer period and does not affect real-time simulation.

Related topics

HowTos

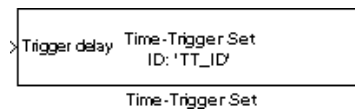
[How to Combine Several Interrupt Sources \(RTI and RTI-MP Implementation Guide !\[\]\(0f848bbd71cef6b345273b16f905912a_img.jpg\)\)](#)
[How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(d873c0073cfd3b74a7c9b5ca09bad0c7_img.jpg\)\)](#)
[How to Subschedule Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(9126fbb278b6412ee8b215b5e71dadba_img.jpg\)\)](#)
[How to Use Hardware Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(bb3ac0ef9759920456d29214b9245205_img.jpg\)\)](#)

References

[Board Details Properties \(ControlDesk Platform Management !\[\]\(a870788d6ed9b8fd294b7654a8c8526b_img.jpg\)\)](#)
 Function-Call Subsystem Block..... 237
 Timetable Start Block..... 244
 Time-Trigger Set Block..... 242

Time-Trigger Set Block

Block



Purpose

To set one or more task trigger delay times after which the corresponding time-triggered task is scheduled.

Description

When the Time-Trigger Set block is executed, it sets one or more task trigger delay times for the associated time-triggered task. The task trigger delay time (in seconds) is given by the Simulink signal connected to the input of the Time-Trigger Set block. If a vector signal is connected, each vector element results in a separate task trigger delay time for the task.

When a task trigger delay time is reached, the time-triggered task is scheduled.

Dialog settings

Time-Trigger ID Associates the Time-Trigger Set block with its Time-Triggered Task block. Make sure to use a unique ID for each pair of Time-Trigger Set and Time-Triggered Task blocks.

Sample time Lets you specify a sample time. If the block is placed in a function-call subsystem, you must always use an inherited sample time (-1).

Execute time-triggered task once at simulation start If selected, the associated time-triggered task is executed once on simulation start independently of the execution of the Time-Trigger Set block.

Note

This block is not available for RTI1104.

Related topics

HowTos

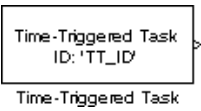
[How to Implement Time-Triggered Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(9a573f036489688f54df56b3b9218e70_img.jpg\)\)](#)

References

Hardware Interrupt Block.....	235
Software Interrupt Block.....	238
Timer Interrupt Block.....	239
Time-Triggered Task Block.....	243

Time-Triggered Task Block

Block



Purpose

To execute the connected function-call subsystem whenever the task trigger delay time set by the corresponding Time-Trigger Set block is reached.

Description

When the associated Time-Trigger Set block is executed, it sets one or more task trigger delay times for the time-triggered task. The task trigger delay time (in seconds) is given by the Simulink signal connected to the inport of the Time-Trigger Set block. If a vector signal is connected, each vector element results in a separate task trigger delay time for the task.

When a task trigger delay time is reached, the time-triggered task is scheduled.

Dialog settings

Time-Trigger ID Associates the Time-Triggered Task block with its Time-Trigger Set block. Make sure to use a unique ID for each pair of Time-Trigger Set and Time-Triggered Task blocks.

Note

This block is not available for RTI1104.

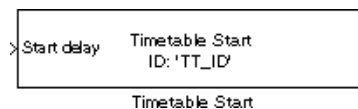
Related topics**HowTos**

[How to Implement Time-Triggered Tasks \(RTI and RTI-MP Implementation Guide !\[\]\(ec9132f1d27c8919987d92907322654d_img.jpg\)\)](#)

References

Function-Call Subsystem Block..... 237
 Timer Task Assignment Block..... 246
 Time-Trigger Set Block..... 242

Timetable Start Block

Block**Purpose**

To specify the timetable start delay time and start the timetable consisting of the associated Timetable Task blocks.

Description

When the Timetable Start block is executed, it sets the effective task trigger delay times for each associated timetable task. The effective task trigger delay times are the sum of:

- The timetable start delay time (in seconds) given by the Simulink signal that is connected to the Start delay inport of the Timetable Start block. If the Start delay inport is disabled, the timetable start delay time is 0.
- The individual task trigger delay times specified for the associated timetable tasks (see [Timetable Task Block](#) on page 245).

Dialog settings

Timetable ID Associates the Timetable Start block with its Timetable Task blocks. Make sure to use a unique ID for each set of Timetable Start block and associated Timetable Task blocks.

Sample time Lets you specify a sample time. If the block is placed in a function-call subsystem, you must always use an inherited sample time (-1).

Start timetable once at simulation start If selected, the timetable is started once on simulation start independently of the execution of the **Timetable Start** block.

Enable timetable start delay inport If selected, the entire timetable is delayed by the value of the Simulink signal connected to the inport of the **Timetable Start** block.

Note

This block is not available for RTI1104.

Related topics

HowTos

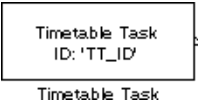
[How to Implement Timetables \(RTI and RTI-MP Implementation Guide !\[\]\(de9ec43bc9071aa4d7fe670071a18867_img.jpg\)\)](#)

References

[Hardware Interrupt Block..... 235](#)
[Software Interrupt Block..... 238](#)
[Timer Interrupt Block..... 239](#)
[Timetable Task Block..... 245](#)

Timetable Task Block

Block



Purpose

To specify the task trigger delay time and execute the connected function-call subsystem when the task trigger delay time (plus the optional timetable start delay time of the **Timetable Start** block) is reached.

Description

When the associated **Timetable Start** block is executed, it sets the effective task trigger delay times for each timetable task. The effective task trigger delay times are the sum of:

- The timetable start delay time (in seconds) given by the Simulink signal that is connected to the **Start delay** inport of the **Timetable Start** block. If the **Start delay** inport is disabled, the timetable start delay time is 0.

- The individual task trigger delay times specified for the timetable tasks. If a vector is specified as the Task trigger delays, one call of that timetable task is scheduled for each vector element.

Dialog settings

Timetable ID Associates the Timetable Task block with its Timetable Start block. Make sure to use a unique ID for each set of Timetable Start block and associated Timetable Task blocks.

Task trigger delays Lets you specify one or more task trigger delay times for the timetable task (i.e., scalar or vector). The effective task trigger delay time is the sum of its task trigger delay and the timetable start delay time provided at the Start delay inport of the Timetable Start block.

Note

This block is not available for RTI1104.

Related topics

HowTos

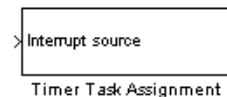
[How to Implement Timetables \(RTI and RTI-MP Implementation Guide !\[\]\(73002692dd5e7a64e60946be3158e719_img.jpg\)](#))

References

Function-Call Subsystem Block.....	237
Timer Task Assignment Block.....	246
Timetable Start Block.....	244

Timer Task Assignment Block

Block



Purpose

To assign the connected hardware interrupt block, Time-Triggered Task block, or Timetable Task block as the trigger source for the model's timer task(s).

Note

Do not connect a Timer Interrupt block or a Software Interrupt block to the Timer Task Assignment block.

Description

Normally, a timer of your dSPACE real-time board provides the timer interrupts for your model's timer task(s). As an alternative, the Timer Task Assignment block allows you to specify external events as the interrupt source for the timer task(s). Therefore, you can easily synchronize the timer task execution with the events that occur on an I/O device connected to your dSPACE system, for example.

You can use the Timer Task Assignment block as follows:

- Connect it to any hardware interrupt block provided by the appropriate I/O library or by the RTI TaskLib. This synchronizes the timer tasks of the model to the interrupt signal of this hardware interrupt.
- Connect it to a Time-Triggered Task or Timetable Task block while the Time-Trigger Set block or Timetable Start block is driven by any hardware interrupt block. This also synchronizes the timer tasks of the model to the interrupt signal of the hardware interrupt block but using a variable delay time. During the real-time simulation you can easily modify the delay time.
- Connect it to a Gigalink_Interrupt block. This allows synchronized communication between a PHS-bus application and a SCALEXIO application.

Note

The computations of the timer task are executed as normal using the time value of the simulation step size, as specified in the Solver dialog of the Model Explorer or in the Multiprocessor Setup dialog. However, the points in time when the computations are performed depend on the external trigger events. RTI cannot check whether the simulation step size of the model and the step size of the external trigger are equal. You must ensure identical step sizes yourself. Otherwise, the model is not calculated in real time.

