Real-Time Workshop Release Notes

The "Real-Time Workshop 5.2 Release Notes" on page 1-1 describe the changes introduced in the latest version of the Real-Time Workshop. The following topics are discussed in these Release Notes:

- "Changes from the Previous Release" on page 1-2
- "Major Bug Fixes" on page 1-3
- "Upgrading from an Earlier Release" on page 1-4
- "Known Software and Documentation Problems" on page 1-5

If you are upgrading from a release earlier than Release 13SP2, you should also see:

- "Real-Time Workshop 5.1.1 Release Notes" on page 2-1
- "Real-Time Workshop 5.1 Release Notes" on page 3-1
- "Real-Time Workshop 5.0.1 Release Notes" on page 4-1
- "Real-Time Workshop 5.0 Release Notes" on page 5-1
- "Real-Time Workshop 4.1 Release Notes" on page 6-1
- "Real-Time Workshop 4.0 Release Notes" on page 7-1

Printing the Release Notes

If you would like to print the Release Notes, you can link to a PDF version.

Contents

1 Real-Time Workshop 5.2 Release Notes

5 Real-Time Workshop 5.0 Release Notes

Real-Time Workshop 5.2 Release Notes

Changes from the Previous Release

The behavior of source block dialog boxes has changed. Note that opening a dialog box for a source block causes Simulink to pause. While Simulink is paused, you can edit the parameter values. You must close the dialog box to have the changes take effect and allow Simulink to continue.

Major Bug Fixes

Real-Time Workshop 5.2 includes important bug fixes made since Version 5.1.1.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Version 5.1, then you should also see "Major Bug Fixes" on page 3-3 of the Real-Time Workshop 5.1 Release Notes.

Upgrading from an Earlier Release

Note If you are upgrading from a version earlier than 5.0, then you should see “Upgrading from an Earlier Release” on page 5-32 in the Real-Time Workshop 5.0 Release Notes.

Inaccessible Signal Reporting

In previous releases, Simulink and the Real-Time Workshop reported an error whenever the Floating Scope or a user-written S-function tried to access an inaccessible signal during simulation or code generation. In this release, Simulink displays only a warning if you use the `sim` command to start the simulation. Real-Time Workshop generates neither a warning nor an error message.

Known Software and Documentation Problems

This section identifies known software and documentation problems in Version 5.2 and ealier.

Closing a Model During Code Generation Causes Segmentation Violation

If you attempt to close a model while Real-Time Workshop is generating code or the code is being compiled, you may experience unpredictable results, including segmentation violations.

The MathWorks plans to address this issue in a future release. When using the current release and all prior releases, do not close a model while a build is in process.

Included Files Documentation Error

The following areas of Real-Time Workshop documentaiton incorrectly state that in generated code, `model.h` includes `model_private.h`. This is incorrect. The file `model.h` does not include `model_private.h`.

- Real-Time Workshop 5.0 Release Notes
- "Application Modules for Application Components" in the Program Architecture chapter
- “Building an Application: Summary of Files Created by the Build Procedure” in “Getting Started with Real-Time Workshop”

The diagram of file inclusions in “Building an Application: Summary of Files Created by the Build Procedure” in “Getting Started with Real-Time Workshop” is correct, however.

Look-Up Table Blocks Do Not Support Changing Number of Repeated Zeros/Points

When Real-Time Workshop generates code for the Look-Up Table 2-D or Look-Up Table 1-D block for floating-point parameters with binary search, extrapolation, and linear interpolation selected, the number and position of repeated zeros is generated explicitly into the generated code. If you then tune the breakpoint parameters in the real-time code during execution and the number or position of repeated zeros changes, the generated code, in most cases, gets an incorrect answer when the input for that breakpoint set is near zero.

To work around this condition, do not change the number or position of reapated zeros in breakpoint parameters for the Look-Up Table 1-D and Look-Up Table 2-D blocks.

Target Language Compiler Crashes When Out of Memory

If the Target Language Compiler exhausts available memory, MATLAB hangs or crashes.

Control-C While Target Language Compiler Is Running Can Leave Model in Unstable State

The Target Language Compiler is run during an RTW build to generate the C code. This phase starts execution with the message

```
### Invoking Target Language Compiler on model.rtw
```

and finishes with the message

```
### TLC code generation complete.
```

Interrupting the TLC Compiler with **Ctrl+C** will abort the Real-Time Workshop build process and return control to the MATLAB prompt. This interruption leaves the model being built in a intermediate (compiled) state. Therefore, you should avoid issuing **Ctrl+C** during the code generation phase of a build. If you do type **Ctrl+C**, you can terminate the model's "compiled" state by issuing the following command:

```
rtwgen('model', 'TerminateCompile', 'on')
```

where `model` is the name of the model (without extension) whose build was aborted.

After you issue the above command, close the model without saving it, and reopen it before continuing.

Blocks That Depend on Absolute Time

Appendix B of the Real-Time Workshop User's Guide lists Simulink blocks that depend on absolute time. In previous releases of this documentation, this list was incomplete. It should contain the following:

- Backlash
- Chirp Signal
- Clock
- Derivative
- Digital Clock
- Discrete-Time Integrator
- From File
- From Workspace
- Pulse Generator
- Ramp
- Rate Limiter
- Repeating Sequence
- Scope
- Signal Generator
- SineWave
- Step
- To File
- To Workspace

The documentation will be updated in a later stage of this release cycle.

Model Parameter Configuration Dialog Source List Panel Description

The description of the **Source List Panel** of the **Model Parameter Configuration** dialog on p. 5-10 of the Real-Time Workshop User's Guide for Release 13 failed to describe one of the two menus on that panel, `Referenced workspace variables`. The updated description is as follows:

Source List Panel. The **Source list** panel displays a menu and a scrolling table of numerical workspace variables.

The menu lets you choose the source of the variables to be displayed in the list. There are two choices: `MATLAB workspace` (lists all variables in the MATLAB workspace that have numeric values), and `Referenced workspace variables` (lists only those variables referenced by the model). The source list displays names of variables defined in the MATLAB base workspace.

Error in Descriptions of Storage Class Declarations for Tunable Parameters

Table 5-2 in the Real-Time Workshop Users Guide (Signal Properties Options and Generated Code) had errors. It and its introduction should read as follows:

Storage Classes of Tunable Parameters

Real-Time Workshop defines four storage classes for tunable parameters. You must declare a tunable parameter to have one of the following storage classes:

- `SimulinkGlobal(Auto): SimulinkGlobal(Auto)` is the default storage class. Real-Time Workshop stores the parameter as a member of `rtP`. Each member of `rtP` is initialized to the value of the corresponding workspace variable at code generation time.
- `ExportedGlobal`: The generated code instantiates and initializes the parameter and *`model`*`.h` exports it as a global variable. An exported global variable is independent of the `rtP` data structure. Each exported global variable is initialized to the value of the corresponding workspace variable at code generation time.
- `ImportedExtern`: *`model`*`_private.h` declares the parameter as an `extern` variable. Your code must supply the proper variable definition and initializer, if any.
- `ImportedExternPointer`: *`model`*`_private.h` declares the variable as an `extern` pointer. Your code must supply the proper pointer variable definition and initializer, if any.

The generated code for *model*`.h` includes *model*`_private.h` in order to make the `extern` declarations available to subsystem files.

Table 5-2: Tunable Parameter Storage Declarations and Code

Storage Class	Generated Variable Declaration and Code
SimulinkGlobal(Auto)	`struct Parameters {`
	`  real_T Kp;`
	`}`
	(in *model*`.h`)
	`.`
	`.`
	`extern Parameters rtP;`
	(in *model*`_types.h`)
	`.`
	`.`
	`typedef struct _Paramaters Parameters;`
	(in *model*`.h`)
	`.`
	`.`
	`Parameters rtP = {`
	`  5.0`
	`};`
	(in *model*`_data.h`)
	`.`
	`.`
	`rtY.Out1 = (rtP.Kp * rtb_u);`
	(in *model*`.c`)
ExportedGlobal	`extern real_T Kp;`
	(in *model*`.h);`
	`.`
	`.`
	`real_T Kp = 5.0;`
	(in *model*`.c`)
	`.`
	`.`
	`rtY.Out1 = (Kp * rtb_u);`
	(in *model*`.c`)

Storage Class	Generated Variable Declaration and Code

ImportedExtern	`extern real_T Kp;`
	(in *model*`_private.h`)
	.
	.
	`rtY.Out1 = (Kp * rtb_u);`
	(in *model*`.c`)

ImportedExternPointer	`extern real_T *Kp;`
	(in *model*`_private.h`)
	.
	.
	`rtY.Out1 = ((*Kp) * rtb_u);`
	(in *model*`.c`)

Error in Descriptions of Storage Class Declarations for Signal Properties

Table 5-5 in the Real-Time Workshop Users Guide (Tunable Parameter Storage Declarations and Code) had errors, and should read as follows:

Table 5-5: Signal Properties Options and Generated Code

Storage Class	Declaration	Code
Auto (with storage optimizations on)	`real_T rtb_SinSig;` (in *model*.c with local scope)	`rtb_SinSig = rtP.Sine_Wave_Amp * sin(rtP.Sine_Wave_Freq * rtmGetT(rtM_Signals_examp) + rtP.Sine_Wave_Phase) + rtP.Sine_Wave_Bias;`
Simulink Global (test point)	`typedef struct _BlockIO { real_T SinSig; real_T Gain1Sig; } BlockIO;` (in *model*.h) `BlockIO rtB;` (in *model*.c)	`rtB.SinSig = rtP.Sine_Wave_Amp * sin(rtP.Sine_Wave_Freq * rtmGetT(rtM_Signals_examp) + rtP.Sine_Wave_Phase) + rtP.Sine_Wave_Bias;`
Exported Global	`extern real_T SinSig;` (in *model*.h) `real_T SinSig;` (in *model*.c)	`SinSig = rtP.Sine_Wave_Amp * sin(rtP.Sine_Wave_Freq * rtmGetT(rtM_Signals_examp) + rtP.Sine_Wave_Phase) + rtP.Sine_Wave_Bias;`
Imported Extern	`extern real_T SinSig;` (in *model*_private.h)	`SinSig = rtP.Sine_Wave_Amp * sin(rtP.Sine_Wave_Freq * rtmGetT(rtM_Signals_examp) + rtP.Sine_Wave_Phase) + rtP.Sine_Wave_Bias;`
Imported Extern Pointer	`extern real_T *SinSig;` (in *model*_private.h)	`(*SinSig) = rtP.Sine_Wave_Amp * sin(rtP.Sine_Wave_Freq * rtmGetT(rtM_Signals_examp) + rtP.Sine_Wave_Phase) + rtP.Sine_Wave_Bias;`

Included Files Documentation Error

The section "Application Modules for Application Components" on page 7-33 of the Program Architecture chapter of the Real-Time Workshop documentation incorrectly states that `model_private.h` is sub-included by `model.h`.

On page 2-9 of “Getting Started with Real-Time Workshop” the section “Building an Application: Summary of Files Created by the Build Procedure” also incorrectly states that `model.h` includes `model_private.h`.

The diagram of file inclusions in Figure 2-12 on page 2-50 is correct, however.

No Code Generation Support for 64-bit Integer Values

In Release 13 MATLAB supports both signed (`INT64`) and unsigned (`UINT64`) integers. There is, however, no corresponding support in Real-Time Workshop for such values, meaning that they cannot be read from the Workspace or declared in generated code, including downcasts.

Missing Examples Intended to Describe Combining Multiple Models

Two example models for the section called “Combining Multiple Models” in the Targeting Real-Time Systems chapter of the documentation were inadvertently not distributed in Release 12.1. These models, `fuelsys1.mdl` and `mcolon.mdl`, have been superseded by models in the `multimallocdemo.mdl` demo, and the documentation has been correspondingly updated.

Setting Environment Variable to Run Rapid Simulation Target Executables on Solaris

To run RSIM executables outside of MATLAB on the Solaris platform, you need to modify your `LD_LIBRARY_PATH` environment variable to include `bin/sol2` directory where MATLAB is installed. For example, if you have installed MATLAB under `/usr/local/MATLAB` then you need to add `/usr/local/MATLAB/bin/sol2` to your environment variable.

