Real-Time Workshop® Embedded Coder™ Release Notes

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Real-Time Workshop® Embedded Coder™ Release Notes

Trademarks

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

Patents

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

Contents

Summary by Version

This table provides quick access to what is new in each version. For clarification, see "Using Release Notes" on page 2.

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V5.5 (R2010a)	Yes Details	Yes Summary	Bug Reports Includes fixes	Printable Release Notes: PDF Current product documentation
V5.4 (R2009b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V5.3 (R2009a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V5.2 (R2008b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V5.1.1 (R2008a+)	No	No	Bug Reports Includes fixes	No
V5.1 (R2008a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V5.0.1 (R2007b+)	No	No	Bug Reports Includes fixes	No
V5.0 (R2007b)	Yes Details	No	Bug Reports Includes fixes	No
V4.6.1 (R2007a+)	No	No	Bug Reports Includes fixes	No
V4.6 (R2007a)	Yes Details	No	Bug Reports Includes fixes	No
V4.5 (R2006b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
V4.4.1 (R2006a+)	No	No	Bug Reports Includes fixes	No
V4.4 (R2006a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.3 (R14SP3)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.2.1 (R14SP2+)	No	No	Bug Reports Includes fixes	No
V4.2 (R14SP2)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.1 (R14SP1)	Yes Details	No	Fixed bugs	No
V4.0 (R14)	Yes Details	Yes Summary	Fixed bugs	No
V3.2.1 (R13SP2)	Yes Details	No	Fixed bugs	V3.2.1 product documentation
V3.2 (R13SP1+)	Yes Details	No	No bug fixes	No
V3.1 (R13SP1)	Yes Details	No	No bug fixes	No

Using Release Notes

Use release notes when upgrading to a newer version to learn about:

- New features
- Changes
- Potential impact on your existing files and practices

Review the release notes for other MathWorks™ products required for this product (for example, MATLAB® or Simulink®). Determine if enhancements, bugs, or compatibility considerations in other products impact you.

If you are upgrading from a software version other than the most recent one, review the current release notes and all interim versions. For example, when you upgrade from V1.0 to V1.2, review the release notes for V1.1 and V1.2.

What Is in the Release Notes

New Features and Changes

- New functionality
- Changes to existing functionality

Version Compatibility Considerations

When a new feature or change introduces a reported incompatibility between versions, the **Compatibility Considerations** subsection explains the impact.

Compatibility issues reported after the product release appear under Bug Reports at The MathWorks™ Web site. Bug fixes can sometimes result in incompatibilities, so review the fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

The MathWorks offers a user-searchable Bug Reports database so you can view Bug Reports. The development team updates this database at release time and as more information becomes available. Bug Reports include provisions for any known workarounds or file replacements. Information is available for bugs existing in or fixed in Release 14SP2 or later. Information is not available for all bugs in earlier releases.

Access Bug Reports using your MathWorks Account.

Version 5.5 (R2010a) Real-Time Workshop Embedded Coder Software

This table summarizes what is new in Version 5.5 (R2010a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary	Bug Reports Includes fixes	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are:

Customize Generated Code Modules Using File Packaging Format Configuration Parameter

R2010a introduces a new configuration parameter, "File packaging format", on the **Code Placement** pane of the Configuration Parameters dialog box. This parameter provides several options to reduce the number of files generated. See "Customizing Generated Code Modules ", for more information.

Code Generation Verification Improvements

R2010a includes additional functionality to the Code Generation Verification (CGV) API. The cgv.CGV class now contains the following methods:

- `getSavedSignals` returns a list of signal names from the input data
- `createToleranceFile` creates a file correlating tolerance information with signal names.
- `compare` compares like-named signals between two models
- `plot` creates a plot of a provided list of signals

See "Verifying Numerical Equivalence with Code Generation Verification" for more information on how to use these methods for code generation verification.

AUTOSAR Compiler Abstraction Macro Generation

R2010a provides a new option to generate AUTOSAR code with compiler abstraction macros. If you select this option, the software generates code with C macros containing platform-independent compiler directives.

For more information, see "Generating AUTOSAR Compiler Abstraction Macros" in the Real-Time Workshop® Embedded Coder™ documentation.

Compatibility Considerations

In R2010a, new reserved keywords for AUTOSAR support are:

- `FUNC`
- `VAR`
- `CONST`
- `P2VAR`
- `P2CONST`

If your AUTOSAR model (system target file `autosar.tlc`) specifies any of these keywords for the name of a variable or function to be generated, then R2010a software produces an error when you build the model. You must reconfigure your model to replace these keywords with names that are not reserved keywords.

AUTOSAR Enhancements

SIL and PIL Simulation Support

SIL and PIL simulation modes now support verification of AUTOSAR code generated from top models.

For more information about this option, see "Verifying the AUTOSAR Code with SIL and PIL Simulations" in the Real-Time Workshop Embedded Coder documentation.

Software-in-the-Loop Simulation Mode

New Software-in-the-Loop (SIL) simulation mode allows you to run top models or Model blocks as a SIL simulation. With this mode, you can:

- Verify generated source code without modifying the original model.
- Reuse test harnesses for the original model with the generated source code.

This mode, which executes code on the host computer, also provides a convenient alternative to PIL simulation when target hardware is not available.

For more information, see "Verifying Code Using the SIL Simulation Mode" in the Real-Time Workshop Embedded Coder documentation.

For examples of SIL verification, see the demo `rtwdemo_sil_pil_script`(previously called `rtwdemo_sil_pil` in R2009b).

Compatibility Considerations

The new Software-in-the-Loop (SIL) simulation mode replaces host-based PIL simulation. R2010a software generates an error if all of the following is true about your model:

- The model has been saved using the R2009b, or earlier, software.
- The simulation mode is set to PIL.
- The model is configured correctly for host-based PIL simulation using R2009b software.

The error message is:

```
ModelName is not correctly configured for PIL simulation. The model was
last saved in an older release and is configured for "host-based PIL"
simulation. "Host-based PIL" simulation is no longer available and has
been replaced by SIL simulation. If ModelName is referenced by a Model
block running in PIL mode, you must change the simulation mode of the
Model block to SIL. If ModelName is a top model running in PIL mode, you
must change the top-model simulation mode to SIL. Re-run the simulation
when you have changed the simulation mode.
```

`ModelName` is the name of a model or the path to a Model block in PIL mode.

Code Coverage Measurement for Generated Code

You now have a new option to measure code coverage using the BullseyeCoverage™ tool from Bullseye Testing Technology™. When you run a top-model or Model block SIL simulation, this option allows you to collect code coverage metrics for generated source code.

For more information, see "Using a Code Coverage Tool in a SIL Simulation" in the Real-Time Workshop Embedded Coder documentation.

Specify Bitfield Declarator Type Using New Configuration Parameter

R2009b introduced a new optimization to "Pack Boolean data into bitfields" on the Optimization pane of the Configuration Parameters dialog box. Selecting this option now enables a new configuration parameter, "Bitfield declarator type specifier", to specify the data type for bitfields as `unsigned int` or `unsigned char`. The optimization benefit of selecting a specific data type is dependent on your target.

Efficiency Objective for Code Generation Separated into Execution, ROM, and RAM

Previously, you could select `Efficiency` as your code generation objective. The Code Generation Advisor provided advice for configuration parameters that were common to RAM, ROM, and execution efficiency objectives. If a parameter was not common to all three objectives, the Code Generation Advisor did not provide advice for that parameter.

In R2010a, the code generation efficiency objective is separated into three objectives: `Execution efficiency`, `ROM efficiency`, and `RAM efficiency`. The new efficiency objectives allow you to explicitly define your goal for generating code. The Code Generation Advisor provides advice on configuration parameters that are specific to the objectives that you specify, resulting in more complete advice.

For details, see "Mapping Application Objectives to Model Configuration Parameters".

Compatibility Consideration

When you open the Configuration Parameters dialog box, if you previously specified `Efficiency` for the code generation objective, the software replaces the efficiency objective with the following priority of the new objectives:

1. Execution efficiency
2. ROM efficiency
3. RAM efficiency

The priority of the new objectives causes the Code Generation Advisor to provide advice that is similar to the obsolete `Efficiency` objective.

If you previously created a custom objective based on the `Efficiency` objective, the custom objective contains the same parameters and checks that it did when you created it.

Generated Code Enhancements

In R2010a, code generation enhancements are the following:

- Enables in-place read and write operations on a data store memory. The generate code operates on the data store memory directly without making extra copies. This optimization uses less RAM and ROM, and executes faster.
- Eliminates unnecessary DWorks associated with an Assignment block in an iterator subsystem. The generated code uses less global memory.
- Uses local variables instead of a global DWork for a Unit Delay or Memory block when their state value is not persistent. For example, when the blocks are in an Iteration system with the block parameter **States when starting** set to `reset`. The generated code uses less global memory and executes faster.
- Eliminates a data copy when passing a submatrix, containing contiguous data, to a function. For example, this optimization occurs when a submatrix

signal is selected through a Selector block and passed into a Stateflow® chart. The generated code now passes the address of the first element of the submatrix to the function. This optimization generates less code, uses less RAM, and executes faster.

- Improve code reuse of identical subsystems in different instances when the subsystem arguments are individually passed by reference and are different storage classes. This optimization reduces ROM usage of the generated code.
- Improves handling of if-else patterns for enabled subsystem pairs, which are driven by enabled signals and the values of the two signals are the opposite at any time. In the generated code, this optimization uses local variables instead of global variables for the enabled subsystem outputs. This optimization uses less RAM and ROM and executes faster.
- Improves expression folding to reduce data copies in a Bus Assignment and Switch block pattern. This optimization improves the execution speed of the generated code.
- Improves C code generation for Embedded MATLAB® code when handling concatenated arrays or matrix signals. The generated code uses less RAM and ROM, and executes faster.
- Uses a local variable instead of global data in the generated code for the output of a While iterator subsystem. This optimization occurs when this system executes at least once for all time steps.
- Enables inlining of generated code for Stateflow charts, even if the generated code calls a subfunction which accesses global Simulink data. This optimization removes a limitation of inlining Stateflow code.
- Improves handling of vector operations. Identifies more opportunities for merging similar loops and replacing indexed vector signals with scalar variables. The generated code uses less RAM and ROM, and executes faster.
- Generates more efficient code for enabled subsystems by
  - Optimizing away conditional mode-checking code when it is superfluous.
  - Replacing a SUBSYS_ENABLED/SUBSYS_DISABLED enumerated type with a Boolean type.

Model Variants Improvements for Code Generation

Previously, when you used a `Simulink.Parameter` object of enumerated type for variant object conditions, the generated code only included a check that the variables were defined. In R2010a, the generated code now includes preprocessor conditionals to check that the variant object condition variables contain valid values of the enumerated type definition. See "Variant Models and Enumerated Types" and "Defining Variant Control Variables and Variant Objects for Generating Code" for more information.

The Code Variants Report in the HTML Code Generation Report now displays the variant objects in alphabetical order for better readability.

Code Generation Template Enhancements for Function and File Banner Customization

Improvements to the function and file banner customization for the generated code include:

- File banner generation in `rtwtypes.h`.
- A new tag attribute, `width`, to specify the width of both function and file banners. See "Customizing a Code Generation Template (CGT) File for File and Function Banner Generation" for more information.
- A token attribute, `style`, for the function banner token, `BlockDescription`. This attribute provides finer control of the content of the block description comments in the generated code. See "Function Banner" for more information.