Dialog settings

None

Related topics

HowTos

[How to Connect Timer Tasks to an External Interrupt Source \(RTI and RTI-MP Implementation Guide !\[\]\(51514032c8ca341817228f39f1307b05_img.jpg\)](#))

References

Hardware Interrupt Block.....	235
Timetable Task Block.....	245
Time-Triggered Task Block.....	243

Extras Block Reference

Where to go from here

Information in this section

Overview

[Overview of the Extras Blockset.....](#) 249

Variable access

[Data Capture Block.....](#) 249

To place the data acquisition host service code for ControlDesk in a specific task to control the rate at which data is sampled.

[TRC Exclusion Block.....](#) 254

To exclude all blocks (and other subsystems) in a subsystem from the generated variable description file (TRC file).

Simulation control

[simState READ Block.....](#) 252

To read the current simulation state in the Simulink model.

[simState SET Block.....](#) 253

To set the current simulation state from within the Simulink model.

Encoder index search

[Ramp Generator for Encoder Index Search Block.....](#) 251

To generate a ramp signal output that can be used as a reference signal during the index search of incremental encoders.

Task configuration

[Background Block.....](#) 234

To execute a function-call subsystem in the background task of the real-time application.

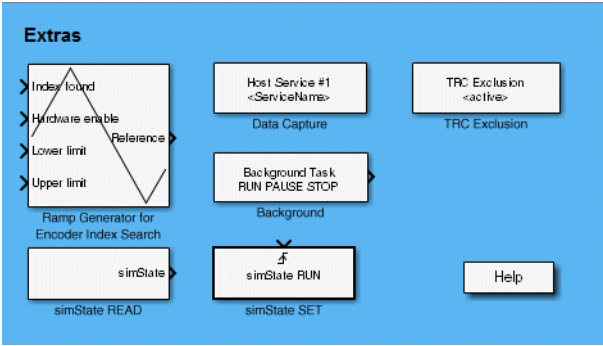
Information in other sections

[TaskLib Block Reference.....](#) 232

Overview of the Extras Blockset

Introduction The Extras library provides blocks that let you modify the data acquisition behavior or control the simulation state, for example.

Overview of the blockset After you double-click the Extras button in the RTI library, the dsextras library is displayed.



Data Capture Block

Block

Host Service #1
<ServiceName>

Data Capture

Purpose To place the data acquisition host service code for ControlDesk in a specific task to control the rate at which data is sampled.

Description ControlDesk needs a special host service code to be executed in the real-time application. This code collects one set of data for each sampling step. By default, the service code for data acquisition is executed in the fastest timer task.

If you use a Data Capture block, the default host service code for data sampling in the fastest timer task is omitted automatically.

Note

- If your model is driven only by interrupt blocks, you need to place the Data Capture block in a triggered subsystem of your model to specify the task in which the host service code for the data acquisition is executed.
- Do not place a Data Capture block in the background task of a model or in a subsystem that is triggered by a Software Interrupt block placed in the background task. Otherwise, you get unpredictable results since the background task does not provide meaningful time stamps.
- With RTI-MP, a Data Capture block is copied to all (sub-)system members, for example, a Data Capture block on the model root is copied to all CPUs.

Dialog settings

Service number The host service to be configured. The possible values are 1 ... 27.

By default, the host services 28 to 31 are reserved for monitoring features when using bus support.

If one of your models is using these host services, except for bus support, you can also set the **Allow usage of reserved host service numbers with Data Capture blocks** option in the **RTI general build options** page of the **Configuration Parameters** dialog to avoid a reconfiguration of the related blocks.

Service name Enter the service name in this edit field. In ControlDesk's **Measurement Configuration** dialog, its **Alias** name is displayed.

Note

If you run a multiprocessor system, the following applies:

- For distributed tracing the service name is used to define system-wide host services.
- If you want to perform distributed tracing for a specific interrupt block-driven task, you must place a Data Capture block in the corresponding triggered subsystem. Via IPC, IPI or Default CPU blocks, the data acquisition host service code generated for this Data Capture block is distributed to all CPUs that are involved in the task driven by the interrupt block. For distributed tracing, you must configure the IPC, IPI and Default CPU blocks so that the Data Capture block appears on all the CPUs of your system.

Sample time Enter the sample time at which the host service code is to be executed. Choose -1 to inherit the sample time – this is the default setting – or any multiple of the *Fixed step* size chosen for the model. If you use the Data Capture block in triggered subsystems, select -1 as the sample time.

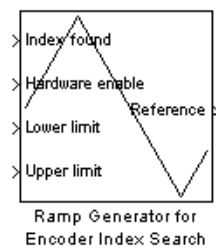
Note

Do not specify sample time offsets.

Related topics**HowTos**

[How to Modify Data Acquisition Behavior \(RTI and RTI-MP Implementation Guide !\[\]\(a03a7eb2f4046e1d3c76772003e549ea_img.jpg\)\)](#)

Ramp Generator for Encoder Index Search Block

Block**Note**

This block has different behavior for the different RTIs.

Purpose

To generate a ramp signal output that can be used as a reference signal during the index search of incremental encoders.

I/O characteristics

- RTI supports data typing for this block. If the block's output is of Boolean type, the block can be used with all logical operators without using **Data Type Conversion** blocks. In this case, the block's input for the **Lower** and **Upper** limit inputs – and for the PHS-bus-based systems also the **Hardware Enable** input – have to be of Boolean type. The **Index found** input supports all data types.
- The data type of the output always is double.
- The DS1104ENC_SW_INDEX_Cx block and also several DS<xxx>ENC_INDEX_Bx_Cy blocks of the I/O boards for the PHS-bus-based systems provide the **Index found** output, which you can connect to the **Index found** input of the Ramp Generator block.

The Index found input of the Ramp Generator block has the following effect:

Index found input	Result
0	Ramp signal is generated
-1	Speed factor is applied
1	Ramp is switched off

If the Hardware enable input is set to 1, searching is enabled. If set to 0, searching is disabled.

For example, if the Type of index search of the DS1104ENC_SW_INDEX_Cx block is set to *Search index twice for speed-up* for RTI1104, and no index has been found yet – that is Index found is set to -1 – then the ramp generator Reference output can be used to speed up the movement of the controlled device. When the index has been found once – that is Index found is set to 0 – the Reference output can be used to slow down the movement of the controlled device to the value defined by the Reference output increment/decrement parameter.

- The Lower Limit and Upper Limit inputs are used to indicate whether a limit is reached (1) or not (0). If one of the limits is set to 1, the Reference output signal is inverted. For example, when the controlled device reaches the end of a linear course, its direction is changed.

Dialog settings

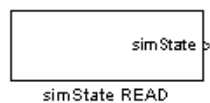
Reference output increment/decrement The value of the quantizing level of the ramp signal at the output.

Speedup factor Divided difference of the ramp signal: angle of lead $\varphi = \arctan(\text{Speedup factor})$.

Sample time The sample time of the task the ramp generator must be executed in. Values are -1 (inherited) or any multiple of the *Fixed step size* chosen for the model.

simState READ Block

Block





Purpose

To read the current simulation state in the Simulink model.

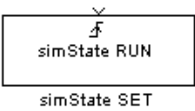
Description

The simState READ block reads the current simulation state.

Dialog settings	None
Related topics	<div>Basics</div> <div>Simulation Control (RUN/STOP Mechanism) (RTI and RTI-MP Implementation Guide )</div> <div>HowTos</div> <div>How to Set the Simulation State (RTI and RTI-MP Implementation Guide )</div> <div>References</div> <div>simState SET Block.....253</div>

simState SET Block

Block



Purpose	To set the current simulation state from within the Simulink model.
Description	<p>This block is a conditionally executed subsystem. When the block is executed, the simulation state of the real-time application is set to the value specified in the block’s dialog.</p> <div>Note RTI-MP This block is not valid for the slave CPUs of RTI-MP models.</div> <p>For further information on the trigger and enable ports, refer to the Simulink documentation.</p>

Dialog settings	<div>Trigger port Lets you specify the signal type of the trigger port:</div> <div><ul style="list-style-type: none">▪ none▪ rising▪ falling</div>
-----------------	---

- either
- function-call

Enable port Lets you specify the activation of the enable port:

- none
- yes

Set simState to Lets you specify the simulation state:

- RUN
- PAUSE
- STOP

Note

The *PAUSE* simState is not supported by DS1007 and MicroLabBox (DS1202).