Limitation Affecting Rolling Regions of Noncontiguous Signals

This note describes a limitation affecting discontiguous signals that have regions that have a width greater than or equal to the **Loop rolling threshold**. (This parameter is set in the **General code generation options** category of the Real-Time Workshop pane.)

Such signal regions are called *rolling* regions.

If a rolling region of a discontiguous signal has storage class `ImportedExternPointer`, all other rolling regions of the signal must also have storage class `ImportedExternPointer`. Otherwise, a code generation error is displayed. If this error occurs, try increasing the **Loop rolling threshold**.

Code Generation Failure in Nested Directories Under Windows 98

This note describes a limitation affecting both the Simulink Accelerator and Real-Time Workshop, under Windows 98. The problem is due to a limitation of Windows 98.

If the present working directory (pwd) is a folder nested in 7 or more levels, Real-Time Workshop (or Simulink Accelerator) cannot generate code. The workaround is to connect to a higher-level (less deeply nested) directory before initiating the build process.

Turn the New Wrap Lines Option Off

The MATLAB Command Window has a new **Wrap lines** option. Real-Time Workshop frequently displays very long message lines as a build progresses. This can cause some display problems. Therefore, when using Real-Time Workshop, you should turn the **Wrap lines** option off using the **Preferences** setting. For more information on this issue, see the Technical Support Solution 29082 from the MathWorks Web page.

Filename Option in Nonvirtual Subsystem Code Generation

To set options for nonvirtual subsystem code generation, you use the subsystem's **Block Parameters** dialog. The operation of the `Auto` option of the **RTW file name options** menu in the **Block Parameters** dialog has changed since the printed version of the *Real-Time Workshop User's Guide* went to press.

In the online version of the *Real-Time Workshop User's Guide*, we have corrected the description of this option. See the "Nonvirtual Subsystem Code Generation" section of the online guide.

We repeat the corrected description here:

`Auto`: The Real-Time Workshop does *not* generate a separate file for the subsystem. Code generated from the subsystem is generated within the code module generated from the subsystem's parent system. If the subsystem's parent is the model itself, code generated from the subsystem is generated within *`model`*`.c`.

DSP Support Documentation Error

The Real-Time Workshop User's Guide section "DSP Processor Support" on p. 14-107 contains obsolete information, and should instead read as follows:

DSP targets may use registers with sizes other than 32 bits and vary in their saturation and overflow behavior. These characteristics are specified by target-specific hookfiles, which are provided for all targets supplied by The MathWorks. Users may create their own hookfiles for custom targets. The contents and naming of hook files are described in "Targeting Real-Time Systems: Components of a Custom Target Configuration: Control Files: Hook Files for Communicating Target-specific Word Characteristics" on page 14-7 of the Real-Time Workshop User's Guide. The `%assign DSP32=1` command to the system target makefile and the `-DDSP32=1` command to the template makefile that formerly handled DSP targets have been deprecated and no longer have any effect.

Real-Time Workshop 5.1.1 Release Notes

New Features

The following new feature is provided in Version 5.1.1 of Real-Time Workshop.

New -dr Command Line Switch in TLC Detects Cyclic Record Creation

The `-dr` command line option enables the Target Language Compiler to detect at run time when cyclic records are created and to produce a diagnostic message.

Cyclic records are problematic because they cause memory leaks in TLC. A cyclic record is one which ends up pointing to itself. They only can be constructed manually, as in the following example:

```
%createrecord x { }    %% create an empty record x
%createrecord y { }    %% create an empty record y

%addtorecord x field y %% add a field to x which points to y
%addtorecord y field x %% add a field to y which points to x
```

At this point, a cyclic record exists, i.e. `x.field.field == x`

As this feature significantly slows Target Language Compiler performance, it is off by default.

Major Bug Fixes

Real-Time Workshop 5.1.1 includes important bug fixes made since Version 5.1.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Version 5.1, then you should also see “Major Bug Fixes” on page 2-3 of the Real-Time Workshop 5.1 Release Notes.

Real-Time Workshop 5.1 Release Notes

New Features

If you are upgrading from a release earlier than Version 5.0.1, then you should also see “New Features” on page 4-2 of the Real-Time Workshop 5.0.1 Release Notes.

Major Bug Fixes

Real-Time Workshop 5.1 includes important bug fixes made since Version 5.0.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Version 5.0.1, then you should also see “Major Bug Fixes” on page 4-4 of the Real-Time Workshop 5.0.1 Release Notes.

Real-Time Workshop 5.0.1 Release Notes

New Features

This section introduces the new features and enhancements added in Real-Time Workshop since Version 5.0 (Release 13).

Expanded Hookfile Options

This update adds new options for specifying target characteristics via hook files.

During its build process, Real-Time Workshop checks for the existence of `<target>_rtw_info_hook.m`, where `<target>` is the base file name of the active system target file. For example, if your system target file is grt.tlc, then the hook file name is `grt_rtw_info_hook.m`. If the hook file is present (i.e., is on the MATLAB path), the target specific information is extracted via the API found in this file. Otherwise, the host computer is the assumed target.

Three hook file keyword options have been added since release 13:

- `TypeEmulationWarnSuppressLevel`: Used to supress warnings about emulation of word sizes. The default value is `0` which gives full warnings. This is the preferred setting when generating code for the production target. Increasing the value gives less warnings. When generating code for a rapid prototyping system, emulation may not be a concern and a suppression level of `2` may be desirable.
- `PreprocMaxBitsSint`: Specify limitations of the target C preprocessor to do math with signed integers. This is used to prevent errors in the preprocessor phase.

  As an example, suppose the target had 64-bit longs. Porting the generated code to a machine that does not have 64-bit longs can lead to errors in the processing of integer data types. To prevent these errors, a check is included in the generated code.

  ```
  #if ( LONG_MAX != (0x7FFFFFFFFFFFFFFFL) )
  #error Code was generated for compiler with different sized
  longs.
  #endif
  ```

  This code requires the preprocessor to compare signed 64-bit integers. Some preprocessors have bugs that cause such comparisons to yield incorrect results. The preprocessor math may only be fully correct for say 32-bit signed integers. To specify, this `PreprocMaxBitsSint` would be set to `32`. Generating the code with this setting causes problematic size checks to be skipped.

```
#if 0
/*
Skip this size verification because of preprocessor
limitation
*/
#if ( LONG_MAX != (0x7FFFFFFFFFFFFFFFL) )
#error Code was generated for compiler with different sized
longs.
#endif
#endif
```

- `PreprocMaxBitsUint`: Specify limitations of the target C preprocessor to do math with unsigned integers. This is just like `PreprocMaxBitsSint` except that it pertains to unsigned integer operations such as

```
#if ( ULONG_MAX != (0xFFFFFFFFFFFFFFFFUL) )
```

If you are not certain about the proper settings for your target, type `rtwtargetsettings` in MATLAB for more details.

Hookfiles for Customizing Make Commands

Custom targets may require a target-specific hook file to generate an appropriate make command when a non-default compiler is used. Such M-files should be located on the MATLAB path and be named `<target>_wrap_make_cmd_hook.m`, e.g. `MPC555pil_wrap_make_cmd_hook.m` for the MPC555 PIL target. When such a file exists, and returns an appropriate make command, Real-Time Workshop will override its default (e.g., LCC) batch file wrapping code. For an example make command hook file, see *`matlabroot`*`/toolbox/rtw/rtw/wrap_make_cmd.m`. Note that such hook files are distinct from the target-specific hook files that are used to describe hardware characteristics (see above).

Major Bug Fixes

Real-Time Workshop 5.0.1 includes several important bug fixes made since Version 5.0.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Release 13, then you should also see “Major Bug Fixes” on page 5-24 of the Real-Time Workshop 5.0 Release Notes.

Real-Time Workshop 5.0 Release Notes

Release Summary

Real-Time Workshop 5.0 includes many new features, numerous improvements in the quality of generated code, as well as enhancements to existing features. This section summarizes new features and enhancements added in the Real-Time Workshop 5.0 since the Real-Time Workshop 4.1 release.

New Features and Enhancements

Code Generation Infrastructure Enhancements

- “Code for Nonvirtual Subsystems Is Now Reusable” on page 5-6
- “Packaging of Generated Code Files Simplified” on page 5-8
- “Most Targets Use rtModel Instead of Root SimStruct” on page 5-10
- “Hook Files for Communicating Target-specific Word Characteristics” on page 5-10
- “Code Generation Unified for Real-Time Workshop and Stateflow” on page 5-11
- “Conditional Input Branch Execution Optimization” on page 5-11

Code Generation Configuration Features

- “Diagnostics Pane Items Classified into Logical Groups” on page 5-12
- “Comments Not Generated for Reduced Blocks When "Show eliminated statements" Is Off” on page 5-12
- “New General Code Appearance Options” on page 5-12
- “Identifier Construction for Generated Code Has Been Simplified” on page 5-14
- “GUI Control over Behavior of Assertion Blocks in Generated Code” on page 5-15
- “GUI Control Over TLC %assert Directive Evaluation” on page 5-16

Block-level Enhancements

- “New Rate Transition Block” on page 5-16

Major Bug Fixes

- “ImportedExtern and ImportedExternPointer Storage Class Data No Longer Initialized” on page 5-25
- “External Mode Properly Handles Systems with no Uploadable Blocks” on page 5-25
- “Nondefault Ports Now Usable for External Mode on Tornado Platform” on page 5-26
- “Initialize Block Outputs Even If No Block Output Has Storage Class Auto” on page 5-26
- “Code Is Generated Without Errors for Single Precision Datatype Block Outputs” on page 5-26
- “Duplicate #include Statements No Longer Generated” on page 5-26
- “Custom Storage Classes Ignored When Unlicensed for Embedded Coder” on page 5-26
- “Erroneous Sample Time Warning Messages No Longer Issued” on page 5-27
- “Discrete Integrator Block with Rolled Reset No Longer Errors Out” on page 5-27
- “Rate Limiter Block Code Generation Limitation Removed” on page 5-27
- “Multiport Switch with Expression Folding Limitation Removed” on page 5-27
- “Pulse Generator Code Generation Failures Rectified” on page 5-27
- “Stateflow I/O with ImportedExternPointer Storage Class Now Handled Correctly” on page 5-28
- “Parameters for S-Function Target Lookup Blocks May Now Be Made Tunable” on page 5-28
- “PreLook-up Index Search Block Now Handles Discontiguous Wide Input” on page 5-28
- “SimViewingDevice Subsystem No Longer Fails to Generate Code” on page 5-28
- “Accelerator Now Works with GCC Compiler on UNIX” on page 5-28

Upgrading from an Earlier Release

New Features and Enhancements

This section introduces the new features and enhancements added in the Real-Time Workshop 5.0 since the Real-Time Workshop 4.1. A number of enhancements to Simulink that can impact code generation are also described.

For information about Real-Time Workshop features that are incorporated from recent releases, see the "Real-Time Workshop 4.1 Release Notes" and the "Real-Time Workshop 4.0 Release Notes" documentation.

Note For information about closely related products that extend the Real-Time Workshop, see the Release Notes sections about the Real-Time Workshop Embedded Coder and the xPC Target.

Code Generation Infrastructure Enhancements

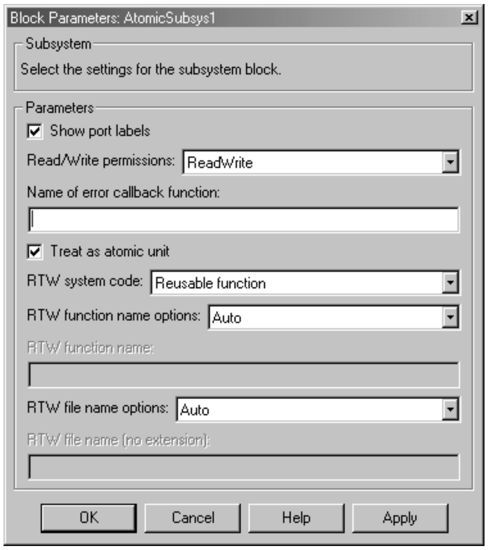
Code for Nonvirtual Subsystems Is Now Reusable

Real-Time Workshop 5.0 alters certain aspects of generated code to implement the capability to reuse code for nonvirtual subsystems. You have the ability to select or override this feature, as well as to specify function and file names from the Real-time Workshop GUI.