Target Function Library Enhancements

R2010a provides several enhancements to target function libraries (TFLs), with corresponding updates to the demo `rtwdemo_tfl_script`.

C++ Name Spaces Supported

R2010a adds a `C++ (ISO)` math library to the set of target-specific math libraries you can select using the **Target function library** parameter on the **Real-Time Workshop > Interface** pane of the Configuration Parameters dialog box. (For more information, see "New C++ (ISO) Math Library" in the Real-Time Workshop® release notes.)

In addition, you can now specify C++ name spaces on TFL replacement functions. R2010a provides a programmatic interface that allows you to register a TFL entry with the implementation defined in a C++ name space. If the conceptual signature is matched for the entry, the C++ name space is emitted to the generated code. For more information, see the reference pages for the new TFL table creation functions registerCPPFunctionEntry, enableCPP, and setNameSpace.

Ability to Create Custom TFL Entries

TFLs now support custom entries that allow you to specify near-arbitrary match criteria. You first create your own TFL entry class, derived from either `RTW.TflCFunctionEntryML` (for function replacement) or `RTW.TflCOperationEntryML` (for operator replacement). In your derived class, you implement a `do_match` method with a fixed preset signature and customize the match criteria. You also can modify the implementation signature to meet your application needs. For more information, see "Refining TFL Matching and Replacement Using Custom TFL Table Entries" in the Real-Time Workshop Embedded Coder documentation.

Enhanced Support for Scalar Operator Replacement

R2010a adds TFL support for replacing scalar complex operations, including addition, subtraction, multiplication, cast, and complex conjugation. Mixed types are supported. For more information, see "Example: Mapping Scalar Operators to Target-Specific Implementations".

Additionally, you can now replace fixed-point shift right for all fixed-point input types. For more information, see "Mapping Fixed-Point Operators to Target-Specific Implementations".

Note Real-Time Workshop Embedded Coder documentation provides a current list of all operators supported or replacement with custom library functions using TFL tables.

Enhanced Support for Nonscalar Operator Replacement

R2010a adds TFL support for replacing Hermitian and complex conjugate operations for nonscalar inputs.

Additionally, you can now replace a transposition or Hermitian operation combined with matrix multiplication with a single BLAS function call.

For more information, see "Mapping Nonscalar Operators to Target-Specific Implementations".

Data Type Replacement Extended to Map Booleans to Signed Integers

R2010a extends data type replacement to allow mapping the `boolean` built-in data type to any of the following integer types:

- `int8`
- `uint8`
- `int`*`n`*
- `uint`*`n`*

where *n* is 8, 16, or 32, matching the integer word size for your production hardware (for example, `int32` for 32-bit hardware). This can increase the execution speed of generated code on some targets. For more information, see "Replacing `boolean` with an Integer Data Type" in the Real-Time Workshop Embedded Coder documentation.

Exported Function Code Enhanced to Allow Bit Packing

To support multithread deployment of exported functions, generated code for exported functions has been enhanced to allow multithread-safe bit packing.

Enhanced Multiple-Instance Subsystem Code

R2010a removes some limitations on code generation for ERT multiple-instance models. The following forced-nonreusable subsystems are now allowed in ERT multiple-instance models:

- Subsystems with noninlined S-functions as triggers
- Subsystems with wide function-calls as triggers

Obsolete Header File rtw/c/ert/ertformat.h Removed

R2010a removes the obsolete static header file `rtw/c/ert/ertformat.h`. Macros equivalent to the macros defined by `ertformat.h` have been automatically generated since R13.

Compatibility Considerations

If you have static C files authored before R13, such as a static `main.c`, and if the files reference the header file `rtw/c/ert/ertformat.h`, you must modify the files as follows:

1. Include the file `rtwtypes.h` in your C code (for example, `#include "rtwtypes.h"`).
2. Replace all occurrences of the macro name `ssGetErrorStatus` with `rtmGetErrorStatus`, and replace all occurrences of the macro name `ssSetErrorStatus` with `rtmSetErrorStatus`.

Name Change to the Real-Time Workshop > Data Placement Pane

In R2010a, the **Real-Time Workshop > Data Placement** pane in the Configuration Parameter dialog box is now the **Real-Time Workshop > Code Placement** pane to better describe existing and new functionality of the parameters on this pane. See "Real-Time Workshop Pane: Code Placement" for more information.

MISRA C Model Advisor Checks

The Simulink Model Advisor includes the following new MISRA C® compliance checks:

- "Check for blocks not recommended for MISRA-C:2004 compliance"
- "Check configuration parameters for MISRA-C:2004 compliance"

For more information about the Model Advisor, see "Consulting the Model Advisor" in the Simulink documentation.

New and Enhanced Demos

The following demos have been added in R2010a:

Demo...	Shows How You Can...
`rtwdemo_cgv`	Compare a model in different simulations using the Code Generation Verification API.

The following demos have been enhanced in R2010a:

Demo...	Now...
`rtwdemo_tfl_script`	Illustrates new target function library (TFL) capabilities, such as custom TFL table entries and enhanced support for scalar and nonscalar operator replacement.
`rtwdemo_rtwecintro`	Illustrates using code generation objectives to configure models.
`rtwdemos` > **Guided Tutorials > Step-by-Step Code Generation Process > Understanding the Model**	Illustrates: • Using code generation objectives to configure models. • Saving configuration sets using the `saveAs` method of the `Simulink.ConfigSet` class.

Version 5.4 (R2009b) Real-Time Workshop Embedded Coder Software

This table summarizes what is new in Version 5.4 (R2009b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary	Bug Reports Includes fixes	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are

Improved User Guide Documentation

The *Real-Time Workshop Embedded Coder User's Guide* has been reorganized and consolidated to better support user workflows.

Code Generation Objectives Now Customizable

Previously, you could run the Code Generation Advisor based on four predefined objectives: efficiency, traceability, safety precaution, and debugging. In R2009b, you can create customized objectives and use the Code Generation Advisor to review models against the custom objectives.

You can use the Code Generation Objective API to create custom objectives by:

- Creating a new objective and adding parameters and checks to a new objective.
- Creating a new objective based on an existing objective, then adding, modifying and removing the parameters and checks within the new objective.

For details, see "Creating Custom Objectives" in the Real-Time Workshop Embedded Coder *User's Guide*.

Code Generation Objectives Limitation Removed

Previously, the code generator did not support reviewing referenced models during the build process. In R2009b, The MathWorks removes this limitation. In the Configuration Parameters dialog box, on the **Real-Time Workshop** pane, select **Check model before build** to review the top model and referenced models during the build process.

For details, see "Reviewing Objectives in Referenced Models" in the Real-Time Workshop Embedded Coder *User's Guide*.

Verify Numerical Equivalence of Results

In R2009b, you can use Code Generation Verification (CGV) to verify the numerical equivalence of results when you execute a model in different modes of execution. CGV supports executing the model in simulation, Software-In-the-Loop (SIL), and Processor-In-the-Loop (PIL).

For details, see "Verifying Numerical Equivalence of Results with Code Generation Verification API" in the Real-Time Workshop Embedded Coder documentation.

Automatically Verify a SIL or PIL Configuration

You might need to change model settings to configure the model correctly for SIL or PIL. To find out what settings you must change, you can use the `cgv.Config` class. Using the `cgv.Config` class, you can review your model configuration and determine which settings you must change to configure the model correctly for SIL or PIL.

For details, see “Verifying a SIL or PIL Configuration” in the Real-Time Workshop Embedded Coder documentation.

New Top-Model Processor-in-the-Loop (PIL) Simulation Mode

New Processor-in-the-Loop (PIL) simulation mode allows you to run a complete model as a PIL simulation on your target processor, debugger, or instruction set simulator. With this mode, you can verify the object code that is generated and compiled from a complete model without creating a separate test harness model. This provides an alternative to block-based approaches where the PIL simulation is performed using a test harness model that contains a block executing in PIL mode (e.g. Model block PIL).

Additionally, by selecting your host machine as the target hardware, you can use the PIL simulation mode to perform Software-in-the-Loop (SIL) verification. Again, this provides an alternative to block-based approaches for SIL.

For more information on this new mode (and how it differs from the block-based PIL and SIL), see “Verifying Compiled Object Code with Processor-in-the-Loop Simulation” and “Choosing a PIL Approach” in the Real-Time Workshop Embedded Coder documentation.

For examples of SIL and PIL verification, see the new demo rtwdemo_sil_pil.

Processor-in-the-Loop Enhancements

Bus Support

Processor-in-the-loop (PIL) now provides support for buses.

For more information on PIL bus support, see “I/O Support ” in the PIL support tables in the Real-Time Workshop Embedded Coder documentation.

Data Stores and Imported Data Definition

PIL now provides support for global data stores and grouped custom storage classes.

PIL support for imported data definition is improved. You can use signals, parameters, data stores, etc., that specify storage classes with imported data definitions. For exceptions, see "Imported Data Definitions" in the PIL support tables.

For more information on PIL support, see "SIL and PIL Simulation Support and Limitations" in the Real-Time Workshop Embedded Coder documentation.

PIL Block Behavior with Goto/From Blocks

You will now see an error if your PIL component includes any Goto / From blocks that cross the boundary of the PIL component.

Previously it was possible but not recommended to use Goto/From blocks for I/O data that crosses the boundary of the PIL block component. For virtual (nonatomic) subsystems, the right-click PIL build transformed boundary-crossing Goto blocks into outports and From blocks into inports. The resulting PIL block had extra I/O ports and you had to rework the model to connect it. This behavior has changed to an error if your PIL component includes any Goto / From blocks that cross the boundary of the PIL component.

Ability to Generate Switch-Case Statements for Embedded MATLAB Function Blocks and Stateflow Charts

You can choose to generate `switch-case` statements during code generation when you have `if-elseif-else` decision logic in one of the following:

- An Embedded MATLAB Function block in a Simulink model
- A flow graph or an Embedded MATLAB function in a Stateflow chart

`Switch-case` statements provide more readable and efficient code than `if-elseif-else` statements when multiple decision branches are possible.

When you load models created in R2009a and earlier, this optimization is off to maintain backward compatibility. In previous versions, `if-elseif-else` logic appeared unchanged in generated code. For more information, see:

- "Real-Time Workshop Pane: Code Style" in the Real-Time Workshop Embedded Coder documentation
- "Enhancing Readability of Generated Code for Embedded MATLAB Function Blocks" in the Simulink documentation
- "Enhancing Readability of Generated Code for Flow Graphs" in the Stateflow documentation

Target Function Library Enhancements

TFLs Now Support Function Replacement for Nonscalar Operators

Target function libraries (TFLs) now support the following nonscalar operators for replacement with custom library functions:

`+` (addition)
(subtraction)
`*` (matrix multiplication)
`.*` (array multiplication)
`'` (matrix transposition)
`.'` (array transposition)

For more information, see "Mapping Nonscalar Operators to Target-Specific Implementations" in the Real-Time Workshop Embedded Coder documentation and the TFL demos page `rtwdemo_tfl_script`.

Ability to Use BLAS Functions for Matrix Multiplication

TFLs now support the ability to map matrix multiplication operations to Basic Linear Algebra Subroutine (BLAS) functions.