Related topics

Basics

[Simulation Control \(RUN/STOP Mechanism\) \(RTI and RTI-MP Implementation Guide !\[\]\(d5d7044e5caf6907399af2dced8d6ff8_img.jpg\)\)](#)

HowTos

[How to Set the Simulation State \(RTI and RTI-MP Implementation Guide !\[\]\(ab4e2b3fc7e7887b7a72f548aa6f5e60_img.jpg\)\)](#)

References

[simState READ Block..... 252](#)

TRC Exclusion Block

Block



Purpose

To exclude all blocks (and other subsystems) in a subsystem from the generated variable description file (TRC file).

Result

All blocks in the subsystem where the TRC Exclusion block resides and all underlying subsystems are not generated into the TRC file.

Description

The block has two states: active and inactive, which you can set via the block dialog. The block itself displays the current state. You can add the block to any subsystem of a model.

When you build the model, TRC file generation stops inside the subsystem where the block resides. Only the outputs of the subsystem are available. If large parts of a model are excluded in this way, this reduces the size of the variable description file, which in turn results in faster loading in ControlDesk. You can also use this block to simply hide model parts from other users. For more information, refer to [Excluding Subsystems from the TRC File \(RTI and RTI-MP Implementation Guide !\[\]\(d84e7ea36f695d92cb39ec32c307ac93_img.jpg\)](#)).

Comments are added to the header of the generated TRC file, indicating whether the model contains active TRC Exclusion blocks. The affected subsystems are also listed.

Note

- An alternative to the TRC Exclusion block provides the Apply subsystem omission tags option. For details, refer to [RTI variable description file options page](#) on page 35 (RTI) or [Code Generation Dialog \(Model Configuration Parameters Dialogs\)](#) on page 80 (RTI-MP).
- With RTI-MP, a TRC Exclusion block is copied to all (sub-)system members, for example, a TRC Exclusion block on the model root is copied to all CPUs.

Dialog settings


Block mode Lets you set the state of the block.

There are two states:

- Active, which is set when 1 is entered.
- Inactive, which is set when 0 is entered.

You can also enter the value as a workspace variable.

Related topics

Basics	
Excluding Subsystems from the TRC File (RTI and RTI-MP Implementation Guide )	
References	
Code Generation Dialog (Model Configuration Parameters Dialogs)	80
Variable Description File (TRC File)	123

RTI-MP Blockset Reference

Where to go from here

Information in this section

[Real-Time Interface for Multiprocessor Systems.....](#) 256

[Configuration Blocks.....](#) 258

The Multiprocessor Setup block and the Default CPU block are used to configure the multiprocessor Simulink model.

[Interprocessor Communication Blocks.....](#) 261

The Interprocessor Communication (IPC) blocks are used to implement communication channels between the CPUs of a multiprocessor system.

[Interprocessor Interrupt Blocks.....](#) 268

The Interprocessor Interrupt (IPI) blocks are used to implement interrupts in a multiprocessor model.

Information in other sections

[Distributing the Model for MP Systems \(RTI and RTI-MP Implementation Guide !\[\]\(4688aadfd656ded00cd6bdfae55089a9_img.jpg\)](#))

Real-Time Interface for Multiprocessor Systems

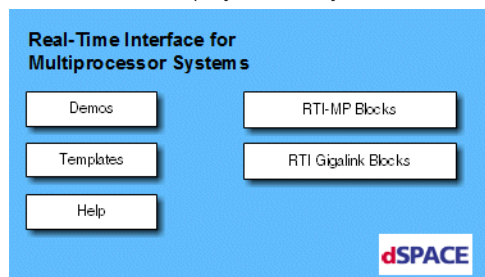
Overview of the RTI-MP Blockset

Introduction

The RTI-MP library `rtimplib` offers RTI-MP sublibraries with blocks for modeling multiprocessor models.

Overview of the blockset

The blockset is displayed when you enter `rtimp`.



It provides RTI-MP blocks; for example, to mark communication connections between submodels. These RTI-MP blocks are designed to distribute the Simulink model to the different CPUs and can be copied to your Simulink model. It also provides RTI Gigalink blocks for implementing data transfer between real-time applications. In addition, the `rtimplib` also provides demo models, templates, and useful information.

The following components are available in the `rtimplib`:

Demos Contains example models. In some cases, you might want to use the demos as templates for your own model.

Templates Contains predefined template models for up to five CPUs so that every CPU has a direct connection to all other CPUs. You can modify their structures according to your needs, for example, by adding further IPC blocks or deleting unused IPC blocks.

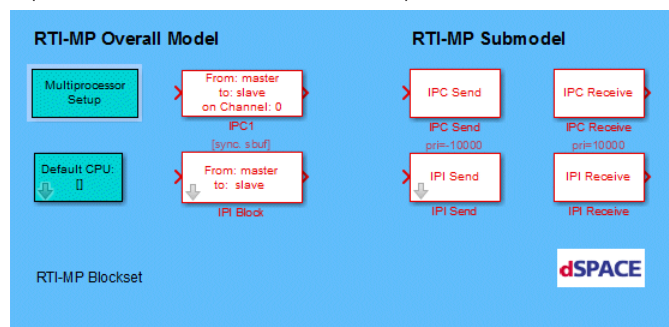
Help Displays this reference. The most important RTI-MP-related topics are described.

RTI-MP Blocks Contains all RTI-MP-specific blocks necessary to implement your multiprocessor model, refer to Overview of the RTI-MP blocks.

RTI Gigalink Blocks (optional) Contains blocks necessary to implement data transfer between different real-time applications via a Gigalink connection, refer to [RTI Gigalink Blockset Reference](#) on page 272.

Overview of the RTI-MP blocks

After you double-click the RTI-MP Blockset button in the Blocksets library, the `rtimplib` library is displayed. The RTI-MP blockset provides blocks that let you implement models for using on several processors, called multiprocessor models. The main feature of these blocks is to offer communication between the separated models for tasks and interrupts.



Hierarchy levels allowed for the RTI-MP blocks

The following table summarizes which blocks in the RTI-MP Overall Model section are allowed on the different hierarchical levels of an RTI-MP model. The Multiprocessor Setup block is mandatory.

(The blocks in the RTI-MP Submodel section are only for internal use.)

Hierarchical Level	Standard Blocks	Interrupt Blocks
Root level	Multiprocessor Setup block Default CPU block IPCx block	Function-call subsystem Hardware interrupt block Software Interrupt block Background block IPI block
Level 1	Default CPU block	Function-call subsystem Hardware interrupt block Software Interrupt block Background block (local only)
Levels 2+	(None)	Function-call subsystem Hardware interrupt block Software Interrupt block Background block (local only)
Function-call subsystem	Default CPU block IPCx block	Function-call subsystem Hardware interrupt block Software Interrupt block
Level 1 below function-call subsystem	Default CPU block	Function-call subsystem Hardware interrupt block Software Interrupt block
Levels 2+ below function-call subsystem	(None)	Function-call subsystem Hardware interrupt block Software Interrupt block

Note

Background blocks that are placed below root level can be used only locally. This means that the corresponding function-call subsystem should not contain any RTI-MP blocks.

Configuration Blocks

Introduction

The Multiprocessor Setup block and the Default CPU block are used to configure the multiprocessor Simulink model.

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

Default CPU.....	259
To assign CPU identities to unconnected blocks on the top level of the Simulink model or within triggered subsystems.	
Multiprocessor Setup.....	260
To set parameters of the multiprocessor Simulink model.	

Default CPU

Block**Purpose**

To assign CPU identities to unconnected blocks on the top level of the Simulink model or within triggered subsystems.

Description

Each Simulink block of a multiprocessor system inherits its CPU identity from a connected IPC block. The **Default CPU** block is used to assign the CPU identity to unconnected blocks in a multiprocessor system. Therefore, a **Default CPU** block can be located on the model root to assign all unconnected blocks to the specified CPU, and in any subsystem (the second hierarchical level of a model) to assign only the current subsystem to a specific CPU.

Since triggered subsystems can be computed on more than one CPU, **Default CPU** blocks can also be used in triggered subsystems. However, it is not possible to have more than one **Default CPU** block on the model root or on the top level of a (triggered) subsystem.