In prior releases, each nonvirtual subsystem in a model generated a separate block of code. In some circumstances—for example, when a library block is used multiple times in the same fashion—it is possible to generate a single shared function for the block and call that function multiple times. Consolidating code in this fashion can significantly improve the size and efficiency of generated code.

To implement code reuse, the Real-Time Workshop must pass in appropriate data elements (as function arguments) for each caller of a reused subsystem. Code generated by Real-Time Workshop 5.0 enables such arguments for functions generated for nonvirtual subsystems.

You enable code reuse through the **Subsystem parameters** dialog box when both **Treat as atomic unit** and `Reusable function` from the **RTW system code** pull-down menu are selected, as illustrated below.



Block Parameters: AtomicSubsys1

Subsystem

Select the settings for the subsystem block.

Parameters

☑ Show port labels

Read/Write permissions: ReadWrite

Name of error callback function:

☑ Treat as atomic unit

RTW system code: Reusable function

RTW function name options: Auto

RTW function name:

RTW file name options: Auto

RTW file name (no extension):

Reusable code will also be generated, when feasible, when you set **RTW system code** to `Auto`. Then, if only one instance of the subsystem exists, it will be inlined; otherwise a reusable function will be generated if other characteristics of the model allow this.

Certain conditions may make it impossible to reuse code, causing Real-Time Workshop to revert to another **RTW system code** option even though you specify `Reusable function` or `Auto`. When `Reusable function` is specified and reuse is not possible, the result will be a function without arguments. When `Auto` is specified and reuse is not possible, the result will be to inline the subsystem's code (or in special cases, create a function without arguments). Diagnostics are available in the HTML code generation report (if enabled; see "Generate HTML Report Option Available for Additional Targets") to help identify the reasons why reuse is not occurring in particular instances. In addition to providing these exception diagnostics, the HTML report's *Subsystems* section also maps each noninlined subsystem in the model to functions or reused functions in the generated code.

Requirements for Generation of Reusable Code from Stateflow Charts. To generate reusable code from a Stateflow chart, or from a subsystem containing a Stateflow chart, all of the following conditions must be met:

- The chart (or subsystem containing the chart) must be a library block (see "Working with Block Libraries" in the Simulink documentation).
- Data in the chart must not be initialized from workspace. The data property **Initialize from workspace** should be off.
- The chart must not output a function call.

See "Nonvirtual Subsystem Code Generation" in the Real Time Workshop documentation for further details.

Packaging of Generated Code Files Simplified

The packaging of generated code into `.c` and `.h` files has changed. The following table summarizes the structure of source code generated by the Real-Time Workshop. All code modules described are written to the build directory.

Note The file packaging of the Real-Time Workshop Embedded Coder differs slightly (but significantly) from the file packaging described here. See the "Data Structures and Code Modules" section in the Real-Time Workshop Embedded Coder User's Guide for further information.

Table 5-1: Real-Time Workshop File Packaging

File	Description
`model``.c`	Contains entry points for all code implementing the model algorithm (`MdlStart`, `MdlOutputs`, `MdlUpdate`, `MdlInitializeSizes`, `MdlInitializeSampleTimes`). Also contains model registration code.
`model``_private.h`	Contains local defines and local data that are required by the model and subsystems. This file is included by *`subsystem`*`.c` files in the model. You do not need to include *`model`*`_private.h` when interfacing hand-written code to a model.

Table 5-1: Real-Time Workshop File Packaging

File	Description
`model``.h`	Defines model data structures and a public interface to the model entry points and data structures. Also provides an interface to the real-time model data structure (*`model`*`_rtM`) via accessor macros. If you are interfacing your hand-written code to generated code for one or more models, you should include *`model`*`.h` for each model to which you want to interface.
`model``_data.c` `(conditional)`	*`model`*`_data.c` is conditionally generated. It contains the declarations for the parameters data structure and the constant block I/O data structure. If these data structures are not used in the model, *`model`*`_data.c` is not generated. Note that these structures are declared `extern` in *`model`*`.h`.
`model``_types.h`	Provides forward declarations for the real-time model data structure and the parameters data structure. These may be needed by function declarations of reusable functions. *`model`*`_types.h` is included by all *`subsystem`*`.h` files in the model.
`rtmodel.h`	Contains `#include` directives required by static main program modules such as `grt_main.c` and `grt_malloc_main.c`. Since these modules are not created at code generation time, they include `rt_model.h` to access model-specific data structures and entry points. If you create your own main program module, take care to include `rtmodel.h`.
`model_pt.c` `(optional)`	Provides data structures that enable a running program to access model parameters without use of external mode. To learn how to generate and use the `model_pt.c` file, see "C API for Parameter Tuning."
`model_bio.c` `(optional)`	Provides data structures that enable your code to access block outputs. To learn how to generate and use the `model_bio.c` file, see "Signal Monitoring via Block Outputs."

If you have interfaced hand-written code to code generated by previous releases of the Real-Time Workshop, you may need to remove dependencies on header files that are no longer generated. Use `#include` *`model`*`.h` directives, and remove `#include` directives referencing any of the following:

- *model*_common.h (replaced by *model*_types.h and *model*_private.h)
- *model*_export.h (replaced by *model*.h)
- *model*_prm.h (replaced by *model*_data.c)
- *model*_reg.h (subsumed by *model*.c)

Most Targets Use rtModel Instead of Root SimStruct

The GRT, GRT-Malloc, ERT, and Tornado targets now use the *rtModel* data structure to store information about the root model. In prior releases, this information was stored in the *SimStruct* data structure. Since the SimStruct data structure was also used by non-inlined S-functions, it contained a number of S-function-specific fields that were not needed to represent root model information. The new `rtModel` is a lightweight data structure that eliminates these unused fields in representing the root model. Fields in the rtModel capture model-wide information pertaining to timing, solvers, logging, model data (such as block I/O, and DWork, parameters), etc. To generate code for the ERT target, the rtModel data structure is further pruned to contain only those fields that are relevant to the model under consideration.

Note If you have previously customized GRT, GRT-Malloc, or Tornado targets, you should upgrade each customized target to use the rtModel instead of the SimStruct. You can find guidelines for this upgrade path in "Upgrading Customized GRT and GRT-Malloc Targets to Work with Release 13" on page 5-32.

Hook Files for Communicating Target-specific Word Characteristics

In order to communicate details about target hardware characteristics, such as word lengths and overflow behavior, you now need to supply an M-file named `<target>_rtw_info_hook.m`. Each system target file needs to implement a hook file. For GRT (`grt.tlc`), for example, the file must be named `grt_rtw_info_hook.m`, and needs to be on the MATLAB path. If the hookfile is not provided, default values based on the host's characteristics will be used, which may not be appropriate. For an example, see `toolbox/rtw/rtwdemos/example_rtw_info_hook.m`. In addition, note that the TLC directive `%assign DSP = 1` no longer has any effect. You need to provide a hook file instead.

Code Generation Unified for Real-Time Workshop and Stateflow

In earlier releases, code generated from Stateflow charts in a model was written to source code files distinct from the source code files (such as `model.c`, `model.h`, etc.) generated from the rest of the model.

Now, by default, Stateflow no longer generates any separate files from the Real-Time Workshop. In addition, Stateflow generated code is seamlessly integrated with other generated code. For example, all Stateflow initialization code is now inlined.

You can override the default and instruct the Real-Time Workshop to generate separate functions, within separate code files, for a Stateflow chart. To do this, use the **RTW system code** options in the **Block parameters** dialog of the Stateflow chart (see “Nonvirtual Subsystem Code Generation” in the Real-Time Workshop documentation). You can control both the names of the functions and of the code files generated.

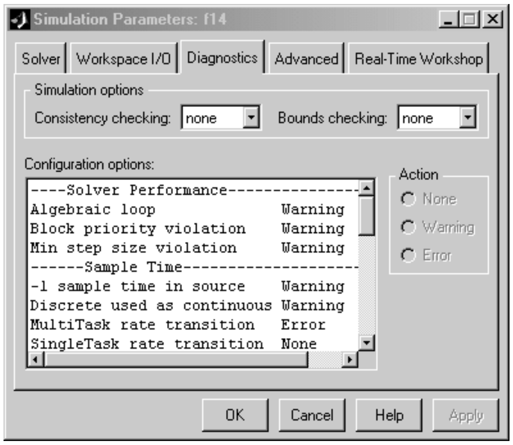
Conditional Input Branch Execution Optimization

This release introduces a new optimization called conditional input branch execution, speeding simulation and execution of code generated from the model. Previously, when simulating models containing Switch or Multiport Switch blocks, Simulink executed all blocks required to compute all inputs to each switch at each time step. In this release, Simulink, by default, executes only the blocks required to compute the control input and the data input selected by the control input at each time step. Likewise, standalone applications generated from the model by Real-Time Workshop execute only the code needed to compute the control input and the selected data input. To explore this feature, look at the `coninputexec` demo.

Code Generation Configuration Features

Diagnostics Pane Items Classified into Logical Groups

To make selecting diagnostics easier, the **Diagnostics** entries on the **Simulation Parameters** dialog have been reorganized according to functionality, and alphabetically within each group, as shown below.



Consistency checking: none   Bounds checking: none

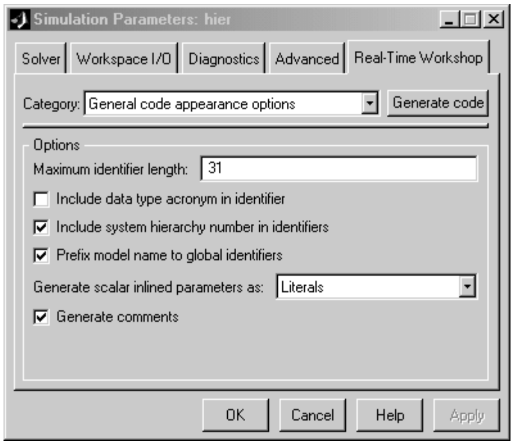
```
----Solver Performance---------------
Algebraic loop              Warning
Block priority violation    Warning
Min step size violation     Warning
------Sample Time--------------------
-1 sample time in source    Warning
Discrete used as continuous Warning
MultiTask rate transition   Error
SingleTask rate transition  None
```

Comments Not Generated for Reduced Blocks When "Show eliminated statements" Is Off

The **Show eliminated statements** option (in the Real-Time Workshop `General code generation options` category) is now off by default. As long as it remains off, Real-Time Workshop no longer generates comments referring to blocks that have been removed from the model via block reduction optimization.

New General Code Appearance Options

A new category has been added to the **Real-Time Workshop** dialog box, named `General code appearance options`. This pane adds four new code formatting options to two existing options that formerly occupied other categories. The `General code appearance` dialog is shown below.


Simulation Parameters: hier

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Category: General code appearance options

Generate code

Options

Maximum identifier length: 31

☐ Include data type acronym in identifier

☑ Include system hierarchy number in identifiers

☑ Prefix model name to global identifiers

Generate scalar inlined parameters as: Literals

☑ Generate comments

OK | Cancel | Help | Apply

The **Maximum identifier length** field allows you to limit the number of characters in function, typedef, and variable names. The default is 31 characters, but Real-Time Workshop imposes no upper limit.

Selecting **Include data type acronym in identifier** enables you to prepend acronyms such as `i32` (for long integers) to signal and work vector identifiers to make code more readable. The default is not to include datatype acronyms in identifiers.

The **Include system hierarchy number in identifiers** option, when selected, prefixes `s#_`, where `#` is a unique integer subsystem index, to identifiers declared in that subsystem. This enhances traceability of code, for example via the `hilite_system<`S#'>` command. The default is not to include a system hierarchy index in identifiers.

The **Prefix model name to global identifiers** checkbox is a new option that is `ON` by default. When this option is on, Real-Time Workshop prefixes subsystem function names with the name of the model (*`model_`*). The model name is also prefixed to the names of functions and data structures at the model level, when appropriate to the code format. This is useful when you need to compile and link code from two or more models into a single executable, as it avoids potential name clashes.

You can now exercise control over the code style for inlined parameters through a new pull-down menu, **Generate scalar inline parameters as:** `[literals | macros]`. When constant parameters are inlined and declared not tunable, the following code generation options are available:

- Vector parameters were formerly stored as constant parameters in `rtP` vectors. Now they are declared as constant vectors of appropriate type, independent of `rtP`.
- Scalar parameters were formerly inlined as literals. In addition to this approach, users now have the option to have scalar parameters expressed as `#define` macro definitions.