For more information, see the BLAS examples in "Mapping Nonscalar Operators to Target-Specific Implementations" in the Real-Time Workshop Embedded Coder documentation and the TFL demos page `rtwdemo_tfl_script`.

Support for Data Type Conversion (Cast) Replacement

TFLs now support function replacement for data type conversion (cast) operations.

For more information, see "Mapping Data Type Conversion (Cast) Operations to Target-Specific Implementations" in the Real-Time Workshop Embedded Coder documentation.

Support for Fixed-Point Shift Left Replacement

TFLs now support function replacement for fixed-point << (shift left) operations.

For more information, see "Mapping Fixed-Point Shift Left Operations to Target-Specific Implementations" in the Real-Time Workshop Embedded Coder documentation.

Support for Replacing Additional Math Functions

TFLs now support function replacement for the following math functions.

`acosh`	`fix`	`min`	`saturate`
`asinh`	`hypot`	`mod/fmod`	`sign`
`atan2`	`ldexp`	`rem`	
`atanh`	`ln`	`round`	
`exactrSqrt`	`max`	`rSqrt`	

Additionally, replacement of the `abs` function now works with integer arguments as well as floating-point arguments.

For more information, see "Example: Mapping Math Functions to Target-Specific Implementations" in the Real-Time Workshop Embedded Coder documentation.

C++ Encapsulation Enhancements

Control Visibility and Access for Data Members In C++ Model Class Using New Configuration Parameters

R2009b adds five model configuration parameters that provide enhanced control over visibility and access for data members in a generated C++ model class. If you select the `ert.tlc` target and the `C++ (Encapsulated)` language for your model, the following new parameters appear on the **Interface** pane of the Simulink Configuration Parameters dialog box:

"Block parameter visibility"
"Internal data visibility"
"Block parameter access"
"Internal data access"
"External I/O access"

Compatibility Considerations. The new parameters replace the following obsolete model configuration parameters, which were supported in R2008b and R2009a, and which have been removed from the **Interface** pane:

Private parameter and state members
Parameter and state access methods
I/O access methods
Inline access methods

To support forward and backward compatibility,

- If you open a model created in R2008b or R2009a with R2009b, the obsolete parameters are automatically converted to the new parameters.
- If you open a model in R2009b and use **Save As** to save the model in R2008b or R2009a format, the new parameters are converted to the R2008b/R2009a parameters.

Select Dynamic Memory Allocation for Registering Referenced C++ Encapsulation Models Using New Configuration Option

R2009b adds the model configuration option **Use operator new for referenced model object registration**. This option specifies whether generated code should use the operator `new`, during model object registration,

to instantiate objects for referenced models configured with a C++ encapsulation interface.

If you select the `ert.tlc` target and the `C++ (Encapsulated)` language for your model, **Use operator new for referenced model object registration** appears on the **Interface** pane of the Simulink Configuration Parameters dialog box. If you select this option, the model build process generates code that uses dynamic memory allocation to instantiate objects for referenced models configured with a C++ encapsulation interface. Specifically, during instantiation of an object for the top model in a model reference hierarchy, the generated code uses `new` to instantiate objects for referenced models.

Selecting this option frees a parent model from having to maintain information about submodels beyond its direct children. Clearing this option means that a parent model maintains information about all of its submodels, including its indirect children as well as its direct children.

For more information, see **"Use operator new for referenced model object registration"** in the Real-Time Workshop Embedded Coder reference documentation.

Function Subsystems and Charts Now Generated as Private Member Functions in C++ Encapsulation Model Class

R2009b enhances the C++ encapsulation code generated for Simulink subsystems and Stateflow charts that are not required or configured to be reusable — that is, subsystems or charts for which the **Real-Time Workshop system code** parameter is set to `Function`.

In previous releases, the build process generated subsystem or chart functions as global reusable functions outside of the class structure, regardless of whether the **Real-Time Workshop system code** parameter was set to `Function` or `Reusable function`. If your coding standard required subsystem or chart functions to be encapsulated in a class interface, the generated code would not meet the requirement. Additionally, if the model required only a single instance of the subsystem or chart, generating a global reusable function added unnecessary memory and processing overhead.

Beginning in R2009b, the build process generates `Function` subsystems and charts as private member functions in the C++ encapsulation model class.

Compatibility Considerations. Beginning in R2009b, if `C++ (Encapsulated)` is selected as the **Language** option for your model, and if your model contains a Simulink subsystem or Stateflow chart for which the **Real-Time Workshop system code** parameter is set to `Function`, the build process generates the subsystem or chart function as a private member function in the C++ encapsulation model class, rather than as a global reusable function outside of the class structure. If you want the subsystem or chart function to be generated as a global reusable function outside of the class structure, go to the subsystem or chart parameters dialog box and set the **Real-Time Workshop system code** parameter to `Reusable function`.

Limitations.

- This enhancement only applies to `Function` subsystems and charts in the top model, not in referenced models.
- If a chart contains a subfunction, the chart is generated as a global reusable function outside of the class structure rather than as a private member function in the C++ encapsulation model class.

Mismatched System Code Options for Nested Subsystems or Charts Now Error Out During C++ Encapsulation Code Generation

When generating C++ encapsulation code for a model, the build process now flags an error if a Simulink subsystem or Stateflow chart for which the **Real-Time Workshop system code** parameter is set to `Function` is nested in a subsystem or chart for which the **Real-Time Workshop system code** parameter is set to `Reusable function`.

Compatibility Considerations. In previous releases, mismatched **Real-Time Workshop system code** parameter values between a nested subsystem or chart and its containing subsystem or chart did not generate an error, because C++ encapsulation code generation would generate both `Function` and `Reusable function` subsystems or charts as global reusable functions outside of the model class structure.

Beginning in R2009b, C++ encapsulation code generation generates `Function` subsystems and charts as private member functions in the model class, so you

can no longer nest a `Function` subsystem or chart in a `Reusable function` subsystem or chart.

To resolve the conflict, do one of the following:

- Set **Real-Time Workshop system code** to `Reusable function` for the nested subsystem or chart. The subsystems or charts will generate global reusable functions outside of the class structure.
- Set **Real-Time Workshop system code** to `Function` for the containing subsystem or chart. The subsystems or charts will generate private member functions in the model class.

AUTOSAR Enhancements

Multi-Argument Client-Server Communication

Support is now provided for multi-argument AUTOSAR client-server communication. You can generate AUTOSAR-compliant code, and import and export AUTOSAR description XML files for Application Software Components and Basic Software. For more information, see "Configuring Client-Server Communication" in the Real-Time Workshop Embedded Coder documentation.

The new demo rtwdemo_autosar_clientserver_script shows how you can generate and export AUTOSAR-compliant code and XML files for a Simulink model with an AUTOSAR client-server interface.

Merging Inter-Runnable Variables

You can now use the Merge block to merge inter-runnable variables. However, there are constraints on the signal names for the Merge block and the destination of its output. For more information on these constraints, see "Using the Merge Block for Inter-Runnable Variables" in the Real-Time Workshop Embedded Coder documentation.

Updating Model Configuration Set

If you saved an AUTOSAR model configuration set as a MAT-file using a previous release of Real-Time Workshop Embedded Coder, then you must run `slupdate` on the model to update the referenced configuration set. After

this, resave the configuration set (as a MAT-file). For more information, see "Setting Up Configuration Sets", "Referencing Configuration Sets", and `slupdate` in the Simulink documentation.

Limitations

You cannot use the new Invoke AUTOSAR Server Operation block within a referenced model. If a referenced model has this block, the software produces a warning when you build the top model. You can only use the Invoke AUTOSAR Server Operation block in the top model.

Pack Boolean Data Into Bitfields in the Generated Code Using New Configuration Parameter

R2009b introduces a new configuration parameter to pack Boolean data into bitfields in the generated code. Choosing this option reduces global RAM usage. For details, see "Pack Boolean data into bitfields" in the Simulink documentation.

Generate Preprocessor Conditional Directives Using New Configuration Parameter

Previously, there was no built-in solution for implementing variants in the generated code. In R2009b, you can generate preprocessor conditional directives using model reference variants. The new configuration parameter **Generate preprocessor conditionals** provides the capability to implement variants globally for a model or locally for each referenced model. For details, see "Generate preprocessor conditionals" in the Real-Time Workshop documentation and "Generating Code Variants for Variant Models" in the Real-Time Workshop Embedded Coder documentation.

Specify Function Argument Names Using New Configuration Parameter

R2009b adds a new **Identifier format control** parameter, "Subsystem method arguments" on the **Real-Time Workshop > Symbols** pane of the Configuration Parameters dialog box. This parameter provides finer control over the naming rules for function argument names in reusable subsystems. To be able to use the new parameter, your model must use an ERT-based

system target file (requires a Real-Time Workshop Embedded Coder license). For a description of the **Identifier format control** parameters and their use, see "Specifying Identifier Formats" in the Real-Time Workshop Embedded Coder documentation.

Remove or Preserve extern Keyword In Generated Code Function Declarations Using New Configuration Parameter

Previously, the Real-Time Workshop software automatically included the `extern` keyword in function declarations in the generated code. In R2009b, a new configuration parameter provides you with the flexibility to remove the `extern` keyword from function declarations. For details, see "Preserve extern keyword in function declarations" in the Real-Time Workshop Embedded Coder documentation.

New Function Banner Customization and Code Generation Template Improvements

R2009b introduces a unified approach to customizing and sharing file and function banner templates in the "Code Generation Template (CGT) Files":

- Custom function banner generation provides consistency in banner comments for functions and shared utility functions.
- Tag format to more easily create file and function banners.
- Predefined tokens include specific information about each function in the function banner.
- Predefined styles provide a consistent format for the boundary of file and function banners throughout your code.

HTML Code Generation Report Provides Hyperlink to Requirements Document

Previously, to trace requirements descriptions in the generated code to the requirements document for that block, you had to perform multiple steps. In R2009b, the code generation report provides hyperlinks directly to the corresponding requirements document. To include requirements descriptions in the generated code, you need a Simulink Verification and Validation™

software license. For details, see “Including Requirements Information with Generated Code” in the Simulink Verification and Validation documentation.

Code Traceability Improvement

Previously, hyperlinks in the HTML code generation report might not work if a model contains international characters and is open on a machine with different character encoding. In R2009b, traceability in the code generation report is enhanced to remove this limitation.

Tunable Iteration Limit for the For Iterator block

Previously, the **Iteration limit** parameter of the For Iterator block appeared only as a number in the generated code. The **Iteration limit** parameter now supports storage classes. The named constant representing the iteration limit appears in the generated code when you define it as a `Simulink.Parameter` object in the base workspace. This capability improves the readability and reusability of the generated code. For details, see the **Iteration limit** parameter description in the For Iterator block.

Generated Code Enhancements

In R2009b, code generation is enhanced to:

- Eliminate unnecessary data copies for the Bus Creator block in referenced models. This optimization reduces RAM usage and improves execution speed of the generated code.
- Eliminate unnecessary data copies from a subsystem output port to the root Outport block for the following subsystems:
  - Nonreusable
  - Function call
  - Conditional
  - Triggered

  This optimization reduces RAM usage and improves execution speed of the generated code.