Note

If a multiprocessor model contains unconnected blocks but not a **Default CPU** block, RTI-MP issues a warning but no error, and automatically assigns them to the master CPU.

Adding a CPU RTI-MP determines the names of the CPUs in a model via the CPU names specified in IPCx blocks, IPI blocks or **Default CPU** blocks. Each CPU name found in such a block is listed in the **Multiprocessor Setup** dialog and gets its own CPU's page to set up the application-specific properties, such as name and step size multiple. To add a new CPU to the system, it is sufficient to specify its name in an IPCx block, IPI block or **Default CPU** block.

Dialog settings**Default CPU** Specify the name of the default CPU.**Related topics****HowTos**

[How to Define a Default CPU for Unconnected Model Parts \(RTI and RTI-MP Implementation Guide !\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\)\)](#)

Multiprocessor Setup

Block**Purpose**

The Multiprocessor Setup dialog is displayed by double-clicking the Multiprocessor Setup block. This dialog lets you:

- Set the global parameters of the multiprocessor Simulink model
- Set the CPU-specific parameters of the multiprocessor Simulink model
- Change the target multiprocessor system
- Get further information about your model and add comments

Note

This block must be on the top level of every RTI-MP model.

Tip

You can double-click the Multiprocessor Setup block found in the RTI-MP Blocks library to create a new RTI-MP model from scratch: A new Simulink model containing a Multiprocessor Setup block and an IPC block is opened. You can insert your own Simulink blocks and add, delete and change communication channels and CPUs.

Related topics

HowTos

[How to Configure Models for MP Systems \(RTI and RTI-MP Implementation Guide !\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\)\)](#)

References

IPCx.....	261
Multiprocessor Setup Dialog.....	48

Interprocessor Communication Blocks

Introduction

The Interprocessor Communication (IPC) blocks are used to implement communication channels between the CPUs of a multiprocessor system.

Where to go from here

Information in this section

IPCx.....	261
To define the parameters of a communication channel between the CPUs of a multiprocessor system.	
IPC Receive.....	265
To implement the interfaces of a communication channel for receiving data from another CPU in a multiprocessor system.	
IPC Send.....	267
To implement the interfaces of a communication channel for sending data to another CPU in a multiprocessor system.	

IPCx

Block



Purpose

To define the parameters of a communication channel between the CPUs of a multiprocessor system.

Note

Before the parameters of the corresponding channel and the IPCx blocks belonging to the current connection can be changed, the Multiprocessor Setup dialog must be closed.

Description

IPCx blocks must be located on the top level of the model or in triggered subsystems. The source and destination CPU plus the channel number are displayed on the IPCx block. IPCx blocks can be copied by drag & drop like any other Simulink block, to get more connector blocks for one communication channel or establish new communication channels and new CPUs. All the Simulink blocks of a multiprocessor system inherit their CPU identities from a connected IPCx block.

To delete an interprocessor communication (IPC) connection, just delete all the IPCx blocks of the connection.

Simulink simulation Each IPCx block contains an S-function that checks the data types of the input signals. This S-function behaves exactly like the standard Simulink Mux and Demux blocks, except in buffering or unbuffering mode. In buffering or unbuffering mode, it emulates the buffering and unbuffering of signals. Therefore, Simulink simulation can always be performed with the same results as a multiprocessor real-time simulation.

Adding a CPU RTI-MP determines the names of the CPUs in a model via the CPU names specified in IPCx blocks, IPI blocks or Default CPU blocks. Each CPU name found in such a block is listed in the Multiprocessor Setup dialog and gets its own CPU's page to set up the application-specific properties, such as name and step size multiple. To add a new CPU to the system, it is sufficient to specify its name in an IPCx block, IPI block or Default CPU block.

For details on interprocessor communication (IPC) connections, refer to [Interprocessor Communication \(RTI and RTI-MP Implementation Guide\)](#).

Data buffering and unbuffering

You can also use the IPCx block to achieve data buffering or unbuffering. You must then specify a value greater than 1 in the **Buffered samples** edit field of the IPCx block. The IPC buffer or IPC unbuffer mode is selected automatically for an IPC connection as follows:

- Step size of source CPU < step size of destination CPU

The IPC connection is operated in IPC buffer mode. For example, if the step size multiples are 1 for the source CPU and 10 for the destination CPU, and the number of buffered samples is 10, a vector of 10 input values is collected and passed to the destination CPU at every 10th simulation step.

- Step size of source CPU > step size of destination CPU

The IPC connection is operated in IPC unbuffer mode. For example, if the step size multiples are 10 for the source CPU and 1 for the destination CPU, and the number of buffered samples is 10, a vector of 10 values is passed from the source to the destination CPU at every 10th simulation step. In each basic simulation step, the destination CPU uses one of the ten values as the model input.

You can use the **Buffered samples** edit field of the **Communication Channel Setup** dialog to define how many signal samples are to be buffered or unbuffered on an IPC connection.

Note

Make sure that you transfer signals of the double data type and use the swinging buffer protocol. The IPCx block does not support other data types or the virtual shared memory protocol if operated in the IPC buffer or IPC unbuffer mode.

For an instructive example of data buffering/unbuffering, refer to [Example of IPC buffer/unbuffer mode \(RTI and RTI-MP Implementation Guide !\[\]\(8d0f0e0fe25b320c33272c52aec1fbca_img.jpg\)](#)).

Communication channel frame

Connection: Source CPU Lets you specify the name of the source CPU (origin of the IPCx block). CPU names are restricted to eight alphanumeric characters.

To add another CPU to the multiprocessor model, simply enter a new CPU name. The parameters of the new CPU, such as application name and solver, can be entered the next time the **Multiprocessor Setup** dialog is opened.

Connection: Destination CPU Lets you specify the name of the target CPU. CPU names are restricted to eight alphanumeric characters.

To add another CPU to the multiprocessor model, simply enter a new CPU name. The parameters of the new CPU, such as application name and solver, can be entered the next time the **Multiprocessor Setup** dialog is opened.

Connection: Channel Lets you specify the channel number.

You can select any non-negative integer number as the channel number.

Step size multiple Specify the step sizes of the communication channel as integer multiples of the basic real-time step size. The step sizes have to be a multiple of the step size of the CPU. The **Step size multiple** must be specified independently for the Source and the Destination CPU.

Protocol From the drop-down list, choose one of the following protocols:

- [Swinging Buffer Protocol \(RTI and RTI-MP Implementation Guide !\[\]\(307ad7be8dd8053938b04a332782a8a1_img.jpg\)](#)) (synchronized or unsynchronized)
- [Virtual Shared Memory Protocol \(RTI and RTI-MP Implementation Guide !\[\]\(4ddfca81566d78050c33f6ae0caaf3e9_img.jpg\)](#))

For details on the protocols, refer to [Interprocessor Communication \(RTI and RTI-MP Implementation Guide\)](#).

Number of signals The number of signals to be transferred via a communication channel is displayed. It results from the values specified in the **Width** fields of the **Member IPC** blocks belonging to the current communication channel.

Buffered samples Specify the number of buffered signal samples to be sent per cycle.

Normally, the number of buffered samples is equal to 1, which means no buffering/unbuffering is performed. If the number of buffered samples is greater than 1, the connection operates in buffering or unbuffering mode, depending on the **Step** size multiple of the **Source CPU** and **Destination CPU** of the current connection.

For details, refer to [Communication with a Sample Time Transition \(RTI and RTI-MP Implementation Guide\)](#).

Note

The communication protocol must be set to **Synchronized Swinging Buffer** or **Unsynchronized Swinging Buffer** if you want to operate an **IPCx** block in buffering/unbuffering mode. The entry for the **Data type** for this **IPC** connection must be **double**.

Member IPC blocks frame

Show Block Highlights the current **IPCx** block in the model.

Name The Simulink names of all **IPCx** blocks belonging to the current connection are displayed. The **IPCx** block that is currently highlighted in the model, is displayed in bold letters.

Width Enter the signal width of the corresponding **IPCx** block.

The **Number of Signals** of the current communication channel is automatically adjusted to the entries in the **Width** windows of the **IPCx** blocks.