The default is to generate scalar inline parameters as literals.

Note S-functions can mark a run-time parameter as being constant in order to guarantee that it never ends up in the `rtP` data structure. Use `ssSetConstRunTimeParamInfo` in the S-function to register a constant run-time parameter.

Generate comments is an existing global option that was moved from the `General code generation options (cont)` category to this one. As in the prior release, the default for **Generate comments** is `ON`.

Identifier Construction for Generated Code Has Been Simplified

The methods which Real-Time Workshop uses to construct identifiers for variables and functions have been enhanced to make identifiers more understandable and more customizable. As a result of these enhancements

- Changes to sections of the model do not cause identifiers elsewhere to change.
- Reused function input arguments now derive their name from the inport block.
- Subsystem function names can be prefixed by the model name to prevent link errors due to name conflicts.
- Users may specify maximum identifier length (can be > 31 characters).
- A new option exists to include a datatype acronym in identifiers.

- Use of `_a`, `_b`, … postfixes to identifiers to prevent name clashes has been dramatically reduced.

See also "New General Code Appearance Options" on page 5-12 for related information.

GUI Control over Behavior of Assertion Blocks in Generated Code

The **Advanced** pane of the **Simulation Parameters** dialog shown above also provides you with a control to specify whether model verification blocks such as Assert, Check Static Gap, and related range check blocks will be enabled, not enabled, or default to their local settings. This **Model Verification block control** popup menu has the same effect on code generated by Real-Time Workshop as it does on simulation behavior, and also may be customized.

For Assertion blocks that are not disabled, the generated code for a model will include one of the following statements

```
utAssert(input_signal);
utAssert(input_signal != 0.0);
utAssert(input_signal != 0);
```

at appropriate locations, depending on the block's input signal type (Boolean, real, or integer, respectively).

By default `utAssert` is a no-op in generated code. For assertions to abort execution you must enable them by including a parameter in the `make_rtw` command. Specify the **Make command** field on the `Target configuration` category pane as follows:

```
make_rtw OPTS='-DDOASSERTS'
```

If you want triggered assertions to not abort execution and instead to print out the assertion statement, use the following `make_rtw` variant:

```
make_rtw OPTS='-DDOASSERTS -DPRINT_ASSERTS'
```

Finally, when running a model in accelerator mode, Simulink will call back to itself to execute assertion blocks instead of using generated code. Thus user-defined callback will still be called when assertions fail.

GUI Control Over TLC %assert Directive Evaluation

Prior versions required specifying the `-da` Target Language Compiler command switch in order for TLC `%assert` directives to be evaluated. Now users can more conveniently trigger `%assert` code by checking the **Enable TLC Assertions** box on the **TLC debugging** section of the **Real-Time Workshop** dialog page. The default state is for asserts not to be evaluated. You can also control assertion handling from the MATLAB command window:

`set_param(`*`model`*`, 'TLCAssertion', 'on|off')` to set or unset. Default is Off.

`get_param(`*`model`*`, 'TLCAssertion')` to see the current setting.

Block-level Enhancements

New Rate Transition Block

In previous releases, Zero-Order Hold and Unit Delay blocks were required to handle problems of data integrity and deterministic data transfer between blocks having different sample rates.

The new Rate Transition block lets you handle sample rate transitions in multi-rate applications with greater ease and flexibility than the Zero-Order Hold and Unit Delay blocks.

The Rate Transition block handles both types of rate transitions (fast to slow, and slow to fast). When inserted between two blocks of differing sample rates, the Rate Transition block detects the two rates and automatically configures its input and output sample rates for the appropriate type of transition.

The Rate Transition block supports the following modes of operation:

- Protected/Deterministic: By default, the Rate Transition block operates exactly like a Zero-Order Hold (for fast to slow transitions) or a Unit Delay (for slow to fast transitions), and can replace these blocks in existing models without any change in model performance. (There is one exception: in a transition between a continuous block and a discrete block, a Zero-Order Hold must be used.)

  In its default mode of operation, the Rate Transition block guarantees the integrity of data transfers and guarantees that data transfers are deterministic.
- Protected/Non-Deterministic: In this mode, data integrity is protected by double-buffering data transferred between rates. The blocks downstream from the Rate Transition block always use the latest available data from the block that drives the Rate Transition block. Maximum latency is less than or equal to 1 sample period of the faster task.

  The drawbacks of this mode are its non-deterministic timing and its use of extra memory buffers. The advantage of this mode is its low latency.
- Unprotected/Non-Deterministic: This mode is the least safe, and is not recommended for mission-critical applications. The latency of this mode is the same as for Protected/Non-Deterministic mode, but memory requirements are reduced since there is no double-buffering.

For further information on the use of the Rate Transition block with the Real-Time Workshop, see "Models With Multiple Sample Rates" in the Real-Time Workshop User's Guide. For information on the use of the Rate Transition block GUI and its use in simulation, see Using Simulink.

S-Function API Extended to Permit Users to Define DWork Properties

The S-Function API has been extended to permit specification of an Real-Time Workshop identifier, storage class, and type qualifier for each `DWork` that an S-Function creates. The extensions consist of the following macros:

- `ssGetDWorkRTWIdentifier(S,idx)`
- `ssSetDWorkRTWIdentifier(S,idx,val)`

- `ssGetDWorkRTWStorageClass(S,idx)`
- `ssSetDWorkRTWStorageClass(S,idx,val)`
- `ssGetDWorkRTWTypeQualifier(S,idx)`
- `ssSetDWorkRTWTypeQualifier(S,idx,val)`

As is the case with data store memory or discrete block states, the Real-Time Workshop identifier may resolve against a Simulink.Signal object. An example has been added to `sfundemos`, in the miscellaneous category.

Lookup Table Blocks Use New Run-time Library for Smaller Code

Lookup Table (2-D), Lookup Table (3-D), PreLook-Up Using Index Search, and Interpolation using PreLook-Up blocks now generate C code that targets one of the many new specific, optimized look-up table operations in the Real-Time Workshop run-time library. This results in dramatically smaller code size. The library look-up functions themselves incorporate further enhancements to the actual look-up algorithms for speed improvements for most option settings, especially for linear interpolations.

Relay Block Now Supports Frame-based Processing

Relay blocks can now handle frame-based input signals. Each row in a frame-based input signal is a separate set of samples in frames and each column represents a different signal channel. The block parameters should be scalars or row vectors whose length is equal to the number of signal channels. The block does not allow continuous frame-based input signals.

Transport Delay and Variable Transport Delay Improvements

Code generation for models containing the Transport Delay and Variable Transport Delay is now more space-efficient.

Storage Classes for Data Store Memory Blocks

You can now control how Data Store Memory blocks in your model are stored and represented in the generated code, by assigning storage classes and type qualifiers. You do this in almost exactly the same way you assign storage classes and type qualifiers for block states. You can also associate a Data Store Memory block with a signal object, and control code generation for the block through the signal object.

See “Storage Classes for Data Store Memory Blocks” in the Real-Time Workshop User’s Guide for further information.

Target and Mode Enhancements

Rapid Simulation Target Now Supports Variable-step Solvers

Executables generated for the Rapid Simulation (rsim) target are now able to use any Simulink solver, including variable-step solvers. To use this feature, the target system must be able to check out a Simulink license when running the generated rsim executable. You can maintain backwards compatibility (i.e., fixed-step solvers only, with no need to check out a Simulink license) by selecting `Use RTW fixed step solver` from the **Solver Selection** popup menu on the `Rapid Simulation code generation options` dialog. The default solver option is `Auto`, which will use the Simulink solver module only when the model requires it.

External Mode Support for Rapid Simulation Target

The Rapid Simulation target now includes full support for all features of Simulink external mode. External mode lets you use your Simulink block diagram as a front end for a target program that runs on external hardware or in a separate process on your host computer, and allows you to tune parameters and view or log signals as the target program executes.

External Mode Support for ERT

The Real-Time Workshop Embedded Coder now includes full support for all features of Simulink external mode. External mode lets you use your Simulink block diagram as a front end for a target program that runs on external hardware or in a separate process on your host computer, and allows you to tune parameters and view or log signals as the target program executes.

External Mode Supports Uploading Signals of All Storage Classes

Signals from all storage classes, including custom, can now be uploaded in external mode, as long as signals or parameters have addresses defined. For example, data stored as bitfields or #defines cannot be uploaded, but few other restrictions exist.

Expanded Support for Borland C Compilers

Real-Time Workshop supports version 5.6 of the Borland C compiler.

In addition, Release 13 reinstates support for Borland Version 5.2 "out-of-the-box" for all targets, except when importing Real-Time Workshop-generated S-functions. In such instances, you will need to designate the build directory where the S-function may be found via the `make_rtw` parameter `USER_INCLUDES`. For example, suppose you had generated S-function target code for model `modelA.mdl` in build directory `D:\modelA_sfcn_rtw` and were using that S-function in model `modelB.mdl`. In `modelB.mdl`, the **Make command** field of your `Target configuration` category should define `USER_INCLUDES` as follows:

```
make_rtw "USER_INCLUDES=-ID:\modelA_sfcn_rtw"
```

TLC, model.rtw, and Library Enhancements

New Simulink Data Object Properties Mapped to model.rtw Files

Simulink data objects include several new string properties that can be exploited for customizing code generation. These properties are:

```
Simulink.Data.Description
Simulink.Data.DocUnits
RTWInfo.Alias
```

In this release the Simulink engine does not make use of these properties nor does the Target Language Compiler. The properties are included in the *`model`*`.rtw` file and are reserved for future use. `RTWInfo.Alias` defines the identifier to be used in place of the parent data object (parameter, signal, or state) in the code. The engine checks that the alias is uniquely used by only that object.

SPRINTF Built-in Function Added to TLC

A C-like `sprintf` formatting function has been added which returns a TLC string encoded with data from a variable number of arguments.

`$assign str = SPRINTF(format,var,...)` formats the data in variable `var` (and in any additional variable arguments) under control of the specified `format` string, and returns a string variable containing the values. Operates like C library `sprintf()`, except that output is the return value rather than contained in an argument to sprintf.

LCC Now Links Libraries in Directory sys/lcc/lib

The template makefiles have been updated to include linking against `sys/lcc/lib`.

The BlockInstanceData Function has been Deprecated

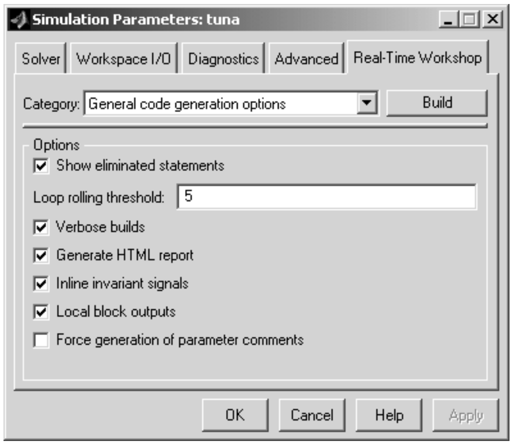
S-function TLC files should no longer use the BlockInstanceData method. All data used by a block should be declared using data type work vectors (DWork).

Documentation Enhancements

Generate HTML Report Option Available for Additional Targets

In earlier releases, the **Generate HTML report** option was available only for the Real-Time Workshop Embedded Coder. In the current release, the report is available for all targets (except the S-Function target and the Rapid Simulation target).

The **Generate HTML report** option is now located in the **General code generation options** category of the Real-Time Workshop page of the **Simulation Parameters** dialog box, as shown in the picture below.


Simulation Parameters: tuna

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Category: General code generation options

Build

Options

- [✓] Show eliminated statements
- Loop rolling threshold: 5
- [✓] Verbose builds
- [✓] Generate HTML report
- [✓] Inline invariant signals
- [✓] Local block outputs
- [ ] Force generation of parameter comments

OK | Cancel | Help | Apply

The option is on by default. Note that an abbreviated report is generated if you do not have Real-Time Workshop Embedded Coder installed.

Expression Folding API Documentation Available

The expression folding API has been documented, and is now promoted for customer use, particularly for user-written, inlined S-Functions. In addition, expanded capabilities are available that support the TLC user control variable (`ucv`) in `%roll` directives, and enable expression folding for blocks such as Selector. See "Supporting Expression Folding in S-Functions" in the Real-Time Workshop documentation.