- Combine similar loops which contain structure member read-only references. This optimization improves execution speed and likely reduces RAM and ROM usage of the generated code.
- Copy data between two structure data types using the assignment operator. Previously, the generated code copied data between two structure data types member by member. This optimization generates less code and improves execution speed for block I/O.
- Reduce global data copies by using local variables for the Vector Concatenate and Matrix Concatenate block outputs whenever possible. This optimization reduces RAM usage and improves execution speed of the generated code.
- Remove unreachable code from nested if statements containing the same test condition. This optimization generates less code and improves execution speed of the generated code.
- Eliminate `If` or `Switch` statements when the condition expression is side-effect free and all branches are identical. This optimization improves code readability, reduces code size, and reduces execution time.

Removed Limitations for Reusable Subsystem Optimization

Previously, the **Pass reusable subsystem outputs as** parameter on the Optimization pane of the Configuration Parameters dialog box had several limitations for the **Individual arguments** option. In R2009b, all limitations have been addressed, except for signals with variable dimensions.

Generated rtwtypes.h Header File Now Omits Unnecessary Type Definitions

Real-Time Workshop Embedded Coder generated production code for the header file `rtwtypes.h` is now optimized to remove unnecessary type definitions. The build process now conditionally generates type definitions and defines based on built-in integer data types, which are needed only when you select MAT-file logging or the C API data exchange interface for your model.

Duplicate Block Comments No Longer Appear in the Generated Code

Previously, if a block includes a description or a requirement, duplicate block comments might appear in the generated code for certain blocks. In R2009b, the Real-Time Workshop Embedded Coder software no longer generates duplicate block comments. See "Adding Custom Comments to Generated Code" for more information on how to include block comments into the generated code.

Block Comments for Custom Blocks are Now Configurable

Previously, block comments for custom blocks were inserted in the generated code by manually inserting comments into the associated TLC files. In R2009b, the code generation process automatically inserts block comments into the generated code for custom blocks. See "Adding Custom Comments to Generated Code" for more information. Note that the code generation process continues to emit manually inserted comments instead of the automatically generated comments. Consider removing existing block comments from your TLC files. Manually inserted comments might be poorly formatted in the generated code and code-to-model traceability might not work.

Name Improvement for Block Parameter Structure in Generated Code

Previously, if you add a Model block to your model, the Real-Time Workshop software changes the name of the instance of a block parameter structure in the generated code to a generic name, for example, `rtP`. In R2009b, naming of the block parameter structure follows the global variable naming rule. See "Identifier Format Control Parameter Values" in the Real-Time Workshop Embedded Coder documentation for more information.

Generate Code for Variable-Size Signals Using New Configuration Parameter

R2009b supports code generation for variable-size signals. You can turn on/off this capability from a new parameter **support variable-size signals** on the **Real-Time Workshop > Interface** pane of the Configuration Parameters

dialog box. This parameter only appears if you select the `ert.tlc` system target file.

New and Enhanced Demos

The following demos have been added in R2009b:

Demo...	Shows How You Can...
`rtwdemo_autosar_clientserver_script`	Configure and generate AUTOSAR-compliant code and export AUTOSAR-compliant XML files for a Simulink model with an AUTOSAR client-server interface.
`rtwdemo_preprocessor_script`	Configure model variants and generate preprocessor conditionals to control which code is linked to the embedded executable.
`rtwdemo_sil_pil`	Carry out software-in-the-loop (SIL) or processor-in-the-loop (PIL) code verification.

The following demos have been enhanced in R2009b:

Demo...	Now...
`rtwdemo_comments`	Includes a code generation template file to illustrate how to customize file banner, function banner, and file trailer comments in the generated code.
`rtwdemo_international` (see "Character Set Limitation" in the Real-Time Workshop documentation)	Includes a code generation template file to illustrate how to customize file banner, function banner, and file trailer comments in the generated code.
`rtwdemo_tfl_script`	Includes models that illustrate how to use target function libraries (TFLs) to replace nonscalar operators and utilize Basic Linear Algebra Subroutine (BLAS) library functions in generated code

Version 5.3 (R2009a) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 5.3 (R2009a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary	Bug Reports Includes fixes	No

New features and changes introduced in this version are

Ability to Specify Code Generation Objectives

Previously, setting up your model for code generation was difficult due to the complexity of options and settings available. In R2009a, you can specify high-level code generation objectives and run Model Advisor checks to identify changes to model constructs and settings that improve the generated code.

The high-level code generation objectives available in R2009a are:

- Efficiency (ERT-based targets)
- Safety precaution (ERT-based targets)
- Traceability (ERT-based targets)
- Debugging (GRT- or ERT-based targets)

After specifying your code generation objectives, you can run Model Advisor checks before generating code to identify suggested changes to your model and make the changes using the Model Advisor interface. The generated code includes comments identifying which high-level objectives you specified, and the results of the Model Advisor run at the time of code generation.

The new `rtwdemo_usingrtwec` demo shows you how you can specify code generation objectives.

For details, see "Mapping Application Objectives to Model Configuration Parameters" in the Real-Time Workshop Embedded Coder *User's Guide*. For limitations that apply, see Limitations on Code Generation Objectives.

Ability to Specify Custom Storage Classes as Signal Properties

In previous releases, applying a custom storage class to a signal required creating a `Simulink.Signal` object in the base workspace, specifying the custom storage class and its custom attributes in the object, and resolving the signal to the object in some way.

A signal object that specifies a custom storage class can be applied to at most one signal. This restriction can cause a proliferation of base workspace signal objects that apply the same custom storage class to multiple signals. In addition, base workspace signal objects cannot be saved with a model; they must be saved in a separate file and loaded whenever the model is loaded.

To overcome these disadvantages, R2009a allows you to apply a custom storage class to a signal by specifying the storage class in the Signal Properties dialog box that defines the signal. This technique creates an embedded signal object that is associated with the port where the signal originates and does not appear in any workspace. See "Applying CSCs to Parameters and Signals" for details.

PIL Enhancements

Processor-in-the-loop (PIL) Model block simulation mode now supports tunable parameters and serial communication between host and target.

Tunable Parameters

For information see "Tunable Parameters and SIL/PIL" in the Real-Time Workshop Embedded Coder documentation.

The demo `rtwdemo_pil` has been enhanced to demonstrate the use of tunable parameters during a PIL simulation.

Serial Communication

For information see "Communications rtiostream API" in the Real-Time Workshop Embedded Coder documentation.

The demo `rtwdemo_rtiostream` has been enhanced to include configuring for serial communication.

PIL Block Behavior with Goto/From Blocks

Currently, it is possible but not recommended to use Goto/From blocks for I/O data that crosses the boundary of the PIL block component. For virtual (nonatomic) subsystems, the right-click PIL build transforms boundary-crossing Goto blocks into outports and From blocks into inports. The resulting PIL block has extra I/O ports and you must rework the model to connect it.

Starting in the next release this behavior will change and you will see an error if your PIL component includes any Goto/From blocks that cross the boundary of the PIL component.

For more information on PIL support, see "SIL and PIL Simulation Support and Limitations" in the Real-Time Workshop Embedded Coder documentation.

C++ Encapsulation Interface Support for Referenced Models

R2008b introduced the ability to generate a C++ class interface to model code, in which required model data is encapsulated into C++ class attributes and model entry point functions are encapsulated into C++ class methods. However, in a model reference hierarchy, you could generate a C++ encapsulation interface only for the top model. If you built a model containing a Model block whose referenced model had `C++ (Encapsulated)` selected as the **Language** setting, the build reported an error.

Beginning in R2009a, you can use C++ encapsulation interfaces throughout a model reference hierarchy. You can now:

- Select `C++ (Encapsulated)` as the **Language** setting for referenced models in a model reference hierarchy, and configure custom C++ encapsulation interfaces for those models.
- Mix C++ encapsulation and C code interface styles within a model reference hierarchy. (The demo model `rtwdemo_cppencap` includes a referenced model that uses the C code interface style, compiled with a C++ compiler, while the other models in the same hierarchy use C++ encapsulation interfaces.)

The demo model `rtwdemo_cppencap` has been enhanced to include a model reference hierarchy that illustrates the use of C++ encapsulation interfaces at multiple levels of the hierarchy.

In addition to the limitations that apply to any model build involving C++ encapsulation ("C++ Encapsulation Interface Control Limitations"), the following limitations apply to building a referenced model that is configured to generate a C++ encapsulation interface:

- The `void-void step method` style of C++ encapsulation interface is not supported for referenced models. You must use the `I/O arguments step method` style.
- In cases for which a referenced model cannot have a combined output/update function, you cannot use a C++ encapsulation interface. Those cases include:
  - The model has multiple sample times.
  - The model has a continuous sample time.
  - The model is saving states.

For more information about using C++ encapsulation, see "Controlling Generation of Encapsulated C++ Model Interfaces" in the Real-Time Workshop Embedded Coder documentation.

Type Cast Applied to Mismatched const Type Qualifiers When Using Reference Models with Function Prototype Control or C++ Encapsulation Interfaces

When you control C function prototypes or C++ encapsulation interfaces in models that use model referencing, if the `const` type qualifier for the root input argument of the referenced model's step function interface is set to `none`, and the `const` qualifier for the source signal in the referenced model's parent is set to a value other than `none`, Real-Time Workshop software honors the referenced model's interface specification by generating a type cast that discards the `const` type qualifier from the source signal. To override this behavior, add an appropriate `const` type qualifier to the referenced model.

Target Function Libraries Allow Remapping of Operator Outputs to Implementation Function Inputs

Target function libraries (TFLs) now allow remapping of operator outputs to implementation function inputs. This capability can help satisfy an established coding pattern or convention in an application environment, perhaps a code environment to which generated code is being relocated.

For example, for a sum operation, the build process might generate code similar to the following:

```
rtY.Out1 = u8_add_u8_u8(rtU.In1, rtU.In2);
```

R2009a allows you to remap the output to any position in the implementation function argument list. For example, remapping the output to the first input generates code similar to the following:

```
uint8_T rtb_Add8;

u8_add_u8_u8(&rtb_Add8, rtU.In1, rtU.In2);
rtY.Out1 = rtb_Add8;
```

For more information, see "Remapping Operator Outputs to Implementation Function Input Positions" in the Real-Time Workshop Embedded Coder documentation.

Target Function Library Parameter AcceptExprInput Controls Code Representation of Expression Inputs in Replacement Results

The target function library (TFL) functions `setTflCFunctionEntryParameters` and `setTflCOperationEntryParameters` have added an `AcceptExprInput` parameter for controlling the code representation of expression inputs in function or operator replacements. The `AcceptExprInput` parameter allows you to control whether an expression input to your TFL implementation function is integrated directly into the generated function call or assigned to a temporary variable. If the value of `AcceptExprInput` is `true`, expression inputs are integrated into the generated code in a form similar to the following:

```
rtY.Out1 = u8_add_u8_u8(u8_add_u8_u8(rtU.In1, rtU.In2), rtU.In3);
```

Beginning in R2009a, you can override that behavior by setting `AcceptExprInput` to `false`, which generates a temporary variable for the expression input, as follows:

```
uint8_T tempVar;

tempVar = u8_add_u8_u8(rtU.In1, rtU.In2);
rtY.Out1 = u8_add_u8_u8(tempVar, rtU.In3);
```

AUTOSAR Enhancements

Support is provided for AUTOSAR schema version 3.0, which is now the default.