If a vector of N signals is to be transmitted by means of an **IPCx** block, its width must be given in brackets as $[n_1 \ n_2 \ \dots \ n_N]$, where $n_1 \ \dots \ n_N$ represent the widths of the signal vectors of the n^{th} **IPCx** block. The total number of signals to be sent over the corresponding block is the sum of all elements of the width vector. For example, if two signal vectors with 3 and 4 elements are to be transmitted, the signal width is $[3 \ 4]$. A specification of $[3]$ corresponds to a single signal vector of 3 elements, whereas 3 (without brackets) represents 3 scalar signals. Square brackets always indicate vector signals. The number of inports and outports of the **IPCx** block is adjusted according to the number of signals entered as **Width**.

Data Type Lets you select a data type that is supported by MATLAB:

- double
- single
- int8
- uint8

- int16
- uint16
- int32
- uint32
- Boolean

The default data type is double.

Note

- If you want an IPC connection to operate in buffering or unbuffering mode, its data type must be double. The Protocol setting in the related IPCx block must be set to *Synchronized* or *Unsynchronized Swinging Buffer*.
- If the data type of an IPCx block and the input signal(s) do not match, an error message is sent by Simulink. To avoid this, either adapt the data type of the IPCx block to the input or provide an input signal of the desired data type.

Change connection Opens the Change Connection dialog (see [Change Connection Dialog](#) on page 47).

Related topics

Basics

[Interprocessor Communication \(RTI and RTI-MP Implementation Guide !\[\]\(17413706fd4997a1a4bdf85c6864eee1_img.jpg\)\)](#)
[Interprocessor Communication Using IPC Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(f419710cbe076aa30a9c6c031b5cbe84_img.jpg\)\)](#)
[Swinging Buffer Protocol \(RTI and RTI-MP Implementation Guide !\[\]\(2726020a4107bdc9042b257034f90eb3_img.jpg\)\)](#)
[Virtual Shared Memory Protocol \(RTI and RTI-MP Implementation Guide !\[\]\(9459655bf14a84f4d775e8d814cca8c9_img.jpg\)\)](#)

References

[Change Connection Dialog](#)..... 47

IPC Receive

Block



Purpose

To implement the interfaces of a communication channel for receiving data from another CPU in a multiprocessor system.

Description

IPC Receive blocks are automatically added to a CPU's submodel when generating or building a multiprocessor model. If you do not use the automatic model separation (see [CPU's Page \(Multiprocessor Setup Dialog\) – Always generate a new submodel](#)), you have to implement the communication interface in your submodel manually by adding and configuring IPC Receive blocks.

Tip

In this case, you can also modify the Simulink block priority. To change the Simulink block priority for IPC blocks automatically generated by model separation, you can use the `PostSeparation` hook function.

The MP main model has to contain a related IPCx block.

Dialog settings

Sample time Lets you specify a sample time. If the block is placed in a function-call subsystem, you must always use an inherited sample time (-1).

Signal Width Lets you specify the signal width in the range 1 ... 1024.

Data Type Lets you select a data type that is supported by MATLAB:

- double
- single
- int8
- uint8
- int16
- uint16
- int32
- uint32
- Boolean

The default data type is double.

Related topics**Basics**

[Interprocessor Communication \(RTI and RTI-MP Implementation Guide !\[\]\(f60b7a900783ac3fd531bfd9c111be6d_img.jpg\)\)](#)

References

IPC Send	267
IPCx	261

IPC Send

Block



Purpose

To implement the interfaces of a communication channel for sending data to another CPU in a multiprocessor system.

Description

IPC Send blocks are automatically added to a CPU's submodel when generating and building a multiprocessor model. If you do not use the automatic model separation (see [CPU's Page \(Multiprocessor Setup Dialog\) – Always generate a new submodel](#)), you have to implement the communication interface in your submodel manually by adding and configuring IPC Send blocks. The MP main model has to contain a related IPCx block.

Dialog settings

Sample time Lets you specify a sample time. If the block is placed in a function-call subsystem, you must always use an inherited sample time (-1).

Signal Width Lets you specify the signal width in the range 1 ... 1024.

Data Type Lets you select a data type that is supported by MATLAB:

- double
- single
- int8
- uint8
- int16
- uint16
- int32
- uint32
- Boolean

The default data type is double.

Related topics

Basics

[Interprocessor Communication \(RTI and RTI-MP Implementation Guide !\[\]\(b64b40baaee5acddc1eab8538ba84754_img.jpg\)](#))

[IPC Receive..... 265](#)

References

[IPCx..... 261](#)

Interprocessor Interrupt Blocks

Introduction The Interprocessor Interrupt (IPI) blocks are used to implement interrupts in a multiprocessor model.

Where to go from here	Information in this section
	IPI.....268 To specify the interrupt source CPU and the target CPUs in multiprocessor systems.
	IPI Receive.....269 To implement the interfaces of an interprocessor interrupt for receiving interrupts on the target CPU in a multiprocessor system.
	IPI Send.....270 To implement the interfaces of an interprocessor interrupt for sending interrupts from the source CPU to another target CPU in the multiprocessor system.

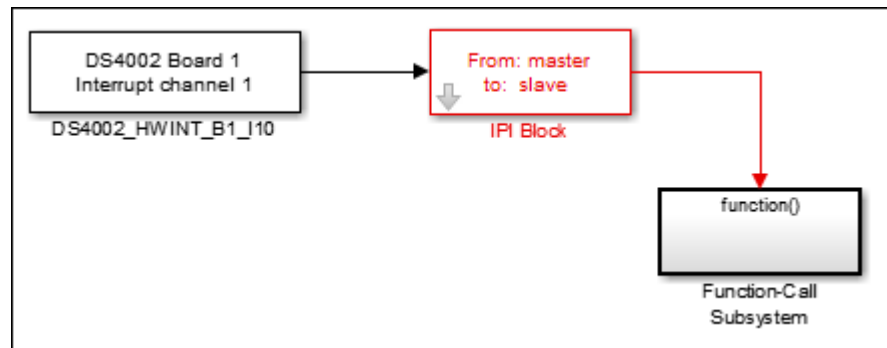
IPI



Purpose To specify the interrupt source CPU and the target CPUs in multiprocessor systems.

- Note**
- IPI blocks must be located on the top level of the RTI-MP model.
 - For local interrupts, i.e., if the source CPU and the destination CPU are identical, no IPI block is required.

Description In multiprocessor systems hardware and software interrupts can trigger tasks on more than one CPU. Therefore, Hardware and Software interrupt blocks from the specific RTI board library can only be used together with IPI blocks. These IPI blocks must be placed between the Hardware or Software interrupt block and the triggered subsystem. See the illustration below.



Adding a CPU RTI-MP determines the names of the CPUs in a model via the CPU names specified in IPCx blocks, IPI blocks or Default CPU blocks. Each CPU name found in such a block is listed in the Multiprocessor Setup dialog and gets its own CPU's page to set up the application-specific properties, such as name and step size multiple. To add a new CPU to the system, it is sufficient to specify its name in an IPCx block, IPI block or Default CPU block.

Dialog settings

Maximum number of Target CPUs Lets you specify the number of involved target CPUs in the range 1 ... 12.

Source CPU Lets you specify the source CPU of the interprocessor interrupt.

Target CPU 1 ... Target CPU x Lets you specify the target CPU(s) of the interprocessor interrupt.

Related topics

HowTos

[How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide 📖\)](#)

IPI Receive

Block



Purpose

To implement the interfaces of an interprocessor interrupt for receiving interrupts on the target CPU in a multiprocessor system.

Description

IPI Receive blocks are automatically added to a CPU's submodel when generating and building a multiprocessor model. If you do not use the automatic model separation (see [CPU's Page \(Multiprocessor Setup Dialog\) – Always generate a new submodel](#)), you have to implement the interprocessor interrupts in your submodel manually by adding and configuring IPI Receive blocks.

The MP main model has to contain a related IPI block.

This block makes interrupts defined by IPI Send blocks available as task trigger sources.

Related topics**Basics**

[IPI Send..... 270](#)

HowTos

[How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(73002692dd5e7a64e60946be3158e719_img.jpg\)\)](#)

References

[IPI..... 268](#)

IPI Send

Block**Purpose**

To implement the interfaces of an interprocessor interrupt for sending interrupts from the source CPU to another target CPU in the multiprocessor system.

Description

IPI Send blocks are automatically added to a CPU's submodel when generating and building a multiprocessor model. If you do not use the automatic model separation (see [CPU's Page \(Multiprocessor Setup Dialog\) – Always generate a new submodel](#)), you have to implement the interprocessor interrupts in your submodel manually by adding and configuring IPI Send blocks.