Real-Time Workshop Documentation

The Real-Time Workshop User's Guide has been significantly updated and reorganized for Version 5.0. Information pertaining to data structures and subsystems has been updated and made more accessible, and new features and GUI changes have been documented. In addition, a new printed and online introductory volume exists, Getting Started with Real-Time Workshop. This document explains basic Real-Time Workshop concepts, organizes tutorial material for easier access, and cross-references more detailed explanations in the User's Guide.

Target Language Compiler Documentation

The Target Language Compiler Reference Guide has been significantly updated and reorganized for Version 5.0. A revised collection of tutorial examples provides new users with a more grounded introduction to TLC syntax. Documentation on the TLC Function Library and contents of `*model*.rtw` files has also been updated.

Major Bug Fixes

Real-Time Workshop 5.0 includes several bug fixes made since Version 4.1. This section describes the particularly important Version 5.0 bug fixes. If you are upgrading from a release earlier than Release 12.1, then you should also see “Bug Fixes” on page 6-11 of the Release 12.1 Release Notes.

- “ImportedExtern and ImportedExternPointer Storage Class Data No Longer Initialized” on page 5-25
- “External Mode Properly Handles Systems with no Uploadable Blocks” on page 5-25
- “Nondefault Ports Now Usable for External Mode on Tornado Platform” on page 5-26
- “Initialize Block Outputs Even If No Block Output Has Storage Class Auto” on page 5-26
- “Code Is Generated Without Errors for Single Precision Datatype Block Outputs” on page 5-26
- “Duplicate #include Statements No Longer Generated” on page 5-26
- “Custom Storage Classes Ignored When Unlicensed for Embedded Coder” on page 5-26
- “Erroneous Sample Time Warning Messages No Longer Issued” on page 5-27
- “Discrete Integrator Block with Rolled Reset No Longer Errors Out” on page 5-27
- “Rate Limiter Block Code Generation Limitation Removed” on page 5-27
- “Multiport Switch with Expression Folding Limitation Removed” on page 5-27
- “Pulse Generator Code Generation Failures Rectified” on page 5-27
- “Stateflow I/O with ImportedExternPointer Storage Class Now Handled Correctly” on page 5-28
- “Parameters for S-Function Target Lookup Blocks May Now Be Made Tunable” on page 5-28

- "PreLook-up Index Search Block Now Handles Discontiguous Wide Input" on page 5-28
- "SimViewingDevice Subsystem No Longer Fails to Generate Code" on page 5-28
- "Accelerator Now Works with GCC Compiler on UNIX" on page 5-28
- "Expression Folding Behavior for Action Subsystems Stabilized" on page 5-28
- "Dirty Flag No Longer Set During Code Generation" on page 5-29
- "Subsystem Filenames Now Completely Checked for Illegal Characters" on page 5-29
- "Sine Wave and Pulse Generator Blocks No Longer Needlessly Use Absolute Time" on page 5-29
- "Generated Code for Action Subsystems Now Correctly Guards Execution of Fixed in Minor Time Step Blocks" on page 5-29
- "Report Error when Code Generation Requested for Models with Algebraic Loops" on page 5-30

ImportedExtern and ImportedExternPointer Storage Class Data No Longer Initialized

Real-Time Workshop now reverts to its previous behavior of not initializing data whose storage class is `ImportedExtern` or `ImportedExternPointer`. Such initializations are the external code's responsibility.

External Mode Properly Handles Systems with no Uploadable Blocks

Connecting to systems with no uploadable blocks in external mode used to fail and cause Simulink to act as though a simulation was running when none was. The only way to kill the model was to kill MATLAB. Connecting to these systems now will display a warning in the MATLAB command window and then run normally.

Nondefault Ports Now Usable for External Mode on Tornado Platform

In the prior release a bug prevented the use of any but the default port to connect to a Tornado (VxWorks) target via external mode. The problem has been fixed and that configuration now works as documented.

Initialize Block Outputs Even If No Block Output Has Storage Class Auto

Previously, block outputs were initialized only if at least one block output had storage class `auto`. Now even if there are no `auto` Block I/O entries, exported globals and custom signals will be initialized.

Code Is Generated Without Errors for Single Precision Datatype Block Outputs

In cases where a reused block outputs entry is the first single-precision datatype block output in the full list of block outputs in the model, Real-Time Workshop now operates without reporting errors. See the Simulink Release Notes for related single-precision block enhancements.

Duplicate #include Statements No Longer Generated

Real-Time Workshop now creates a unique list of C header files before emitting `#include` statements in the *`model`*`.h` file (formerly placed in *`model`*`_common.h`). For backwards compatibility, the old text buffering method for includes is still available for use, but can cause multiple includes in the generated code. We urge you to update your custom code formats to use the `(S)LibAddToCommonIncludes()` functions instead of `LibCacheIncludes()`, which has been deprecated.

Custom Storage Classes Ignored When Unlicensed for Embedded Coder

If a user loads a model that uses custom storage classes, and the user is not licensed for Embedded Coder, the custom storage class is ignored (storage class reverts to auto) and a warning is produced. Previously, this situation would have generated an error.

Erroneous Sample Time Warning Messages No Longer Issued

Erroneous warnings regarding sample times not being in the sample time table for models that contain a variable sample time block and a fixed step solver are no longer issued during model compilation.

Discrete Integrator Block with Rolled Reset No Longer Errors Out

Simulink Accelerator / Real-Time Workshop used to error out if they had a Discrete Integrator block configured in 'ForwardEuler', non-level external reset, and the reset signal was a 'rolled' signal (having a width greater than 5). This has beeen fixed.

Rate Limiter Block Code Generation Limitation Removed

Simulink Accelerator will now generate code for variable step solver models that contain a rate limiter block inside an atomic subsystem.

Multiport Switch with Expression Folding Limitation Removed

Simulink Accelerator and Real-Time Workshop no longer generate a Fatal Error for Multiport Switch when expression folding is enabled.

Pulse Generator Code Generation Failures Rectified

Several problems with code generation for the pulse generator block have been eliminated:

- If the block type is PulseGenerator instead of Discrete PulseGenerator, code can now be generated.
- The scalar expansion for the delay variable is now correct.
- The start function for the Time-based mode in a variable step solver now can generate code.

Note: The first two problems also affected the Simulink Accelerator.

Stateflow I/O with ImportedExternPointer Storage Class Now Handled Correctly

Stateflow input pointers for signals of ImportedExternPointer storage class are now correctly initialized, and no longer error out for charts producing output signals that are nonscalar and of ImportedExternPointer storage class.

Parameters for S-Function Target Lookup Blocks May Now Be Made Tunable

The S-Function target code will now compile for models having lookup and Lookup Table (2-D) blocks when parameters for those blocks are tunable.

PreLook-up Index Search Block Now Handles Discontiguous Wide Input

The PreLook-up Index Search block formerly only generated code for signals from the first roll region of discontiguous wide inputs, such as from a Max block. This has been fixed.

SimViewingDevice Subsystem No Longer Fails to Generate Code

Code generation no longer aborts for atomic subsystems configured with `SimViewingDevice=on`.

Accelerator Now Works with GCC Compiler on UNIX

The previous version of the Accelerator did not work when the user selected the gcc compiler with `mex -setup`. The Accelerator now supports using the gcc compiler on UNIX systems.

Expression Folding Behavior for Action Subsystems Stabilized

When a model contains an action subsystem (e.g., a for-loop or while-iterator subsystem) and expression folding is enabled, invalid or inefficient code sometimes was generated for the model. This problem has been fixed.

Dirty Flag No Longer Set During Code Generation

In previous releases a model would be marked as *dirty* during the code generation process and the status would be restored when the process was finished. With this release the model's dirty status does not change during code generation.

Subsystem Filenames Now Completely Checked for Illegal Characters

In previous releases it was possible to specify a subsystem filename that contained illegal (non-alphanumeric) characters, if the name was long enough and the invalid characters were toward the end of the string. In this release this bug has been fixed, and the entire character string is now validated.

Sine Wave and Pulse Generator Blocks No Longer Needlessly Use Absolute Time

Previously, code generated for the Sine Wave and Pulse Generator blocks accessed absolute time when the blocks were configured as sample based. This access is not necessary and its overhead has been removed from the generated code.

Generated Code for Action Subsystems Now Correctly Guards Execution of Fixed in Minor Time Step Blocks

All blocks contained in an action subsystem must have the same rate unless some are continuous and some are fixed in minor step (a.k.a. *zoh continuous*). If there are both continuous and fixed in minor step blocks then the generated code needs to guard the code for the fixed in minor time step blocks to protect it from being executed in minor time steps.

These guards were not being generated causing some models to have wrong answers and consistency failures. This problem has been fixed and the guards are now generated.

Note This is also a fix for the Simulink Accelerator.

Report Error when Code Generation Requested for Models with Algebraic Loops

Real-Time Workshop does not support models containing algebraic loops. Version 4.1 contained a bug that enabled some models having algebraic loops to generate code which could compute incorrect answers. The models affected were those containing no algebraic loops in their root level but having algebraic loops in one or more subsystems. This bug has been fixed, and now building these models will always cause an error to be reported.

Platform Limitations for HP and IBM

Note The Release 12.0 platform limitation for Real-Time Workshop for the HP and IBM platforms still apply to Release 13. That limitation is described below.

On the HP and IBM platforms, the Real-Time Workshop opens the Release 11 **Tunable Parameters** dialog box in place of the **Model Parameter Configuration** dialog box. Although they differ in appearance, both dialogs present the same information and support the same functionality.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from Real-Time Workshop 4.1 to Version 5.0.

If you are upgrading from a version earlier than 4.1, then you should see "Upgrading from an Earlier Release" on page 6-15 in the Real-Time Workshop 4.1 Release Notes.

Replacing Obsolete Header File #includes

Generated code is packaged into fewer files in this release (see "Packaging of Generated Code Files Simplified" on page 5-8). If you have interfaced code to code generated by previous releases of Real-Time Workshop, you may need to remove dependencies on header files that are no longer generated (such as `*model*_common.h`, `*model*_export.h`, `*model*_prm.h`, and `*model*_reg.h`) and add `#include *model*.h` directives.

Custom Code Blocks Moved from Simulink Library

The Custom Code blocks have been removed in Real-Time Workshop version 5.0 (R13). These blocks are now located in a new library, named `custcode.mdl` (type `custcode` to access them). Because custom code blocks are linked to this new library, backward compatibility is assured.

Updating Custom TLC Code

In this release, a number of changes have been made to `*model*.rtw` files. If your applications depend on parsing `*model*.rtw` files using customized TLC scripts, please read "model.rtw Changes Between Real-Time Workshop 5.0 and 4.1" in Appendix A of the Target Language Compiler documentation, which describes the structure and contents of compiled models.

Upgrading Customized GRT and GRT-Malloc Targets to Work with Release 13

Substantial changes have been made to the GRT and GRT-Malloc targets in Release 13 to improve the efficiency of generated code. If you have customized either type of target, you should make changes to your modified files to ensure that your target works properly with Release 13 (Real-Time Workshop Version 5.0).

We highly recommend that you begin with the versions of the target files included in this release, and introduce all of your existing customizations to them. If you are unable to follow this upgrade path, then you would need to perform all of the steps outlined in sections A and B below.

A. Changes Resulting from the Replacement of SimStruct with the rtModel

Prior to Release 13 of Real-Time Workshop, the GRT and GRT-Malloc targets used the SimStruct data-structure to capture and store model-wide information. Since the SimStruct was also used by noninlined S-functions, it suffered from the drawback that some of its fields remained unused when it was used to capture root (model-wide) information. To avoid this drawback, Version 5.0 introduces a special data structure called the *rtModel* to capture root model data.