For information, see "Generating Code for AUTOSAR Software Components" in the Real-Time Workshop Embedded Coder User's Guide documentation.

Enhanced emlc Support for Embedded Real-Time (ERT) Targets

`emlc` now supports:

- Integer-only code generation
- Option to turn off nonfinite support

  You can now turn off nonfinite support if you do not need it. Turning off nonfinite support results in leaner generated code.
- Custom naming of identifiers

  You can now specify the identifier format for:

  - Global variables
  - Global types
  - Local functions
  - Local temporary variables
  - Constant macros

For more information, see "Real-Time Workshop Dialog Box for Embedded MATLAB Coder" in the Real-Time Workshop reference.

Optimized ERT-Based Targets Deprecated

Starting in R2009a, two optimized ERT-based targets are no longer available:

- `ert.tlc  Real-Time Workshop Embedded Coder (auto configures for optimized fixed-point code)`
- `ert.tlc  Real-Time Workshop Embedded Coder (auto configures for optimized floating-point code)`

Compatibility Considerations

If you selected either of these targets for your model in previous releases, the target remains available for that model. For new models, use the code generation objectives parameters to configure your model for fixed-point or floating-point code. For more information, see "Ability to Specify Code Generation Objectives" on page 34.

Right-Click Builds No Longer Generate Unused Functions in Production Code

Prior to R2009a, if all of the following conditions existed, Real-Time Workshop Embedded Coder generated unused functions in production code:

- The subsystem includes a block that has an enable or disable method, or the subsystem itself has an enable or disable method. An example is an Enabled Subsystem block with the Enable block **States when enabling** parameter set to `reset`.
- You right-clicked the subsystem block and did one of the following:
  - Selected the **Tools > Real-Time Workshop > Build Subsystem** submenu option, and the root model is configured for an ERT target.
  - Selected the **Tools > Real-Time Workshop > Generate S-function** submenu option and the **Use Embedded Coder** option on the Generate S-function for Subsystem dialog box.

In a production code environment, the enable and disable methods in your generated *`model`*`.c` file are dead code. The methods are called by the ERT S-function wrapper only if the generated S-function resides in a conditionally executed subsystem that has an enable or disable method.

Starting with R2009a, the code generator no longer generates the enable and disable methods in *model*.c, eliminating the dead code.

Compatibility Considerations

If you place an S-function generated under the preceding conditions in a conditionally executed subsystem that requires the enable or disable method, the S-function block reports a runtime error message similar to the following:

```
The model is attempting to invoke the enable method of the
S-function block block-name for model-name, yet the code
generated for the model does not include an enable method.
```

Pass Reusable Subsystem Outputs as Individual Arguments in Generated Code Using New Configuration Parameter

R2009a introduces a new configuration parameter to pass reusable subsystem outputs as individual arguments to reduce global RAM usage and data copy operations. See "Pass reusable subsystem outputs as" in the Simulink documentation for more information.

Simplify Array Indexing in Generated Code Using New Configuration Parameter

R2009a introduces a new configuration parameter to remove costly multiply operations when accessing arrays in a loop in the generated code. See "Simplify array indexing" in the Simulink documentation for more information.

Generated Code Enhancements

In R2009a, code generation is enhanced to:

- Suppress overflow protection code if the build process determines that the range for block operations is sufficiently restricted. This enhancement improves memory usage and execution speed.
- Identify inputs and outputs of function-call subsystems as well as Simulink functions in Stateflow charts that do not need to use global memory. This capability reduces global memory usage and increases execution speed.

Merge blocks with initialization value have been updated in R2009a to use this enhancement.

- Reduce global data copies in the generated code when the output port of a reusable subsystem connects to a Vector Concatenate block.
- Eliminate unnecessary data copies in the generated code for a SinCos block with outputs to global storage. This capability improves execution speed and reduces RAM usage.
- Identify similar loops and combine them into a single loop iteration, thereby improving the execution speed of the generated code and reducing RAM usage.
- Eliminate unnecessary data copies at the output ports for the Selector and Assignment blocks. This capability generates less code and reduces RAM usage.

  The code efficiency improvement is not applied if:

  - The Selector block connects to more than one block.
  - The input signal of the Assignment block connects to at least one other block.

Code Generation Report Improvement

In R2009a, you can more easily read and use the code generation report:

- In the Generated Source files list in the left pane, the source file name is highlighted to indicate the file displayed in the right pane.
- The destination of the hyperlink in the right pane when navigating hyperlinks to functions, variables, and data types in the generated source files is highlighted.

For more information, see “Generating Reports for Code Reviews and Traceability Analysis”.

Code Traceability Improvement

In previous releases, if the model was built in a different directory from where the model was stored, the hyperlinks in the code generation report did not navigate to the model after the model was closed. In R2009a, the

code generation report records the location of the model at the time of code generation. If a model is closed, clicking a hyperlink opens the model and highlights a block. For more information, see "Tracing Code to Model Objects Using Hyperlinks".

New and Enhanced Demos

Beginning with R2009a, the Real-Time Workshop Demos in the Help contents uses a new format to display Real-Time Workshop and Real-Time Workshop Embedded Coder demos. The differences include:

- Indentation now is used to designate a functionally-related group of demos. Indentation no longer identifies demos for which the Real-Time Workshop Embedded Coder software is required.
- If a demo requires Real-Time Workshop Embedded Coder software, the demo naming or text identifies the requirement.

The following demo has been added in R2009a:

Demo...	Shows How You Can...
`rtwdemo_usingrtwec`	Generate C code for embedded systems requiring production code efficiency and quality. You select a Real-Time Workshop Embedded Coder target for a model, quickly configure options based on your code generation objectives, generate code, and view the resulting files.

The following demos have been enhanced in R2009a:

Demo...	Now...
`rtwdemo_cppencap`	Extends C++ encapsulation to a model reference hierarchy.
`rtwdemo_cscpredef`	Illustrates the use of embedded signal objects.
`rtwdemo_pil`	Shows how to use tunable parameters during a processor-in-the-loop simulation.
`rtwdemo_rtiostream`	Includes configuring for serial communication.

Version 5.2 (R2008b) Real-Time Workshop Embedded Coder Software

This table summarizes what is new in Version 5.2 (R2008b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary	Bug Reports Includes fixes	No

New features and changes introduced in this version are

New Option to Generate Encapsulated C++ Class Interfaces for Model Data and Functions

The Real-Time Workshop language option `C++`, introduced in R14SP2, generates a C++ compatible interface to code generated from Simulink models. However, code generated using the `C++` option has the limitation that model data and functions are not encapsulated into C++ class attributes and methods.

With the addition of the language option `C++ (Encapsulated)` in R2008b, the Real-Time Workshop Embedded Coder software can generate a C++ class interface to model code, in which all required model data is encapsulated into C++ class attributes and all model entry point functions are encapsulated into C++ class methods. The benefits of encapsulation include

- Greater control over access to model data
- Ability to multiply instantiate model classes
- Easier integration of Real-Time Workshop Embedded Coder generated model code into C++ programming environments

C++ encapsulation also can be applied to right-click builds of nonvirtual subsystems.

For more information, see "Controlling Generation of Encapsulated C++ Model Interfaces" in the Real-Time Workshop Embedded Coder documentation.

For a demonstration of the C++ encapsulation capability, see the demo model `rtwdemo_cppencap`.

New CPPClassGenCompliant Target Configuration Parameter

In conjunction with the C++ encapsulation feature described in the previous section, this release introduces the `CPPClassGenCompliant` target configuration parameter. This parameter is set in the `SelectCallback` function for an ERT-based target to indicate whether the target supports the ability to generate and configure C++ encapsulation interfaces to model code. The default is `off` for custom and non-ERT targets and `on` for ERT (`ert.tlc`) targets.

To make an ERT-based target compliant, use the `SelectCallback` function to set `CPPClassGenCompliant` to `on`. This enables the feature infrastructure and user interface.

When the language option `C++ (Encapsulated)` is selected, code generation always generates an example main program, `ert_main.cpp`, demonstrating how the generated code can be deployed. The generated example main file has been updated to declare model data and call the C++ encapsulation interface configured model step method appropriately.

New Processor-in-the-Loop (PIL) Model Block Simulation Mode and API

Real-Time Workshop Embedded Coder now provides Processor-in-the-Loop (PIL) Model block simulation mode and API for verification of embedded code on your target processor.

- Model block PIL mode

  Allows easy switching between Normal, Accelerator and PIL simulation modes.

  You can test object code with no need to modify the original model.

  You can reuse test suites, resulting in faster iteration between model development and generated code verification.

- PIL Connectivity API

  Allows you to access the power of PIL verification for your target processor.

  You can integrate third party or custom tools for:

- Building the PIL application
- Downloading and running the application
- Communicating with the application

For more information see "Verifying Compiled Object Code with Processor-in-the-Loop Simulation" in the Real-Time Workshop Embedded Coder documentation.

See the following new demos for step-by-step examples:

- `rtwdemo_pil`

  This model shows you how to compare results from Normal mode simulation and PIL mode execution for the same referenced model. You simply run the simulation to compare the simulation behavior with the behavior of the corresponding generated code. This uses the default host-based PIL configuration, so the generated code is compiled for and executed on your host workstation.

- `rtwdemo_custom_pil`

  This demo shows you how to create a custom PIL configuration using the target connectivity APIs. You can examine the code that configures the PIL build process, a tool to use for downloading and execution, and a communication channel between host and target. Follow the steps to activate a full host-based PIL configuration.

- `rtwdemo_rtiostream`

  This demo shows you how to implement a communication channel to enable exchange of data between different processes. This is required for PIL, which requires exchange of data between the Simulink software (running on your host machine) and deployed code (executing on target hardware). A default TCP/IP implementation is provided.

AUTOSAR Enhancements

AUTOSAR support has been enhanced to include the following functionality:

- Multiple Runnable Entities

You can model multiple AUTOSAR runnable entities as function-call subsystems.

- Calibration Parameters

  You can import your calibration parameters into the Workspace and assign them to block parameters in your model.

- Access to Basic Software

  You can designate inports and outports as access points to AUTOSAR services and device drivers (basic software).

- Error Status

  You can designate inports to receive error status.

- Enumerations

  You can map MATLAB enumerated types to and from AUTOSAR enumerated types.

- Absolute Time

  You can now use blocks that depend on time, such as the Discrete-Time Integrator block, which you may need to use in a multiple runnable software component.

For information, see "Generating Code for AUTOSAR Software Components" in the Real-Time Workshop Embedded Coder User's Guide documentation.

New functions also provide command-line access to all new AUTOSAR options. For more information, see "Configuring AUTOSAR Options Programmatically" in the Real-Time Workshop Embedded Coder User's Guide documentation.

Also, a new AUTOSAR demo is provided, `rtwdemo_autosar_multirunnables_script`, and the existing two demos are updated.

Flexible Configuration Options for Referenced Models in TLC-Based Custom Targets

In R2008b, you can specify that a configuration option for a TLC-based custom target need not have the same value in a referenced model that it has in

the parent model. By default, the values must be the same in both models. See "Controlling Configuration Option Value Agreement" in the Real-Time Workshop documentation for information about overriding this default.