The MP main model has to contain a related IPI block.

This block makes the interrupt that is connected to the block's inport available to other CPUs in the multiprocessor system.

Dialog settings

Enable local interrupt port Lets you enable an output that provides the interrupt signal for local use.

Related topics**Basics**

[IPI Receive..... 269](#)

HowTos

[How to Implement Interprocessor Interrupts \(RTI and RTI-MP Implementation Guide !\[\]\(2b376d1a92330ab09dad2665d2f89bf5_img.jpg\)\)](#)

References

[IPI..... 268](#)

RTI Gigalink Blockset Reference

Where to go from here

Information in this section

Overview of the RTI Gigalink Blockset.....	272
Gigalink_Interrupt.....	274
To receive interrupts and synchronize two applications connected via Gigalink.	
Gigalink_Receive.....	275
To receive signals from a specified Gigalink channel.	
Gigalink_Send.....	277
To send signals to a specified Gigalink channel.	
Gigalink_Status.....	279
To return the connection status of a specified Gigalink.	

Overview of the RTI Gigalink Blockset

Introduction

The RTI Gigalink Blockset provides blocks for implementing data transfer between real-time applications using DS1006 and/or DS1007, and SCALEXIO platforms. With a SCALEXIO platform, synchronization of timer tasks of a DS1006/DS1007 application and blocking communication is additionally possible.

Supported connection scenarios

With the RTI Gigalink blockset you can implement the following connection scenarios.

Connection Scenarios	From ...	To ...
Data Connection Without Task Synchronization		
The non-blocking receive mode is applicable. In this connection scenarios the timer tasks between communicating applications are not synchronized. Between non-synchronized tasks a blocking communication will lead to task overruns.	DS1006	DS1006
	DS1006	DS1007
	DS1006	SCALEXIO
	DS1007	DS1007
	DS1007	DS1006
	DS1007	SCALEXIO
	SCALEXIO	DS1006
	SCALEXIO	DS1007
	SCALEXIO	SCALEXIO ¹⁾

Connection Scenarios	From ...	To ...
Data Connection With Synchronization of Timer Tasks		
The timer tasks in the DS1006/DS1007 application are synchronized by a timer event sent by the connected SCALEXIO system. The blocking receive mode is applicable.	SCALEXIO SCALEXIO	DS1006 DS1007

¹⁾ RTI and RTI Gigalink Blockset are not used in this scenario.

To implement a synchronized communication between a SCALEXIO application and a DS1006 or DS1007 application, the following conditions must be met:

- The SCALEXIO application must send a synchronization interrupt via Gigalink.
- The DS1006 or DS1007 application must receive this synchronization interrupt with the `Gigalink_Interrupt` block.
- The `Gigalink_Interrupt` block must be connected to a Timer Task Assignment block.

Note

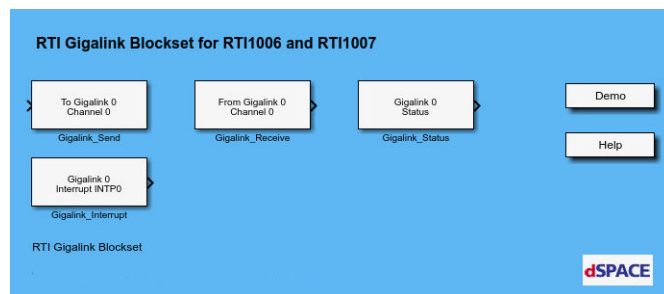
- By default, the data connections are non-blocking. You can configure the connection to receive data in blocking mode. However, this requires task synchronization between the data sender and the data receiver that is only supported for connections between a SCALEXIO system and a DS1006 or DS1007 and for timer tasks.
- If you use the Timer Task Assignment block, you must ensure that the simulation step size of the model and the step size of the external trigger are identical.

For further information about using Gigalink connections with a SCALEXIO system, refer to [Building the Signal Chain for Gigalink Communication \(ConfigurationDesk Real-Time Implementation Guide !\[\]\(003082e50e3009141f59bd5df831749f_img.jpg\)](#)).

Library access

After you double-click the RTI Gigalink Blocks button in the `rtimp` library, the `rtiglbslib` library opens.

The RTI Gigalink Blockset requires the RTI-MP license, but it can be also used without an installed RTI-MP Blockset. You can therefore access the RTI Gigalink Blockset also via the Simulink Library Browser, or by entering `rtiglbs` in the MATLAB Command Window.



Gigalink_Interrupt

Block



Purpose

To receive interrupts and synchronize two applications connected via Gigalink.

Description

The Gigalink_Interrupt block is used to receive interrupts from a SCALEXIO application connected via Gigalink. This allows you to synchronize the timer tasks of a real-time application running on a DS1006 or DS1007 with those of a real-time application running on a SCALEXIO system. The SCALEXIO application provides the overall timer source.

The connection between the Gigalink_Interrupt block of a DS1006/DS1007 application and the SCALEXIO application is established by specifying the same Gigalink interrupt number. For further information, refer to [Building the Signal Chain for Gigalink Communication \(ConfigurationDesk Real-Time Implementation Guide\)](#).

The Gigalink_Interrupt block has to be connected to a Timer Task Assignment block. This establishes the synchronization of the timer tasks between the connected applications.

It is not checked, whether both applications to be synchronized are loaded or the Gigalinks are physically connected.

Note

- Only one Gigalink_Interrupt block can be used per RTI/RTI-MP application.
- The Gigalink_Interrupt block can only receive interrupts. Sending interrupts is not supported.
- The Gigalink Blockset does not support MicroLabBox.

Dialog settings

Gigalink number Lets you specify the number of the Gigalink from which the interrupt is to be received in the range 0 ... 3.

Interrupt number Lets you specify the number of the interrupt to be received in the range INTP0 ... INTP7.

Related topics**Basics**

[Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\)\)](#)

HowTos

[How to Implement Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(a03a7eb2f4046e1d3c76772003e549ea_img.jpg\)\)](#)

References

Gigalink_Receive.....	275
Gigalink_Send.....	277
RTI Gigalink Blockset Reference.....	272
Timer Task Assignment Block.....	246

Gigalink_Receive

Block**Purpose**

To receive signals from a specified Gigalink channel.

Description

A Gigalink_Receive block can be inserted in a model using a DS1006 or DS1007 platform. It receives signals from its counterpart, the Gigalink_Send block, via a Gigalink connection. Signals can be transmitted to processor boards of the same type or of different types. Thus, it is possible to connect DS1006, DS1007 and SCALEXIO applications.

If the specified Gigalink is not connected or the opposite application is not loaded, no data is received by the Gigalink.

Note

When implementing Gigalink connections, note the following limitations:

- All processor boards must be equipped with Gigalink modules.
 - Consistency of data transfer is ensured by the swinging buffer protocol in blocking or non-blocking mode. For details, refer to [Swinging Buffer Protocol \(RTI and RTI-MP Implementation Guide !\[\]\(cdf2842d82858164c68c92720a337fb9_img.jpg\)](#)).
- Blocking communication should be combined with the synchronization of timer tasks. This use case is only supported by SCALEXIO to DS1006/DS1007 connections.
- Signals must be scalar or vectorial and of **double** data type. Otherwise an error message appears during model initialization.
 - At most 1024 signals of **double** data type can be transmitted via a Gigalink channel.
 - Each combination of Gigalink number and channel number can be used for a block only once. Otherwise an error message appears during the build process.
 - If Gigalink_Send and Gigalink_Receive blocks and RTI-MP blocks for interprocessor communication use the same Gigalink number, they cannot use the same channel numbers. The RTI Gigalink Blockset and RTI-MP do not check the channel numbers and the related Gigalink number. You must ensure that the channel numbers used for RTI-MP interprocessor communication are not the same channel numbers used by the RTI Gigalink Blockset Send and Receive blocks.
 - The RTI Gigalink Blockset does not support MicroLabBox.

Dialog settings

Gigalink number Lets you specify the Gigalink port number that is used for data transmission in the range 0 ... 3.

Channel number Lets you specify the Gigalink channel number that is used for data transmission in the range 0 ... 7.

Signal width Lets you specify the number of elements to be received in the range 1 ... 1024. The correct range is checked by the underlying Block Properties dialog.