As a result, `grt_main.c` and `grt_malloc_main.c` need to be updated to accommodate rtModel. Following are the changes that you need to make to these files to use the rtModel instead of the SimStruct:

- Include `rtmodel.h` instead of `simstruc.h` at the top.
- Since the rtModel data-structure has a type that includes the model name, you need to include the following lines at the top of the file:

  ```
  #define EXPAND_CONCAT(name1,name2) name1 ## name2
  #define CONCAT(name1,name2) EXPAND_CONCAT(name1,name2)
  #define RT_MODEL            CONCAT(MODEL,_rtModel)
  ```

- Change the extern declaration for the function that creates and initializes the SimStruct to be:

  ```
  extern RT_MODEL *MODEL(void);
  ```

- Change the definitions of `rt_CreateIntegrationData` and `rt_UpdateContinuousStates` to be as shown in the Release 13 version of `grt_main.c` (or `grt_malloc_main.c`).
- Change all function prototypes to have the argument `'RT_MODEL'` instead of the argument `'SimStruct'`.
- The prototypes for the functions `rt_GetNextSampleHit`, `rt_UpdateDiscreteTaskSampleHits`, `rt_UpdateContinuousStates`,

rt_UpdateDiscreteEvents, rt_UpdateDiscreteTaskTime, and rt_InitTimingEngine have changed. You need to change their names to use the prefix rt_Sim instead of rt_ and then change the arguments you pass into them.

See grt_main.c (or grt_malloc_main.c) for the list of arguments that need to be passed into each function.

- You need to modify the all macros that refer to the SimStruct to now refer to the rtModel. Examples of these modifications include changing
  - ssGetErrorStatus to rtmGetErrorStatus
  - ssGetSampleTime to rtmGetSampleTime
  - ssGetSampleHitPtr to rtmGetSampleHitPtr
  - ssGetStopRequested to rtmGetStopRequested
  - ssGetTFinal to rtmGetTFinal
  - ssGetT to rtmGetT

In addition to the changes to the main C files, you need to change the target TLC file and the template make files.

- In your template make file, you need to define the symbol USE_RTMODEL. See one of the GRT or GRT-Malloc template make files for an example.
- In your target TLC file, you need to add the following global variable assignment:

```
%assign GenRTModel = TLC_TRUE
```

B. Changes Resulting from Moving the Logging Code to the Real-Time Workshop Library:

In Release 13, all the support functions used for logging data have been moved from rtwlog.c to the Real-Time Workshop library. As a result, you need to make the following changes to ensure compatibility with the new logging functions:

- Remove rtwlog.c from all of your template make files.
- In your target's main C file (which was derived from grt_main.c or grt_malloc_main.c), include rt_logging.h instead of rtwlog.h.

Real-Time Workshop 4.1 Release Notes

Release Summary

Real-Time Workshop 4.1 includes significant new and enhanced features and many improvements in the quality of generated code, including:

- Expression folding, which increases code efficiency and decreases code usage
- External mode support for inlined parameters
- Block states can now be interfaced to externally written code, in a manner similar to signals
- New debugger for Target Language Compiler (TLC) programs
- Support for new Simulink blocks, including control flow constructs such as do-while, for, and if
- Numerous bug fixes

New Features

This section introduces the new features and enhancements added in the Real-Time Workshop 4.1 since the Real-Time Workshop 4.0.

For information about Real-Time Workshop features that are incorporated from recent releases, see "Release Summary" on page 7-2.

Note For information about closely related products that extend the Real-Time Workshop, see the Release Notes sections about the Real-Time Workshop Embedded Coder and xPC Target.

Block Reduction Option On by Default

The **Block reduction** option (on the Advanced page of the **Simulation Parameters** dialog box) is now turned on by default. In prior releases, this option was off by default.

Block reduction collapses certain groups of blocks into a single, more efficient block, or removes them entirely. This results in faster model execution during simulation and in generated code.

See "Block Reduction Option" in the *Real-Time Workshop User's Guide* for further information.

Buffer Reuse Code Generation Option

The **Buffer reuse** option is now available via the **General Code Generation Options (cont.)** category of the Real-Time Workshop page. When the **Buffer reuse** option is selected, signal storage is reused whenever possible.

In previous releases, this option was available only through MATLAB `set_param` and `get_param` commands, such as:

```
set_param(gcs,'bufferreuse','on')
```

The `set_param` and `get_param` commands are still supported.

See "Buffer Reuse Option" and "Signals: Storage, Optimization, and Interfacing" in the *Real-Time Workshop User's Guide* for further information.

Build Directory Validation

The build process now disallows building programs in the MATLAB directory tree. If you attempt to generate code in the MATLAB directory tree, an error message will be displayed prompting you to change to a working directory that is not in the MATLAB directory tree. On a PC, you can continue to build in the directory `*matlabroot*/Work`.

The build process also prevents building programs when *`matlabroot`* has a dollar sign (`$`) in its MATLAB directory name.

Build Subsystem Enhancements

The **Build Subsystem** feature, introduced in Real-Time Workshop 4.0, lets you generate code and build an executable from any nonvirtual subsystem within a model. In Real-Time Workshop 4.1, the Build Subsystem feature has been enhanced as follows:

- The **Build Subsystem** window now displays additional information about block parameters referenced by the subsystem.
- From the **Build Subsystem** window, you can now inline any parameter or set the storage class of any parameter.

See "Generating Code and Executables from Subsystems" in the *Real-Time Workshop User's Guide* for further information.

C API for Parameter Tuning Documented

The Real-Time Workshop provides data structures and a C API that enable a running program to access model parameters without use of external mode. This API has now been fully documented.

To access model parameters via the C API, you generate a model-specific parameter mapping file, `model_pt.c`. This file contains parameter mapping arrays containing information required for parameter tuning.

See "C API for Parameter Tuning" in the *Real-Time Workshop User's Guide* for information on how to generate and use the parameter mapping file.

Code Readability Improvements

Improvements to the readability of generated code include:

- Elimination of redundant parentheses.
- Long C statements in the generated code are now split across multiple lines.
- Block comments are more informative.

Control Flow Blocks Support

Simulink 4.1 implements a number of blocks that support logic constructs such as if-else and switch, and looping constructs such as do-while, for, and while. The Real-Time Workshop 4.1 supports code generation from these blocks.

For further information on the flow control blocks, see "Control Flow Statements" in *Using Simulink*.

Expression Folding

Expression folding is a code optimization technique that minimizes the computation of intermediate results at block outputs, and the storage of such results in temporary buffers or variables. Wherever possible, the Real-Time Workshop collapses, or "folds," block computations into single expressions, instead of generating separate code statements and storage declarations for each block in the model.

Expression folding dramatically improves the efficiency of generated code, frequently achieving results that compare favorably to hand-optimized code. In many cases, model computations fold into a single highly optimized line of code.

Most Simulink blocks support expression folding.

For further information, see "Expression Folding" in the *Real-Time Workshop User's Guide.*

External Mode Enhancements

Inline Parameters Support

The Real-Time Workshop now lets you use the **Inline parameters** code generation option when building an external mode target program. When you inline parameters, you can use the **Model Parameter Configuration** dialog to remove individual parameters from inlining and declare them to be tunable. This allows you to improve overall efficiency by inlining most parameters, while at the same time retaining the flexibility of run-time tuning for selected parameters that are important to your application. In addition, the **Model Parameter Configuration** dialog offers you options for controlling how parameters are represented in the generated code.

Each time Simulink connects to a target program that was generated with **Inline parameters** on, the target program uploads the current value of its tunable parameters (if any) to the host. These values are assigned to the corresponding MATLAB workspace variables. This procedure ensures that the host and target are synchronized with respect to parameter values.

All targets that support external mode (i.e., `grt`, `grt_malloc`, and Tornado) now allow inline parameters.

See “Overview of External Mode Communications” in the *Real-Time Workshop User’s Guide* for further information.

Status Bar Display

When Simulink is connected to a running external mode target program, the simulation time and other status bar information is now displayed and updated just as it would be in normal mode.

Generate Comments Option

This option lets you control whether or not comments are written in the generated code. See “Generate Comments” in the *Real-Time Workshop User’s Guide* for further information.

Include System Hierarchy in Identifiers

When this option is on, the Real-Time Workshop inserts system identification tags in the generated code (in addition to tags included in comments). The tags help you to identify the nesting level, within your source model, of the block that generated a given line of code.

See “Include System Hierarchy in Identifiers” in the *Real-Time Workshop User’s Guide* for further information.

Rapid Simulation Target Supports Inline Parameters

The Rapid Simulation Target now works with **Inline parameters** on. Note that when **Inline parameters** is on, the storage class for all parameters and signals is silently forced to `auto`.

S-Function Target Enhancements

The S-Function Target **Generate S-function** feature, introduced in Real-Time Workshop 4.0, lets you generate an S-function from a subsystem. This feature has been enhanced as follows:

- The **Generate S-function** window now displays additional information about block parameters referenced by the generating subsystem.
- If you have installed and licensed the Real-Time Workshop Embedded Coder, the **Generate S-function** window lets you invoke the Embedded Coder to generate an S-function wrapper.

See “Automated S-Function Generation” in the *Real-Time Workshop User’s Guide* for details.

Storage Classes for Block States

For certain block types, the Real-Time Workshop lets you control how block states in your model are stored and represented in the generated code. Using the **State Properties** dialog, you can:

- Control whether or not states declared in generated code are interfaceable (visible) to externally written code. You can also specify that signals are to be stored in locations declared by externally written code.
- Assign symbolic names to block states in generated code.

For further information, see “Block States: Storing and Interfacing” in the *Real-Time Workshop User’s Guide*.

Support for tilde (~) in Filenames on UNIX Platforms

All filename fields in Simulink now support the mapping of the tilde (~) character in filenames. For example, in a To File block you can specify `<code>~/outdir/file.mat</code>`. On most systems, this will expand to `/home/$USER/outdir/file.mat`. The Real-Time Workshop uses the expanded names.

Target Language Compiler 4.1

This section summarizes changes that have been made to the Target Language Compiler in this release. See also “TLC Compatibility Issues” on page 6-16.

New TLC Debugger

The TLC debugger helps you identify programming errors in your TLC code. The debugger lets you set breakpoints in your TLC code, execute TLC code line-by-line, examine and change variables, and perform many other useful operations.

The TLC debugger operates during code generation, incurring almost no overhead in the code generation process. You can invoke the debugger:

- By selecting options in the **TLC debugging options** category of the Real-Time Workshop page
- By including `%breakpoint` statements in your TLC file.
- By using the MATLAB `tlc` command, as in

  ```
  tlc -dc <options>
  ```

- By using the `-dc` build option in the **System target file** field of the **Real-Time Workshop** page.

For further information, see “Debugging TLC” in the *Target Language Compiler Reference Guide*.

model.rtw File Format Changes

The format of the *`model`*`.rtw` file has changed. See “TLC Compatibility Issues” on page 6-16.

Cleanup of Block I/O Connection Handling in TLC

The handling of signal connections in `rtw/c/tlc/blkiolib.tlc` and `rtw/ada/tlc/blkiolib.tlc` was reworked. See the discussion of `LibBlockInputSignal` in “TLC Function Library Reference” in the *Target Language Compiler Reference Guide*.

Literal String Support

If a string constant is prefixed by the `L` format specifier, then no escape character processing is performed on that string. This is useful for specifying PC style path or directory names, as in this example.

```
%addincludepath L"C:\mytlc"
```

New TLC Library Functions

The following functions have been added to the TLC Function Library:

- `LibBlockInputSignalConnected`
- `LibBlockInputSignalLocalSampleTimeIndex`
- `LibBlockInputSignalOffsetTime`
- `LibBlockInputSignalSampleTime`
- `LibBlockInputSignalSampleTimeIndex`
- `LibBlockOutputSignalOffsetTime`
- `LibBlockOutputSignalSampleTime`
- `LibBlockOutputSignalSampleTimeIndex`
- `LibBlockMatrixParameterBaseAddr`
- `LibBlockParameterBaseAddr`
- `LibBlockNonSampledZC`

See “TLC Function Library Reference” in the *Target Language Compiler Reference Guide* for information on these functions.

TLC Bug Fixes

- Fixed a bug where local variables of calling functions were sometimes incorrectly visible to called functions.
- The `ISINF`, `ISNAN`, and `ISFINITE` functions now work for complex values.
- The `%filescope` directive now works as documented.

- Zero indexing on complex numbers is now supported.

  In prior releases, the Target Language Compiler allowed 0 indexing for integer and real values, but not for complex values. This restriction has been removed in the Target Language Compiler 4.1, as shown in the following example.

  ```
  %assign a = 1.0 + 3.0i
  %assign b = a[0] %% zero index now allowed
  ```

- Fixed a crash that occurred if `ROLL_ITERATIONS` was called outside of a `%roll` construct. `ROLL_ITERATIONS` returns `NULL` if called outside of a `%roll` construct.
- TLC now allows use of any path separator character independent of operating system. You can use either `\` or `/` as a path separator character on Unix or Windows).
- Fixed a bug in the compare for equality operation. `0.0` now compares equal to `-0.0`.