Generated Code Enhancements

In R2008b, code generation is enhanced to:

- Eliminate data copies for transposing matrices in code. Expression folding reduces the computation into the Transpose block, which improves stack consumption and memory access, and improving execution speed.
- Compute constants for evenly spaced constant data, (for example, 1, 2, 3, ...), rather than defining them in `model_data.c`. This capability reduces ROM consumption and memory access, avoiding compiler stress due to large data sets.
- Optimize inlining of code generated from Simulink blocks, Stateflow charts, and Embedded MATLAB code.
- Remove unnecessary saturation logic code, reducing RAM and ROM, improving code efficiency.
- Provide bidirectional traceability for control ports, such as Trigger and Enable blocks.
- Provide local and reusable input buffers for Model blocks. This capability reduces global data usage and data copying when interfacing with code from a referenced model, which can increase the efficiency of generated code.

Target Function Library Enhancements

Functions pow, power, and memcpy Now Supported for Function Replacement

This release adds `pow`, `power`, and `memcpy` to the list of functions that you can replace with custom or target-specific implementations using target function libraries (TFLs). This allows you to replace default implementations of `pow`, `power`, or `memcpy` that may be insufficient for your application or your target environment.

For a demonstration of `pow`, `power`, and `memcpy` function replacements, see the demo model `rtwdemo_tflmath`. For detailed information about TFLs, see "Replacing Math Functions and Operators Using Target Function Libraries" in the Real-Time Workshop Embedded Coder documentation.

Improved Fixed-Point Multiplication and Division Operator Replacement

In previous releases, TFL support for fixed-point multiplication and division operator replacement did not provide a generalized net slope matching capability. This limited the ability to generate the same TFL replacement function for a range of different input slopes in the model. This release adds a new Net Slope TFL entry, which allows you to match fixed-point slope information between a multiplication or division operator and the TFL entry in a more general and intuitive way.

Enhanced emlc Support for Target Function Libraries

Support for Embedded Real-Time (ERT) Target Function Libraries

You can now configure `emlc` to use ERT target function libraries (TFLs) when generating C code. You enable this feature by defining a configuration object for C code generation using a new `ert` parameter at the MATLAB command prompt, as in this example:

```
rtwcfg = emlcoder.RTWConfig('ert')
```

When you open the properties dialog box for the configuration object, the ERT TFLs will appear in the drop-down list for the **Target function library** field.

This feature requires a Real-Time Workshop Embedded Coder license.

New API for Customizing Target Function Libraries

This release introduces an API for registering custom embedded real-time (ERT) target function libraries (TFLs) for use with `emlc`. You register one or more custom TFLs in a file called `rtwTargetInfo.m` that `emlc` reads on the MATLAB path. This feature requires a Real-Time Workshop Embedded Coder license.

New Code Interface Report Details Generated Code Architecture

A new **Code Interface Report** section has been added to the standard Real-Time Workshop Embedded Coder HTML code generation report. The **Code Interface Report** section provides documentation of the generated code interface, including model entry point functions and interface data, for consumers of the generated code. The information in the report can help facilitate code review and code integration.

The **Code Interface Report** section is automatically included in every Real-Time Workshop Embedded Coder HTML code generation report, so to generate it, you need only select the model option **Create code generation report**. To view the code interface report for a model, build the model, go to the **Contents** pane of the code generation report, and click the **Code Interface Report** link.

For more information, see "Using the Code Interface Report to Analyze the Generated Code Interface" in the Real-Time Workshop Embedded Coder documentation.

Scheduling Code Now Separated from Model Application Code

In previous releases, algorithmic code and rate-monotonic scheduling code were interleaved in multirate model code generated by Real-Time Workshop Embedded Coder software. This could make it difficult to deploy multirate algorithmic code independently of the Real-Time Workshop Embedded Coder scheduling code.

In R2008b, scheduling code has been moved from the generated file *`model`*`.c` to the example main program, creating a clean separation of algorithmic code and scheduling code. This has the following benefits:

- Simplifies deployment and integration of multirate algorithmic code
- Improves efficiency of multirate algorithmic code (less code runs at the base rate)
- Improves memory usage (may reduce or eliminate `rtModel` data)

Code generation assumes the same run-time environment, with a rate-monotonic scheduler in use, as in previous releases.

Note This change is for Real-Time Workshop Embedded Coder code generation only; Real-Time Workshop code generation and Simulink simulation are unaffected. Handling of rate transitions during simulation is unchanged.

Compatibility Considerations

In R2008b, Real-Time Workshop Embedded Coder no longer generates rate-monotonic scheduling code that is interleaved with multirate algorithmic code in *`model`*`.c`. Code for controlling the execution of subrate tasks now resides exclusively in `ert_main`.

Multirate code generated using R2008b will be compatible with previously existing models, except when *both* of the following conditions are true:

- You are using a static main program based on the R2008a version of `ert_main.c` or `.cpp`.
- You are using a system target file based on the R2008a version of `ert.tlc`, and in that file, you have retained the following assignment, which specifies that the `SetEvents` function is inlined in the static main program:

```
%assign InlineSetEventsForThisBaseRateFcn  = TLC_TRUE
```

If both of the listed conditions apply, and if you want to continue to work with your existing static main program and system target file, you must add an `if` statement to your static main program. In the section of your static main that updates event flags and checks for subrate overrun conditions, add the two lines shown in bold below:

```
for (i = FIRST_TID+1; i < NUMST; i++) {
    if (rtmStepTask(RT_MDL,i) && eventFlags[i]++) {
        OverrunFlags[0]--;
        OverrunFlags[i]++;
        /* Sampling too fast */
        rtmSetErrorStatus(RT_MDL, "Overrun");
```

```
            return;
        }
        if (++rtmTaskCounter(RT_MDL,i) == rtmCounterLimit(RT_MDL,i))
            rtmTaskCounter(RT_MDL, i) = 0;
    }
```

Model Initialization Function Prototype Control

Previously, nonreentrant code supported control of the model step function prototype. In R2008b, function prototype control includes control for initialization functions, simplifying code integration and testing, and reducing global RAM usage.

As part of this enhancement, the Model Interface dialog box has the following changes:

Previous Model Interface Dialog Box	New Model Interface Dialog Box
Function Specification parameter options: • `Default model-step function` • `Model Specific C prototype`	**Function Specification** parameter options: • `Default model initialize and step functions` • `Model Specific C prototypes`
Preview subpane	**Step function preview** subpane
Configure function arguments subpane	**Configure model initialize and step functions** subpane
N/A	**Initialize function name** parameter
Function name parameter	**Step function name** parameter
Function argument table	**Step function arguments** table

See “Controlling Generation of Function Prototypes” in the Real-Time Workshop Embedded Coder documentation for more information.

Compatibility Considerations

Previously, if your model included a Stateflow chart that has the chart property **Execute (enter) Chart At Initialization** selected and the chart was connected to model inports or outports, during the build process a Stateflow diagnostic referenced this construct, but code was always generated.

In R2008b, if this construct is in your model and you are controlling the model function prototypes, the build process results in an error and code is not generated. To avoid this error, you can:

1. Place a Signal Conversion block between the root inport or outport and the Stateflow chart.
2. Select the **Override optimizations and always copy signal** parameter of the Signal Conversion block.

Support for Model-Level Control of Test Points

Previously, disabling test points set during debugging required you to disable each test point individually. Not disabling test points can lead to suboptimal generated code. In R2008b, the Real-Time Workshop software provides a single option to ignore all test points during code generation, facilitating transition from prototyping to deployment and avoiding accidental degradation of generated code due to workflow artifacts. See “Ignore test point signals” in the Real-Time Workshop documentation.

Custom Storage Class Enumerated Type Name Misspelling Fixed

The enumerated type name `BuiltinCSCAttributes_IncludeDelimeter` has been renamed `BuiltinCSCAttributes_IncludeDelimiter`.

Custom Storage Class File Type Changed in R2008a

Up until R2007b, The MathWorks released built-in custom storage class definitions only in `.m` files. In R2008a and R2008b (the current release) The MathWorks released built-in custom storage class definitions in both `.m` files and `.p` files, creating a possible incompatibility. No R2008a release note appeared that described this change, but a note now appears in its correct location. See the R2008a release note “Custom Storage Class File

Type Changed” on page 62 for information about the change and the possible “Compatibility Considerations” on page 63 that it introduced.

Minimize Local Variables in Code Generation Using New Configuration Parameter

R2008b introduces a new configuration parameter to eliminate copies between local and global variables, reducing stack size and data copy operations. See “Minimize data copies between local and global variables” in the Simulink documentation for more information.

Simulink Configuration Parameters

In R2008b, the following Simulink Configuration Parameters dialog box button label is updated:

Location	Previous Parameter Label	New Parameter Label
Real-Time Workshop > Interface	**Configure Step Function**	**Configure Model Functions**

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_autosar_multirunnables_script`	Configure and generate AUTOSAR-compliant code and export AUTOSAR software component description XML files for a Simulink model with multiple runnable entities.
`rtwdemo_cppencap`	Generate a customizable C++ encapsulation interface to code generated by the Real-Time Workshop Embedded Coder software.
`rtwdemo_custom_pil`	Create a custom processor-in-the-loop (PIL) configuration using target connectivity APIs.

Demo...	Shows How You Can...
`rtwdemo_iec61508_script`	Use Model Advisor checks to help you develop a model and code that comply with the IEC 61508 standard (uses Simulink® Verification and Validation™ software).
`rtwdemo_pil`	Compare results from Normal mode simulation and PIL mode execution for the same referenced model.

The following demo has been enhanced in R2008b:

- `rtwdemo_tfl_script`

Version 5.1.1 (R2008a+) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 5.0.1 (R2008a+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	No

Version 5.1 (R2008a) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 5.1 (R2008a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary	Bug Reports Includes fixes	No

New features and changes introduced in this version are

New AUTOSAR Compliant Code Generation Capability and Demos

Real-Time Workshop Embedded Coder V5.1 (R2008a) provides AUTOSAR-compliant code generation configurable by GUI, command line functions, or AUTOSAR-compliant XML files. For information, see "Generating Code for AUTOSAR Software Components" in the Real-Time Workshop Embedded Coder User's Guide documentation.

Select the new system target file `autosar.tlc` to access the **AUTOSAR Code Generation Options** pane in the Configuration Parameters dialog box. You can then launch the Model Interface dialog to access all options for configuring AUTOSAR-compliant code generation and import/export to and from XML files.

New functions provide command-line access to all AUTOSAR options. For more information, see "Configuring AUTOSAR Options Programmatically" in the Real-Time Workshop Embedded Coder User's Guide documentation.

Also, two AUTOSAR demos are provided, `rtwdemo_autosar_roundtrip_script` and `rtwdemo_autosar_legacy_script`.

Bidirectional Traceability for Stateflow Charts and Embedded MATLAB Functions

In previous releases, the Real-Time Workshop Embedded Coder software provided bidirectional traceability for Simulink blocks only. In R2008a, bidirectional traceability is added between generated code and Stateflow chart objects and Embedded MATLAB scripts. For embedded real-time (ERT) based targets, you can choose to include traceability comments in the generated code. Using the enhanced traceability report, you can click hyperlinks to go from a line of code to its corresponding item in the model. You can also right-click an item in your model to find its corresponding line of code.