Note

The Channel number and the Signal width must be equal for a Gigalink block pair (Send block and Receive block).

Protocol Lets you specify the receiving mode of the block.

Protocol	Description
Blocking	The swinging buffer for receiving data will be synchronized. When reading data in blocking mode, the receiver waits as long as the required amount of data is read or a task overrun occurs.
Non-Blocking	The swinging buffer for receiving data will not be synchronized.

Sample time Lets you specify the sample time of the block.

Related topics

Basics

[Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(0aff635c4179ba9e710b00f4b01d3b20_img.jpg\)\)](#)

HowTos

[How to Implement Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(0b5e7e25e8775f7e7e80906ada4f0021_img.jpg\)\)](#)

References

Gigalink_Send.....	277
Gigalink_Status.....	279
RTI Gigalink Blockset Reference.....	272

Gigalink_Send

Block



Purpose

To send signals to a specified Gigalink channel.

Description

A Gigalink_Send block can be inserted in a model using a DS1006 or DS1007 board. It sends signals to its counterpart, the Gigalink_Receive block, via a Gigalink connection. Signals can be transmitted to processor boards of the same type or of different types. Thus, it is possible to connect DS1006, DS1007 and SCALEXIO applications.

If the specified Gigalink is not connected or the opposite application is not loaded, no data is sent to the Gigalink.

Note

When implementing Gigalink connections, note the following limitations:

- All processor boards must be equipped with Gigalink modules.
 - Consistency of data transfer is ensured by the swinging buffer protocol in blocking or non-blocking mode. For details, refer to [Swinging Buffer Protocol \(RTI and RTI-MP Implementation Guide !\[\]\(065aacad479feea1b3f501fa02b79a7a_img.jpg\)](#)).
- Blocking communication should be combined with the synchronization of timer tasks. This use case is only supported by SCALEXIO to DS1006/DS1007 connections.
- Signals must be scalar or vectorial and of **double** data type. Otherwise an error message appears during model initialization.
 - At most 1024 signals of **double** data type can be transmitted via a Gigalink channel.
 - Each combination of Gigalink number and channel number can be used for a block only once. Otherwise an error message appears during the build process.
 - If Gigalink_Send and Gigalink_Receive blocks and RTI-MP blocks for interprocessor communication use the same Gigalink number, they cannot use the same channel numbers. The RTI Gigalink Blockset and RTI-MP do not check the channel numbers and the related Gigalink number. You must ensure that the channel numbers used for RTI-MP interprocessor communication are not the same channel numbers used by the RTI Gigalink Blockset Send and Receive blocks.
 - The RTI Gigalink Blockset does not support MicroLabBox.

Dialog settings

Gigalink number Lets you specify the Gigalink port number that is used for data transmission in the range 0 ... 3.

Channel number Lets you specify the Gigalink channel number that is used for data transmission in the range 0 ... 7.

Signal width Lets you specify the number of elements to be sent in the range 1 ... 1024. The correct range is checked by the underlying Block Properties dialog.

Note

The Channel number and the Signal width must be equal for a Gigalink block pair (Send block and Receive block).

Sample time Lets you specify the sample time of the block.

Related topics**Basics**

[Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(feabb98897b440bc8695a03336a6e2df_img.jpg\)\)](#)

HowTos

[How to Implement Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\)\)](#)

References

[Gigalink_Receive..... 275](#)
[Gigalink_Status..... 279](#)
[RTI Gigalink Blockset Reference..... 272](#)

Gigalink_Status

Block**Purpose**

To return the connection status of a specified Gigalink.

Description

The `Gigalink_Status` block can be inserted in a model containing a `Gigalink_Send` block or a `Gigalink_Receive` block. It checks, whether data can be transmitted. The block's output can be the following:

Output	Description
1	The specified Gigalink channel is ready for data transmission. The <code>Gigalink_Receive</code> block using this Gigalink channel has a counterpart that sends data.
0	The specified Gigalink channel is not ready for data transmission. The <code>Gigalink_Receive</code> block using this Gigalink channel has no counterpart that sends data. The <code>Gigalink_Receive</code> block returns an old value from the buffer or 0 if the buffer is empty.

Note

- The state will be updated only while the related task is running. This behavior might be different, if synchronized Gigalink communication is used.
- The RTI Gigalink Blockset does not support MicroLabBox.

Dialog settings

Gigalink number Lets you specify the Gigalink number whose synchronization status is to be checked in the range 0 ... 3.

Related topics

Basics

[Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(17acf1afa8cdf0b67c53d4865a5ed469_img.jpg\)](#))

HowTos

[How to Implement Interprocessor Communication Using Gigalink Blocks \(RTI and RTI-MP Implementation Guide !\[\]\(f95dab70c751fda7d824b8b03650f7aa_img.jpg\)](#))

References

Gigalink_Interrupt.....	274
Gigalink_Receive.....	275
Gigalink_Send.....	277
RTI Gigalink Blockset Reference.....	272

Symbols

_author 145
 _description 146
 _floating_point_type 146
 _gendate 147
 _genname 147
 _genversion 147
 _integer_type 148
 _model 148

A

accessing custom variables 172
 addr 156
 alias 158
 array-incr 159

B

Background block 234
 bitmask 159
 block 160
 buffered samples 264
 buffering 262
 build options 64

C

category
 rti general build options 33
 rti load options 34
 rti variable description file options 35
 Change Connection dialog (RTI-MP) 47
 clean.bat file 106
 Code Generation dialog (RTI) 25
 Code Generation dialog (RTI-MP) 80
 coefficient of initial simsteps increasement 43, 62
 coefficient of number of increased simsteps 43, 63
 comments 142
 Common Program Data folder 11
 Communication Channels display (RTI-MP) 47
 compiler options
 -DRTIMP_GL_NO_TIMEOUT 65
 Configure Initial Step Size Dialog (RTI) 42
 Configure Initial Step Size Dialog (RTI-MP) 62
 CPU identity 259
 currentTime 90

D

data buffering 262
 Data Capture block 249
 data set storage 34
 data type specifications 131
 default 160
 Default CPU block 259
 Demos
 RTI-MP library 257
 desc 161

Diagnostics dialog (RTI) 19
 Diagnostics dialog (RTI-MP) 75
 Documents folder 11
 download.bat file 106
 ds_trc_multiplelabeloccurrence 184
 dsextras 248
 DSFINISH.M file 115
 DSPOSTSTARTUP.M file 114
 DSSTARTUP.M file 114
 DsVdOmit
 TRC exclusion 37, 69

E

EESPort real-time variables 131
 endstruct 149
 enhanced TRC support for bus objects 128
 enum 150
 error file 142
 errorNumber 91
 Extras library 248
 Data Capture block 249
 Ramp Generator for Encoder Index Search block 251
 simState READ block 252
 simState SET block 253
 TRC Exclusion block 254

F

file overview 104
 file specifics for model referencing 110
 files
 clean.bat 106
 download.bat 106
 DSFINISH.M 115
 DSPOSTSTARTUP.M 114
 DSSTARTUP.M 114
 IPC.H 105
 linker command 122
 LK 122
 MAP 108
 MAPX 108
 PPC 107
 RTA 107
 RTI.C 105
 RTI.MK 106
 RTI.PRJ 106
 rti1xxx_template_rtimphook.m 116
 SDF 107
 SIMENG.C 105
 STARTUP.M 113
 TH.C 105
 TRC 108, 123
 TRT 108
 user makefile 119
 user system description file 123
 user variable description file 132
 User-Code 118
 USR.C 118
 USR.MK 119
 USR.SDF 123

USR.TRC 132
 variable description file 123
 X86 107
 finalTime 92
 flags 161
 Function-Call Subsystem block 237

G

Gigalink_Interrupt block 274
 Gigalink_Receive block 275
 Gigalink_Send block 277
 Gigalink_Status block 279
 group 150
 grouping 139

H

Hardware Implementation dialog (RTI) 21
 Hardware Implementation dialog (RTI-MP) 76
 Hardware Interrupt block 235
 Help
 RTI-MP library 257

I

interrupt handling
 Function-Call Subsystem block 237
 Hardware Interrupt block 235
 Software Interrupt block 238
 Timer Interrupt block 239
 IPC Receive block 265
 IPC Send block 267
 IPC.H file 105
 IPCx block 261
 IPI block 268
 IPI Receive block 269
 IPI Send block 270