Bug Fixes

The Real-Time Workshop 4.1 includes the bug fixes described in this section. See also “TLC Bug Fixes” on page 6-9 for bug fixes specific to the Target Language Compiler.

Block Reduction Crash Fixed

A problem that crashed MATLAB due to a segmentation fault during the block reduction process has been fixed. This problem occurred only if the **Block Reduction** option was on, and if a Scope block was connected to a block that was removed due to block reduction.

Build Subsystem Gives Better Error Message for Function Call Subsystems

The **Build Subsystem** feature does not currently support triggered function-call subsystems. The Real-Time Workshop now gives a a more informative error message when a **Build Subsystem** is attempted with a triggered function-call subsystem.

Check Consistency of Parameter Storage Class and Type Qualifier

The Real-Time Workshop now checks for consistency of parameter storage class and type qualifier when a parameter is specified by both the **Model Parameter Configuration** dialog box and a referenced Simulink data object.

Code Optimization for Unsigned Saturation and DeadZone Blocks

When the lower limit of a Saturation or DeadZone block is a zero and is nontunable, and the data type is unsigned, the comparison against the lower limit is omitted from the code. Similarly, if the upper or lower limit of the Saturation block is nontunable and nonfinite, the comparison against the infinite limit is omitted.

Correct Code Generation of Fixed-Point Blockset Blocks in DSP Blockset Models

A code generation bug involving some DSP Blockset blocks (see list below) was fixed. When these blocks were driven by a block from the Fixed-Point Blockset, generated code would write outside array memory bounds. The following DSP Blockset blocks generated incorrect code:

- Delay Line
- Frame Status Conversion
- Matrix Multiply
- Multiport Selector
- Pad
- Submatrix
- Window Function
- Zero Pad

Correct Compilation with Green Hills and DDI Compilers

Compilation errors for files associated with matrix inversion in the `*matlabroot*/rtw/c/libsrc` directory were fixed. These errors occurred with the Green Hills and DDI compilers.

Fixed Build Error with Models Having Names Identical to Windows NT Commands

This fix prevents an error that occurred when building models having names identical to Windows NT internal commands. Examples would be models named `verify` or `path`. Such model names are now allowed.

Fixed Error Copying Custom Code Blocks

An error in the Custom Code block `Copyfcn` callback was fixed. The problem caused an error when copying a custom code block within a model.

Fixed Error in commonmaplib.tlc

A typo in rev 1.17 of `commonmap.tlc` was fixed. This typo caused an error during code generation, when using the `grt_malloc` target with **External mode** selected.

Fixed Name Clashes with Run-Time Library Functions

The Real-Time Workshop now uses the macros `rt_min` and `rt_max` to avoid clashing with run-time library `min` and `max` functions.

Improved Handling of Sample Times

The sample time handling for the S-function and ERT targets has been improved to use the compiled sample time instead of the user specified sample time on the input port blocks.

Look-Up Table (n-D) Code Generation Bug Fix

The Real-Time Workshop now generates correct code for Look-Up Table (n-D) blocks having 5 or more dimensions with different dimension sizes.

Parenthesize Negative Numerics in Fcn Block Expressions

Fcn block expressions in the generated code failed to compile in the case of a unary operator preceding a workspace variable with a negative value, such as the expression

```
-v*u
```

Such expressions are now enclosed in parentheses, as in

```
(-v) * u
```

Removed Unnecessary Warnings and Declarations from Generated Code

Several unnecessary warnings and declarations in the generated code have been removed. These include:

- In functions where the `tid` argument is not referenced, the declaration

  ```
  (void)tid
  ```

  is no longer generated. (The `tid` argument is required because the function API is predefined.)
- Warnings involving `const` casts were suppressed in some of the `rtw/c/libsrc` modules.

Retain .rtw File Option Now Works in Accelerator Mode

In previous releases, the **Retain .rtw file** option (on the TLC Debugging Options page of the **Simulation Parameters** dialog box) was ignored if Simulink was in Accelerator mode. Now, you can retain the *`model`*`.rtw` file during a build, regardless of the simulation mode.

S-Function Target Memory Allocation Bug Fix

A segmentation fault during generation of S-functions was removed by fixing the memory management of the port data structure.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from the Real-Time Workshop 4.0 (Release 12.0) to the Real-Time Workshop 4.1.

For information about upgrading from a release earlier than 4.0, see the “Upgrading from an Earlier Release” on page 7-12 in the Real-Time Workshop 4.0 Release Notes.

RTWInfo Property Changes

If you use the Simulink Data Object classes `Simulink.Signal` or `Simulink.Parameter`, or have implemented classes derived from these, note the following information concerning the `RTWInfo` properties.

In Release 12.0, the `RTWInfo` class had a `TypeQualifier` property, corresponding to the **RTW storage type qualifier** field of signal ports and parameters.

Real-Time Workshop 4.1 now supports creation of custom storage classes, removing the need for the `TypeQualifier` property. We recommend that you use custom storage classes when type qualification is needed.

By default, the `TypeQualifier` property of `RTWInfo` objects is no longer visible in the Simulink Data Explorer. Also, the `TypeQualifier` property is no longer written to `ObjectProperties` records in the *`model`*`.rtw` file. For backward compatibility, the `TypeQualifier` property remains active. The property can be set and retrieved through a direct reference. For example,

```
Kp.RTWInfo.TypeQualifier = 'const'
```

or

```
tq = Kp.RTWInfo.TypeQualifier
```

You can make the `TypeQualifier` property visible in the Simulink Data Explorer for the duration of a MATLAB session. To do this, execute the following command prior to opening the Simulink Data Explorer),

```
feature('RTWInfoTypeQualifier',1)
```

The above command also directs the Real-Time Workshop to include the `TypeQualifier` property in `ObjectProperties` records in the *`model`*`.rtw` file.

For further information see “Simulink Data Objects and Code Generation” in the *Real-Time Workshop User’s Guide*.

S-Function Target MEX-Files Must Be Rebuilt

S-function MEX-files generated by the S-function target under Release 11 are not compatible with Release 12. The incompatibilities are due to new features, such as parameter pooling, introduced in Release 12.0.

If you have built S-function MEX-files with the S-function target under Release 11, you must rebuild them. See “The S-Function Target” in the *Real-Time Workshop User’s Guide* for further information.

TLC Compatibility Issues

model.rtw File Format Changes

The format of the `model.rtw` file has changed. For further information, see “model.rtw Changes Between Real-Time Workshop 4.0 and 4.1” in the *Target Language Compiler Reference Guide*.

Reordering of BlockTypeSetup and BlockInstanceSetup Calls

During the initialization phase of code generation, the Target Language Compiler makes a pass over all blocks in the model and executes several functions, including:

- Each block’s `BlockTypeSetup` function the first time that block type is encountered.
- Each block’s `BlockInstanceSetup` function. `BlockInstanceSetup` is called for all instances of a given block type in the model.

The order in which these calls are made is significant, because the `BlockInstanceSetup` function may depend upon global variables that are initialized by the `BlockTypeSetup` function.

In Release 12.1, the `BlockTypeSetup` function is called before the `BlockInstanceSetup`. This corrects a problem in previous releases, where `BlockInstanceSetup` was erroneously called first. You may need to change your S-functions or block implementations if they depend upon the previous behavior.

Obsolete Code Generation Variables

The code generation variables `FunctionInlineType` and `PragmaInlineString` are now obsolete. These variables controlled the generation of inlined functions. In the current release, you can generate inlined functions from subsystems, as described in “Nonvirtual Subsystem Code Generation” in the *Real-Time Workshop User’s Guide*.

Real-Time Workshop 4.0 Release Notes

Release Summary

Release 4.0 of the Real-Time Workshop is a major upgrade, incorporating significant new and enhanced features and many improvements in the quality of generated code. These include:

- Significantly faster Target Language Compiler (TLC) code generation process
- TLC Profiler report for debugging TLC programs
- New efficiencies in generated code include improved signal storage reuse, constant block elimination, and parameter pooling.
- New Real-Time Workshop Embedded Coder add-on product replaces and significantly enhances the Embedded Real-Time (ERT) target.
- User interface improvements, including a redesigned Real-Time Workshop page and Model Parameter Configuration (tunable parameters) dialog
- Support for additional Simulink blocks, including Look-Up table blocks with very efficient generated code
- S-Function Target support for variable step solvers and parameter tuning
- Support for matrix operations for most Simulink blocks
- Support for frame-based processing for DSP blocks
- External mode support for many additional block types for signal uploading
- Automatic generation of S-function wrappers for embedded code, allowing for validation of generated code in Simulink
- Support for generation of code and executables from subsystems
- Support for Simulink data objects in code generation
- Support for generation of ASAP2 data definition files

New Features

This section introduces the new features and enhancements added in the Real-Time Workshop 4.0 since the Real-Time Workshop 3.0.1.

Real-Time Workshop Embedded Coder

The Real-Time Workshop Embedded Coder is a new add-on product that replaces and enhances the Embedded Real-Time (ERT) target.

The Real-Time Workshop Embedded Coder is 100% compatible with the ERT target. In addition to supporting all previous functions of the ERT target, the Real-Time Workshop Embedded Coder includes many enhancements.

See the Real-Time Workshop Embedded Coder documentation for details..

Simulink Data Object Support

The Real-Time Workshop supports the new Simulink data objects feature. Simulink provides the built-in `Simulink.Parameter` and `Simulink.Signal` classes for use with the Real-Time Workshop. Using these classes, you can create parameter and signal objects and assign storage classes and storage type qualifiers to the objects. These properties control how the generated code represents signals and parameters. The `Simulink.Parameter` and `Simulink.Signal` classes can be extended to include user-defined properties.

See “Simulink Data Objects and Code Generation” in the *Real-Time Workshop User’s Guide* for complete details.

ASAP2 Support

ASAP2 is a data definition standard proposed by the Association for Standardization of Automation and Measuring Systems (ASAM). This standard is used for data measurement, calibration, and diagnostic systems.

The Real-Time Workshop now lets you export an ASAP2 file containing information about your model during the code generation process. See “Generating ASAP2 Files” in the Real-Time Workshop online documentation.

Enhanced Real-Time Workshop Page

The Real-Time Workshop page of the **Simulation Parameters** dialog box has been reorganized and made easier to use. See “Overview of the Real-Time Workshop User Interface” in the *Real-Time Workshop User’s Guide* for complete details.

Other User Interface Enhancements

The **Tools** menu of the Simulink window now contains a **Real-Time Workshop** submenu with shortcuts to frequently used features. See “Real-Time Workshop Submenu” in the *Real-Time Workshop User’s Guide* for details.

You can now select a target configuration from the System Target File Browser by double-clicking on the desired entry in the target list. The previous selection method — selecting an entry and clicking the **OK** button — is still supported.

Advanced Options Page

An Advanced options page has been added to the **Simulation Parameters** dialog box. The Advanced page contains new code generation options, as well as options formerly located in the Diagnostics and Real-Time Workshop pages. See “Advanced Options Page” for details.

Model Parameter Configuration Dialog

The **Model Parameter Configuration** dialog box replaces the **Tunable Parameters** dialog box. The **Model Parameter Configuration** dialog box enables you to declare individual parameters to be tunable and to control the generated storage declarations for each parameter. See “Parameters: Storage, Interfacing and Tuning” in the *Real-Time Workshop User’s Guide* for details.

Tunable Expressions Supported

A tunable expression is an expression that contains one or more tunable parameters. Tunable expressions are now supported during simulation and in generated code.

Tunable expressions are allowed in masked subsystems. You can use tunable parameter names or tunable expressions in a masked subsystem dialog. When referenced in lower-level subsystems, such parameters remain tunable.

See “Tunable Expressions” in the *Real-Time Workshop User’s Guide* for a detailed description of the use of tunable parameters in expressions.

S-Function Target Enhancements

S-function target enhancements include:

- The S-function target now supports variable-step solvers.
- The S-function target now supports tunable parameters.
- The new **Generate S-function** feature lets you automatically generate an S-function from a subsystem.

The S-function target is now documented the *Real-Time Workshop User’s Guide*. See the chapter “The S-Function Target” for a full description of S-function target features.