The following parameters are added or updated for bidirectional traceability:

Previous Parameter Location and Name	New Parameter Location and Name
Real-Time Workshop > General pane: **Generate HTML report**	**Real-Time Workshop > Report** pane: **Create code generation report**
Real-Time Workshop > General pane: **Launch report automatically**	**Real-Time Workshop > Report** pane: **Launch report automatically**
Real-Time Workshop > General pane: **Code-to-block highlighting**	**Real-Time Workshop > Report** pane: **Code-to-model**
Real-Time Workshop > General pane: **Block-to-code highlighting**	**Real-Time Workshop > Report** pane: **Model-to-code**
N/A	**Real-Time Workshop > Report** pane: **Eliminated / virtual blocks**
N/A	**Real-Time Workshop > Report** pane: **Traceable Simulink blocks**
N/A	**Real-Time Workshop > Report** pane: **Traceable Stateflow objects**
N/A	**Real-Time Workshop > Report** pane: **Traceable Embedded MATLAB functions**
Real-Time Workshop > Comments pane: **Simulink block comments**	**Real-Time Workshop > Comments** pane: **Simulink block / Stateflow comments**

Also, the right-click **Real-Time Workshop > Highlight Code** menu option is now **Real-Time Workshop > Navigate to Code**.

For more information, see "Traceability of Stateflow Objects in Real-Time Workshop Generated Code" in the Stateflow and Stateflow® Coder™ documentation, "Using Traceability in Embedded MATLAB Function Blocks" in the Simulink documentation, and "Generating Reports for Code Reviews

and Traceability Analysis" in the Real-Time Workshop Embedded Coder documentation.

Generated Code Enhancements

In R2008a, code generation is enhanced to

- Enable cross product optimizations between Simulink blocks and Stateflow charts.
- Reduce the size of code and improve code execution speed for the Bus Assignment, Bus Creator and Bus Selector blocks.

Function Prototype Control Enhancements

In R2008a, function prototype control:

- Adds a preview function prototype command, `getPreview`, when configuring the prototype programmatically.
- Adds the capability to work with model references.
- Ignores the **Pass scalar root inputs by value** model reference configuration parameter when a *`model`*`_step` function prototype is specified.

For more information, see "Controlling Generation of Function Prototypes" in the Real-Time Workshop Embedded Coder documentation.

Compatibility Considerations

Previously, the code generator ignored function prototype control specifications for reference models. In R2008a, when you configure a referenced model for function prototype control and multi-instance implementations, the code generator reports an error. You can fix the error by doing one of the following:

- Configure the referenced model for single-instance implementation.
- Disable function prototype control for the model.

Custom Storage Class File Type Changed

In previous releases, a built-in class's custom storage class definitions were released in a `.m` file. In R2008a, they are released in both a `.m` file and a functionally equivalent `.p` file. The `.p` files take precedence over `.m` files, so when both files exist MATLAB ignores the `.m` file and loads the `.p file`.

The `.m` file exists for human readability and to support customizing built-in custom storage classes (CSCs).

You can examine a built-in class's custom storage class definitions in the Custom Storage Class Designer. In previous releases, the designer loaded the `.m` file that defined the CSCs. You can edit `.m` files, so you could edit built-in CSC definitions and save the changes, which would take effect immediately. In R2008a, the CSC designer loads the `.p` file rather than the `.m` file. However, `.p` files cannot be edited, so all CSC Designer editing capabilities are disabled when the CSC Designer obtained the CSCs that it displays from a `.p` file.

You can delete a `.p` file that contains CSC definitions, in which case MATLAB and the CSC Designer load the corresponding `.m` file, which defines exactly the same CSCs as the `.p` file. However, `.m` files can be edited, so you can then edit built-in CSCs as in previous releases. However, The MathWorks discourages this practice. If you delete both the `.p` file and the `.m` file for a package that defines CSCs, an error occurs when you try to use the package.

Compatibility Considerations

No compatibility problems arise unless you changed built-in custom storage class definitions in earlier releases and want to continue to do so in R2008a. As of R2008a, The MathWorks discourages customizing built-in custom storage class files. The preferred technique is to create user-defined packages and custom storage classes, as described in "Subclassing Simulink Data Classes" and "Creating and Using Custom Storage Classes". The CSC designer stores user-defined CSC definitions in `.m` files, so the interface to that capability is unchanged in R2008a.

If you want to continue the practice of customizing the CSC definitions for a built-in package, you must first delete the P-file that defines the CSCs for the package. That file is always named `csc_registration.p`, and exists on the MATLAB path in a directory named *`@package`*, where *`package`* is the package name, e.g. `@simulink`. With the P-file deleted, the CSC designer loads the `.m` file, which you can then edit and save as in earlier releases. MATLAB then loads the changed CSC definitions for that package from the `.m` file.

Improved MISRA C Compliance for Matrix Math Utilities and Lookup Block Utilities

This release improves the MISRA C compliance of matrix math utilities and lookup block utilities that are used in generated code.

math.h Header File Inclusion Now Controllable Through Target Function Library Customization

In previous releases, code generated by the Real-Time Workshop Embedded Coder software automatically included the `math.h` header file, defining C standard math functions, regardless of the math requirements of the target environment. In this release, selecting or customizing a target function library (TFL) for your model controls which header files are included, and generated code does not automatically include `math.h` unless it is needed.

For more information about selecting TFLs, see "Selecting and Viewing Target Function Libraries" in the Real-Time Workshop documentation. For more information about customizing TFLs, see "Replacing Math Functions and Operators Using Target Function Libraries" in the Real-Time Workshop Embedded Coder documentation.

"What's This?" Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

R2008a introduces "What's This?" context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

1. Place your cursor over the label of a parameter.
2. Right-click. A **What's This?** context menu appears.

   For example, the following figure shows the **What's This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.

3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_autosar_legacy_script`	Prepare, implement, and verify an existing model for AUTOSAR by using the AUTOSAR target.
`rtwdemo_autosar_roundtrip_script`	Import, modify, and export AUTOSAR software components.
`rtwdemo_polyspace`	Use PolySpace® products to prove both the absence and presence of run-time errors for code generated by Real-Time Workshop Embedded Coder software. It also shows the results of MISRA C compliance for the generated code.

The following demo has been enhanced to illustrate code traceability improvements in R2008a:

- `rtwdemo_hyperlinks`

Version 5.0.1 (R2007b+) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 5.0.1 (R2007b+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	No

Version 5.0 (R2007b) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 5.0 (R2007b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes	No

New features and changes introduced in this version are

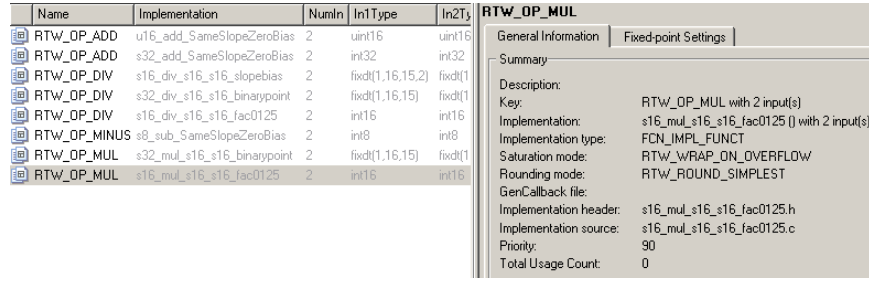
New Target Function Library (TFL) API for Mapping Math Functions and Operators to Target-Specific Code

In previous releases, the **Target floating-point math environment** parameter on the **Interface** pane of the Configuration Parameters dialog box allowed you to select a math library (ANSI®, ISO®, or GNU®) to which function calls would be generated for appropriate/supported functions within the generated code for your model. However, no general mechanism was provided for creating and registering generic target-specific function libraries.

This release provides the target function library (TFL) API, which allows you to create and register function replacement tables. When selected for a model, these TFL tables provide the basis for replacing default math functions and operators in your model code with target-specific code. The ability to control function and operator replacements in this manner potentially allows you to optimize target performance (speed and memory) and better integrate model code with external and legacy code.

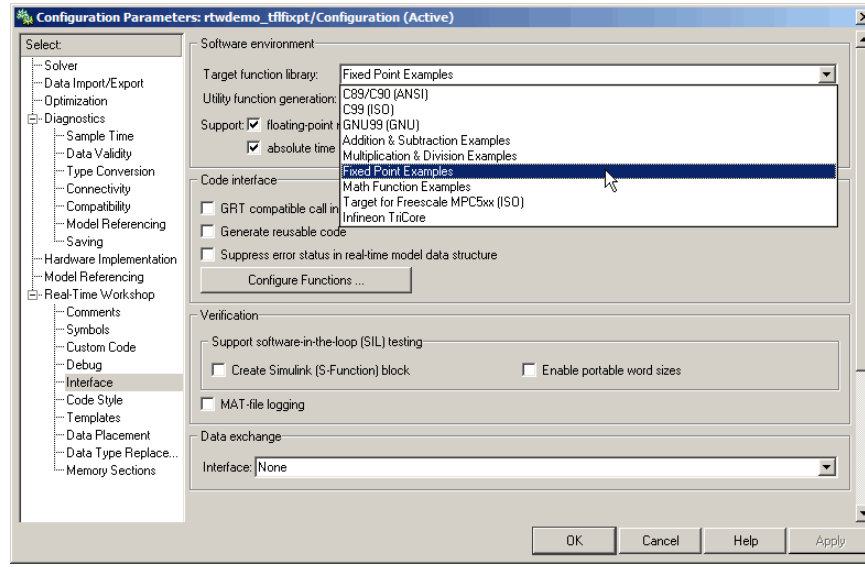
The general steps for creating and using a target function library are as follows:

1. Create one or more TFL tables containing replacement entries for math operators (+, –, *, /) and functions using a MATLAB-based API. (The demo `rtwdemo_tfl_script` provides example tables that can be used as a starting point for customization.)



2. Register a target function library, consisting of one or more replacement tables, using a Simulink `sl_customization` API.

3 Open your model and select the desired target function library from the **Target function library** drop-down list, located on the **Interface** pane in the Configuration Parameters dialog box.



4 Build your model.

For more information, see “Replacing Math Functions and Operators Using Target Function Libraries” in the Real-Time Workshop Embedded Coder documentation.

Additionally, see the new demo `rtwdemo_tfl_script`, which illustrates how to use TFLs to replace operators and functions in generated code. With each example model included in this demo, a separate TFL is provided to illustrate the creation of operator and function replacements and how to register the replacements with Simulink.

Bidirectional Traceability Now Supported Through Automated Block-to-Code and Code-to-Block Highlighting

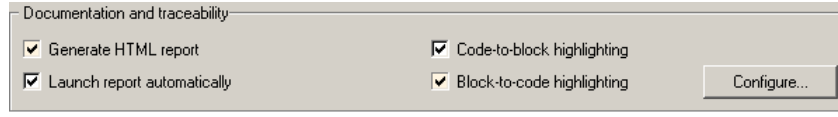
In previous releases, Real-Time Workshop Embedded Coder software provided traceability from generated code back to model source blocks through the **Include hyperlinks to model** option on the **Real-Time Workshop** pane of the Configuration Parameters dialog box.

This release provides bidirectional traceability between model source blocks and generated code by

- Renaming the existing option from **Include hyperlinks to model** to **Code-to-block highlighting**
- Adding the **Block-to-code highlighting** option, which allows you to select a block and highlight its generated code.