K

keyword
 _author 145
 _description 146
 _gendate 147
 _genname 147
 _genversion 147
 _integer_type 148
 _model 148
 endgroup 149
 endstruct 149
 enum 150
 floating_point_type 146
 group 150
 struct 153
 typedef 154
 keywords 144

L

linker command file 122
 LK file 122
 Local Program Data folder 11

M

MAP file 108
 MAPX file 108
 Math and Data Types dialog
 RTI 19, 74
 Math and Data Types dialog (RTI) 19
 Math and Data Types dialog (RTI-MP) 74
 model configuration parameters
 Code Generation dialog (RTI) 25
 Code Generation dialog (RTI-MP) 80
 Diagnostics dialog (RTI) 19
 Diagnostics dialog (RTI-MP) 75
 Hardware Implementation dialog (RTI) 21
 Hardware Implementation dialog (RTI-MP) 76
 Math and Data Types dialog (RTI) 19
 Math and Data Types dialog (RTI-MP) 74
 Model Parameter Configuration dialog (RTI) 38
 Model Parameter Configuration dialog (RTI-MP) 85
 Model Referencing dialog (RTI) 22
 Model Referencing dialog (RTI-MP) 78
 RTI 14
 RTI-MP 73
 Simulation Target dialog (RTI) 24
 Simulation Target dialog (RTI-MP) 80
 Solver dialog (RTI) 15
 Model Parameter Configuration dialog (RTI) 38
 Model Parameter Configuration dialog (RTI-MP) 85
 Model Referencing dialog (RTI) 22
 Model Referencing dialog (RTI-MP) 78
 model verification block enabling 20, 75
 modelStepSize 92
 monitoring variables for electrical error simulation (EESPort) 131
 Multiprocessor Setup block 260
 Multiprocessor Setup dialog 48
 Advanced page 57
 Build Options page 64
 CPU Options dialog 64
 CPU's page 53
 Documentation page 59
 Main page 49
 Multiprocessor Topology Setup dialog 71
 Rename CPU dialog 70
 Variable Description File Options page 67
 multiprocessor topology setup 71

N

name of member IPC block 264
 number of signals 264

O

offs 162
 origin 163
 overrunCheckType 93
 overrunCount 94
 overrunQueueCount 95

P

PPC file 107
 priority 96
 properties
 addr 156
 alias 158
 array-incr 159
 bitmask 159
 block 160
 default 160
 desc 161
 flags 161
 offs 162
 origin 163
 range 163
 refelem 163
 refgroup 165
 refvar 166
 scale 166
 scaleback 167
 type 167
 unit 171
 value 171
 protocol for RTI-MP 263

R

Ramp Generator for Encoder Index Search
 block 251
 range 163
 Real-Time Interface for Multiprocessor Systems 231
 refelem 163
 refgroup 165
 refvar 166
 renaming CPU 70
 reserved words 144
 RTA file 107
 RTI commands
 ds_trc_multiplelabeloccurrence 184
 rti_build2 185
 rti_mdldcleanup 189
 rti_optionget 189
 rti_optionset 191
 rti_sdfmerge 193
 rti1xxx 185
 rtimp_blktargetcpunameget 193
 rtimp_build2 194
 rtimp_targetswitch 197
 rtiver 198
 set_rti 198
 rti general build options category 33
 RTI Gigalink Blocks
 RTI-MP library 257
 RTI Gigalink Blockset 272
 Gigalink_Interrupt block 274
 Gigalink_Receive block 275
 Gigalink_Send block 277
 Gigalink_Status block 279
 rti load options category 34
 RTI Task Configuration dialog (RTI) 39

RTI Task Configuration dialog (RTI-MP) 59
 rti variable description file options category 35
 RTI.C file 105
 RTI.MK file 106
 RTI.PRJ file 106
 rti_build2 185
 rti_mdldcleanup 189
 rti_optionget 189
 rti_optionset 191
 rti_sdfmerge 193
 rti1006 185
 rti1104 185
 rti1401 185
 rti1xxx 185
 rti1xxx_template_rtimphook.m file 116
 RTI-MP Blocks
 RTI-MP library 257
 RTI-MP Blockset 256
 configuration blocks 258
 Default CPU block 259
 interprocessor communication blocks 261
 interprocessor interrupt blocks 268
 IPC Receive block 265
 IPC Send block 267
 IPCx block 261
 IPI block 268
 IPI Receive block 269
 IPI Send block 270
 Multiprocessor Setup block 260
 RTI-MP library 231
 Demos 257
 Help 257
 RTI Gigalink Blocks 257
 RTI Gigalink Blockset 272
 RTI-MP Blocks 257
 Templates 257
 RTI-MP Overall Model 256
 RTI-MP Submodel 256
 rtimp_blktargetcpunameget 193
 rtimp_build2 194
 RTIMP_GL_NO_TIMEOUT 65
 rtimp_targetswitch 197
 rtimplib 231, 256
 rtitasklib 232
 rtiver 198

S

sampleTime 97
 sampling_period[host_service_index] 151
 scale 166
 scaleback 167
 SDF file 107
 set_rti 198
 Signal Properties dialog (RTI) 44
 Signal Properties dialog (RTI-MP) 86
 signal width of an IPC block 264
 SIMENG.C file 105
 simState 97
 simState READ block 252
 simState SET block 253
 simulation state 97

- Simulation Target dialog
 - RTI 24
 - RTI-MP 80
- Simulation Target dialog (RTI) 24
- Simulation Target dialog (RTI-MP) 80
- Software Interrupt block 238
- Solver dialog (RTI) 15
- STARTUP.M file 113
- state 98
 - simState 97
 - task 98
- struct 153
- SubArray 128
- subscheduling of tasks 238

T

- task
 - state 98
- task configuration API
 - classes 202
 - Commit 208
 - GetAllTasks 209
 - GetGroupTasks 210
 - GetModelTasks 211
 - GetSubModelNames 212
 - GetSubModelTaskManager 213
 - GetTaskGroupNames 213
 - GetTaskGroupPriority 214
 - GetTaskNamesByGroup 215
 - GetTaskOverrunBehavior 217
 - GetTaskOverrunQueueLength 218
 - GetTaskPriority 219
 - GetTaskSampleTime 220
 - IsTaskConfigurationChanged 221
 - MainModelTaskManager 203
 - methods 207
 - SetOverrunQueueLength 222
 - SetTaskGroupPriority 225
 - SetTaskOverrunBehavior 226
 - SetTaskPriority 224
 - SubModelTaskManager 205
 - TaskManager 202
 - TurnInfoMessageOff 228
 - TurnInfoMessageOn 228
- Task Configuration Blockset 232, 235
 - Background block 234
 - Function-Call Subsystem block 237
 - Software Interrupt block 238
 - Timer Interrupt block 239
- task groups
 - RTI 41
 - RTI-MP 61
- task overrun 40, 60, 65, 93
- TaskLib 232
- Templates
 - RTI-MP library 257
- TH.C file 105
- Timer Interrupt block 239
- Timer Task Assignment block 246
- Timetable Start block 244
- Timetable Task block 245

- Time-Trigger Set block 242
- Time-Triggered Task block 243
- Time-Triggered Task Blockset 232
 - Timer Task Assignment block 246
- Timetable Start block 244
- Timetable Task block 245
- Time-Trigger Set block 242
- Time-Triggered Task block 243
- TRC exclusion
 - DsVdOmit 37, 69
- TRC Exclusion block 254
- TRC file 108, 123
 - comments 142
 - endgroup keyword 149
 - endstruct keyword 149
 - error file 142
 - example 138
 - group keyword 150
 - grouping 139
 - keywords 144
 - principles 139
 - struct keyword 153
 - syntax 138
 - variable names 141
- TRC file generated by RTI 175
- TRT file 108
- tunable parameters 38, 85
- turnaroundTime 100
- type 167
- typedef 154

U

- unbuffering 262
- unit 171
- user makefile 119
- user system description file 123
- user variable description file 132
- User-Code file 118
- USR.C file 118
- USR.MK file 119
- USR.SDF file 123
- USR.TRC file 132
- usr_background 118
- usr_initialize 118
- usr_input 118
- usr_output 118
- usr_sample_input 118
- usr_terminate 118

V

- value 171
- variable and group properties 154
- variable description file 123
- variable description file groups 125
- variable description file options 67
- variable names 141
- variables 89

X

- X86 file 107