External Mode Enhancements

Several new features have been added to external mode:

- The default operation of the **External Signal & Triggering** dialog box has been changed to make monitoring the target program simpler. See “External Signal & Triggering Dialog Box” in the *Real-Time Workshop User’s Guide* for details.
- Signal Viewing Subsystems have been implemented to let you encapsulate processing and viewing of signals received from the target system. Signal Viewing Subsystems run only on the host, generating no code in the target system. This is useful in situations where you want to process or condition signals before viewing or logging them, but you do not want to perform these tasks on the target system. See “Signal Viewing Subsystems” in the *Real-Time Workshop User’s Guide* for details.
- Previously, only Scope blocks could be used in external mode to receive and view signals uploaded from the target program. The following now support external mode:
  - Dials & Gauges Blockset
  - Display blocks
  - To Workspace blocks
  - Signal Viewing Subsystems

  - S-functions

  See "External Mode Compatible Blocks and Subsystems" in the *Real-Time Workshop User's Guide* for details.

- In Release 12, we have documented the external mode communications application program interface (API). If you want to implement external mode communications via your own low-level protocol, see "Creating an External Mode Communications Channel" in the *Real-Time Workshop User's Guide*.

Build Directory

The Real-Time Workshop now creates a *build directory* within your working directory. The build directory stores generated source code and other files created during the build process. The build directory name, `model_target_rtw`, derives from the name of the source model and the chosen target.

See "Directories Used in the Build Process" in the *Real-Time Workshop User's Guide* for details.

Note If you have created custom targets for the Real-Time Workshop under Release 11, you must update your custom system target files and template makefiles to create and utilize the build directory. See "Updating Release 11 Custom Targets" on page 7-12.

Code Optimization Features

This section describes new or modified code generation options that are designed to help you optimize your generated code. The options described are located on the Advanced page of the **Simulation Parameters** dialog box. See "Advanced Options Page" for full details.

- **Block reduction**: When the **Block reduction** option is selected, Simulink collapses certain groups of blocks into a single, more efficient block, or removes them entirely. This results in faster model execution during simulation and in generated code.
- **Parameter pooling**: When multiple block parameters refer to storage locations that are separately defined but structurally identical, you can use this option to save memory.

- **Signal storage reuse**: This option replaces the (Enable/Disable) **Optimized block I/O storage** option of previous releases. **Signal storage reuse** is functionally identical to the older feature. Turning **Signal storage reuse** on is equivalent to enabling **Optimized block I/O storage**.

See the chapter “Optimizing the Model for Code Generation” in the *Real-Time Workshop User’s Guide* for further information on code optimization.

Subsystem Based Code Generation

The Real-Time Workshop now generates code and builds an executable from any subsystem within a model. The build process uses the code generation and build parameters of the root model. See “Generating Code and Executables from Subsystems” in the *Real-Time Workshop User’s Guide* for details.

Nonvirtual Subsystem Code Generation

The Real-Time Workshop now lets you generate code modules at the subsystem level. This feature applies only to nonvirtual subsystems. With nonvirtual subsystem code generation, you control how many files are generated, as well as the file and function names. To set options for nonvirtual subsystem code generation, you use the subsystem’s **Block Parameters** dialog.

Nonvirtual subsystem code generation is a more general and flexible method of controlling the number and size of generated files than the **Function management** code generation options (**File splitting** and **Function splitting**) used in previous releases. The **Function management** code generation options have been replaced by nonvirtual subsystem code generation.

See “Nonvirtual Subsystem Code Generation” in the *Real-Time Workshop User’s Guide* for details.

Note The operation of the `Auto` option of the **RTW file name options** menu in the **Block Parameters** dialog has changed since the printed version of the *Real-Time Workshop User’s Guide* went to press

---

Filename Extensions for Generated Files

In previous releases, some generated files were given special filename extensions, such as `.prm` or `.reg`. All the Real-Time Workshop generated code and header files now use standard filename extensions (`.c` and `.h`). The file naming conventions for the following generated files have changed:

- Model registration file (formerly `model.reg`) is now named `model_reg.h`.
- Model parameter file (formerly `model.prm`) is now named `model_prm.h`.
- `BlockIOSignals` struct file (formerly `model.bio`) is now named `model_bio.c`.
- `ParameterTuning` file (formerly `model.pt`) is now named `model_pt.c`.
- External mode data type transition file (formerly `model.dt`) is now named `model_dt.c`.

hilite_system and Code Tracing

The Real-Time Workshop writes system/block identification tags in the generated code. The tags are designed to help you identify the block, in your source model, that generated a given line of code. In previous releases, the `locate_system` command was used to trace a tag back to the generating block.

The new `hilite_system` command replaces `locate_system`, for the purposes of tracing the Real-Time Workshop identification tags. You should use the `hilite_system` command to trace a tag back to the generating block. For further information on identification tags and code tracing, see “Tracing Generated Code Back to Your Simulink Model.”

Generation of Parameter Comments

The **Force generation of parameter comments** option in the **General code generation options** category of the Real-Time Workshop page controls the generation of comments in the model parameter structure (`rtP`) declaration in `model_prm.h`. This lets you reduce the size of the generated file for models with a large number of parameters. See “Force Generation of Parameter Comments Option” in the *Real-Time Workshop User’s Guide* for details.

Borland 5.4 Compiler Support

The Real-Time Workshop now supports Version 5.4 of the Borland C/C++ compiler.

Enhanced Makefile Include Path Rules

Two new rules and macros have been added to Real-Time Workshop template makefiles. These rules let you add source and include directories to makefiles generated by Real-Time Workshop without having to modify the template makefiles themselves. This feature is useful if you need to include your code when building S-functions.

See “Customizing the Makefile Include Path” in the Real-Time Workshop online documentation for details.

Target Language Compiler 4.0

TLC File Parsing Before Execution

The Target Language Compiler 4.0 completes parsing of the TLC file just before execution. This aids development because syntax errors are caught the first time the TLC file is run instead of the first time the offending line is reached.

Enhanced Speed

The Target Language Compiler 4.0 features speed improvements throughout the software. In particular, the speed of block parameter generation has been enhanced.

Build Directory

The Target Language Compiler 4.0 creates and uses a `build` directory. The `build` directory is in the current directory and prevents generated code from clashing with other files generated for other targets, and keeps your model directories maintenance to a minimum.

TLC Profiler

An entirely new TLC Profiler has been added to the Target Language Compiler to help you find performance problems in your TLC code.

model.rtw Changes

This release contains a new format and changes to the *`model`*`.rtw` file. The size of the *`model`*`.rtw` file has been reduced.

Block Parameter Aliases

Aliases have been added for block parameters in the `model.rtw` file.

Improved Text Expansion

This release of the Target Language Compiler contains new, flexible methods for text expansion from within strings.

Column-Major Ordering

Two-dimensional signal and parameter data now use column-major ordering.

Improved Record Handling

The Target Language Compiler 4.0 utilizes new record data handling.

New TLC Language Semantics

Many changes have been made to the language including:

- Improved `EXISTS` behavior (see “TLC Compatibility Issues” on page 7-13)
- New TLC primitives for record handling
- Functions can return records.
- Records can be printed.
- Records can be empty.
- Record aliases are available.
- Built-in functions cannot be undefined via `%undef`.
- Short circuit evaluation for Boolean operators, `%if`-`%elseif`-`%endif`, and `?:` expressions are handled properly
- Conversions of values to and from MATLAB. (See “FEVAL Function” in the Target Language Compiler documentation for more information.)
- Relational operators can be used with nonfinite values.
- Loop control variables are local to loop bodies.

New Built-In Functions

The following built-in functions have been added to the language.

`FIELDNAMES`, `GENERATE_FORMATTED_VALUE`, `GETFIELD`, `ISALIAS`, `ISEMPTY`, `ISEQUAL`, `ISFIELD`, `REMOVEFIELD`, `SETFIELD`

New Built-In Values

The following built-in values have been added to the language.

`INTMAX`, `INTMIN`, `TLC_FALSE`, `TLC_TRUE`, `UINTMAX`

Added Support for Inlined Code

Support has been added for two-dimensional signals in inlined code.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from the Real-Time Workshop 3.0 (Release 11.0) to the Real-Time Workshop 4.0.

Column-Major Matrix Ordering

The Real-Time Workshop now uses column-major ordering for two-dimensional signal and parameter data. In previous releases, the ordering was row-major.

If your hand-written code interfaces to such signals or parameters via `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` declarations, make sure to review any code that relies on row-major ordering, and make appropriate revisions.

Including Generated Files

Filename extensions for certain generated files have changed. If your application code uses `#include` statements to include the Real-Time Workshop generated files (such as `model.prm`), you may need to modify these statements. See “Filename Extensions for Generated Files” on page 7-8.

Updating Release 11 Custom Targets

If you have created custom targets for the Real-Time Workshop under Release 11, you must update your custom system target files and template makefiles to create and utilize the build directory. See `matlabroot/rtw/c/grt` for examples.

To update a Release 11 target:

1 Add the following to your system target file.

```
/%
BEGIN_RTW_OPTIONS
...
rtwgensettings.BuildDirSuffix = '_grt_rtw';
END_RTW_OPTIONS
%/
```

2 Add `".."` to the `INCLUDES` rule in your template makefile. The following example is from `grt_lcc.tmf`.

```
INCLUDES = -I. -I.. $(MATLAB_INCLUDES) $(USER_INCLUDES)
```

The first `-I.` gets files from the build directory, and the second `-I..` gets files (e.g., user written S-functions) from the current working directory.

Conceptually, think of the current directory and the build directory as the same (as it was in Release 11). The current working directory contains items like user written S-functions. The reason `".."` must be added to the `INCLUDES` rule is that `make` is invoked in the build directory (i.e., the current directory was temporarily moved).

3 Place the generated executable in your current working directory. The following example is from `grt_lcc.tmf`.

```
PROGRAM = ../$(MODEL).exe
$(PROGRAM) : $(OBJS) $(RTWLIB)
   $(LD) $(LDFLAGS) -o $@ $(LINK_OBJS) $(RTWLIB) $(LIBS)
```

hilite_system Replaces locate_system

If you use the `locate_system` command, in MATLAB programs for tracing the Real-Time Workshop system/block identification tags, you should use `hilite_system` instead. See "hilite_system and Code Tracing" on page 7-8.

TLC Compatibility Issues

In bringing Target Language Compiler files from Release 11 to Release 12, the following changes may affect your TLC code base:

- Nested evaluations are no longer supported. Expressions such as

  ```
  %<LibBlockParameterValue(%<myVariable>,"", "", "")>
  ```

  are no longer supported. You will have to convert these expressions into equivalent non-nested expressions.

- Aliases are no longer automatically created for Parameter blocks while reading in the Real-Time Workshop files.

- You cannot change the contents of a “Default” record after it has been created. In the previous TLC, you could change a “Default” record and see the change in all the records that inherited from that default record.
- The `%codeblock` and `%endcodeblock` constructs are no longer supported.
- `%defines` and macro constructs are no longer supported.
- Use of line continuation characters (`...` and `\`) are not allowed inside of strings. Also, to place a double quote (`"`) character inside a string, you must use `\"`. Previously, the Target Language Compiler allowed you to use `"""` to get a double quote in a string.
- Semantics have been formalized to `%include` files in different contexts (e.g., from generate files, inside of `%with` blocks, etc.) `%include` statements are now treated as if they were read in from the global scope.
- The previous the Target Language Compiler had the ability to split function definitions (and other directives) across include file boundaries (e.g., you could start a `%function` in one file and `%include` a file that had the `%endfunction`). This no longer works.
- Nested functions are no longer allowed. For example,

```
%function foo ()
  %function bar ()
  %endfunction
%endfunction
```

- Built-in functions cannot be undefined via `%undef`. It is possible to undefine built in values, but this practice is not encouraged.
- Recursive records are no longer allowed. For example,

```
Record1    {
  Val      2
  Ref      Record2
}
Record2    {
  Val      3
  Ref      Record1
}
```

- Semantics of the `EXISTS` function have changed. In the previous release of TLC, `EXISTS(var)` would check if the variable represented by the string value in `var` existed. In the current release of TLC, `EXISTS(var)` checks to see if `var` exists or not.

To emulate the behavior of `EXISTS` in the previous release, replace

```
EXISTS(var)
```

with

```
EXISTS( %<var> )
```