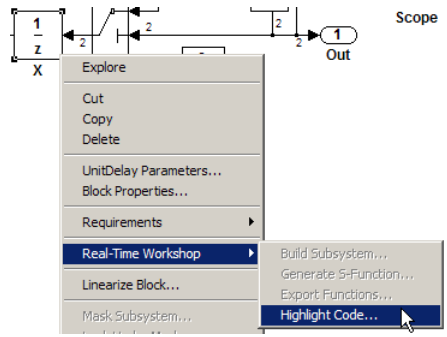
To use **Block-to-code highlighting**,

1. Open an ERT-based model and go to the **Real-Time Workshop** pane of the Configuration Parameters dialog box. Select the options **Generate HTML report**, **Launch report automatically**, and **Block-to-code highlighting**. (Selecting **Block-to-code highlighting** also enables the **Configure** button, which you can click to select a build directory to be traced.)

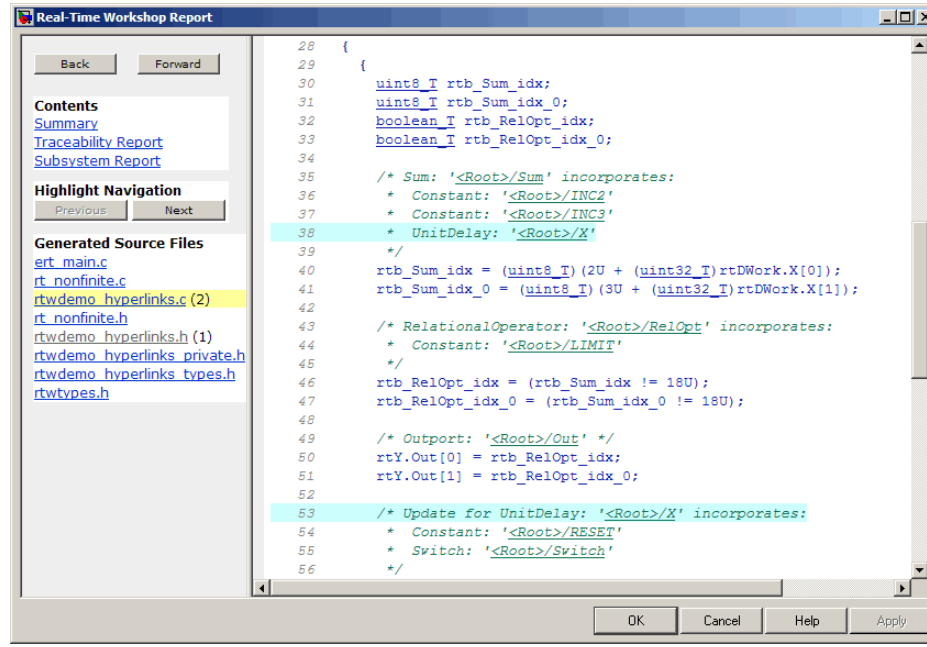


2. Build your model. This will launch an HTML code generation report.

3. In the model window, right-click any block. In the right-click menu, select **Real-Time Workshop > Highlight Code**.



4 This selection highlights the generated code for the block in the HTML code generation report. The total number of highlighted lines is displayed next to each source file name in the left panel of the HTML report. **Previous** and **Next** buttons help you navigate through the highlighted lines.

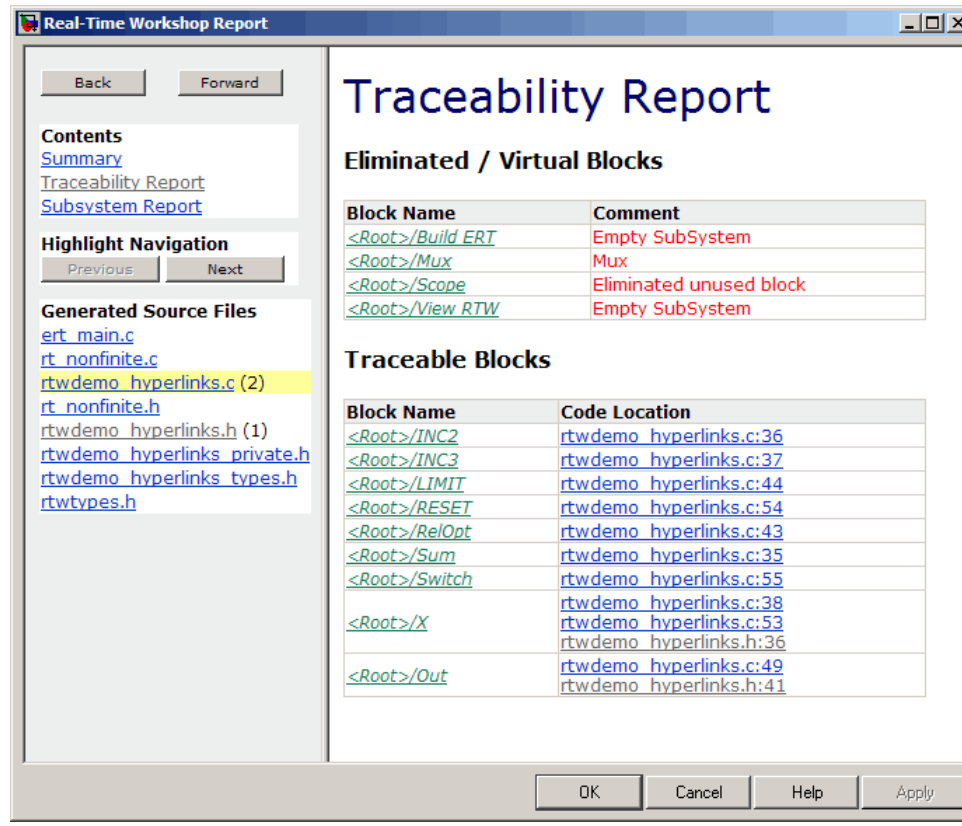


For more information, see "Generating Reports for Code Reviews and Traceability Analysis" in the Real-Time Workshop Embedded Coder documentation.

Additionally, see the enhanced demo `rtwdemo_hyperlinks`, which walks you through using **Code-to-block highlighting**, **Block-to-code highlighting**, and the traceability report discussed in the next section.

HTML Code Generation Report Adds Traceability Report

In previous releases, HTML code generation reports did not provide information to help explain why some model blocks do not generate corresponding code. In this release, when you select the **Block-to-code highlighting** parameter discussed in the previous section, the generated HTML report contains a traceability report. The traceability report contains sections that allow you to account for **Eliminated / Virtual Blocks** versus **Traceable Blocks**, providing a complete mapping between blocks and code.



For more information, see "Generating Reports for Code Reviews and Traceability Analysis" in the Real-Time Workshop Embedded Coder documentation.

Additionally, see the enhanced demo `rtwdemo_hyperlinks`, which walks you through using the new traceability report.

Elimination of Wrapper Generated by R2007a model_step Function Prototype Control Feature

In R2007a, function prototype control generated a wrapper function to implement the *`model`*`_step` function. In R2007b, function prototype control modifies the *`model`*`_step` function directly, reducing execution time.

Optimized External I/O Data Structures with Function Prototype Control

In R2007b, function prototype control optimizes external I/O data structures in the following ways:

- The data structure of a model's external input is removed unless the value of the external input is used in a subsystem implemented by a nonreusable function.
- The data structure for the model's external output is removed except when MAT-file logging is enabled, or if the sample time of the outport is constant.

For more information, see "Controlling Generation of Function Prototypes" in the Real-Time Workshop Embedded Coder documentation.

MISRA C Compliance Enhanced for Enabled Subsystem Code

R2007b improves MISRA C compliance of generated code by:

- Implementing enabled subsystem logic more efficiently using if-then-else statements
- Improving compliance of library code used in generated code

User-Defined Data Classes Can Reference Custom Storage Classes from Other Packages

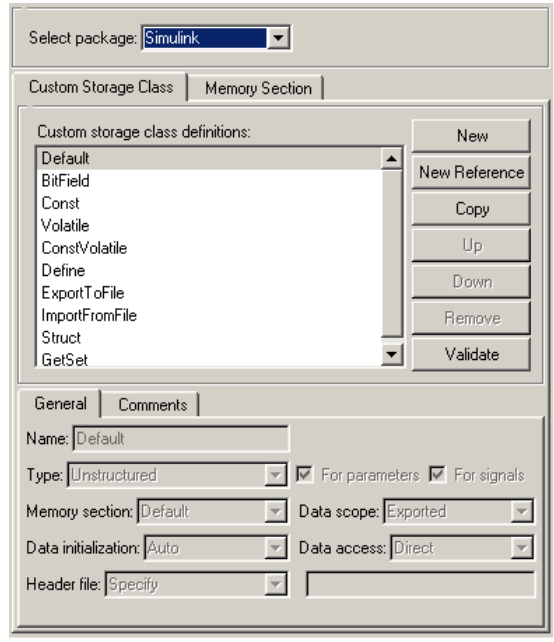
In previous releases, the custom storage classes and memory section definitions contained in a package were unique to that package, and could not be used directly by other packages. The only recourse was to create a duplicate definition in every package that needed it. If a global change was needed in the definition of the class or section, each local copy had to be updated separately.

Any package can access and use custom storage classes and memory sections that are defined in any other package, including both user-defined packages and predefined packages such as `Simulink` and `mpt`. Only one copy of the class or section exists, in the package that first defined it; other packages refer to it by pointing to it in its original location. Thus any changes to the class or

section, including changes to a predefined class/section in later MathWorks product releases, are available in every referencing package.

To configure a package to use a custom storage class or memory section that is defined in another package:

1. Type `cscdesigner` to launch the Custom Storage Class Designer. The relevant part of the designer window looks like this:



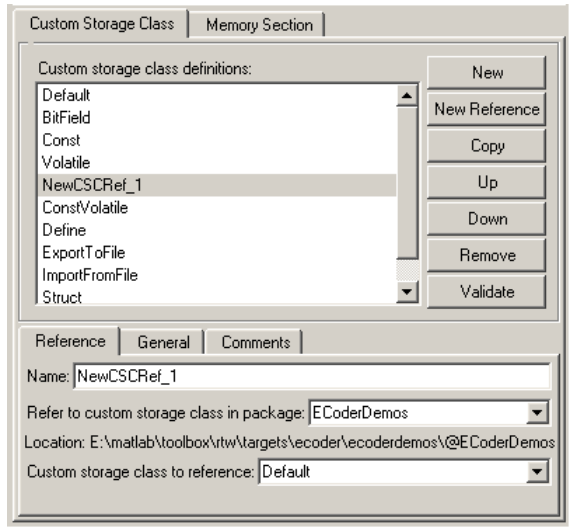
2. Select the **Custom Storage Class** or **Memory Section** tab as appropriate. The following example assumes **Custom Storage Class**. Memory section references work the same way.

3. Use **Select Package** to select the package in which you want to reference a class (or section) defined in some other package.

4. In the **Custom storage class definitions** pane, select the existing definition below which you want to insert the reference.

5 Click **New Reference**.

A new reference with a default name and properties appears below the previously selected definition. The new reference is selected, and a **Reference** tab appears that shows the reference's initial properties. This tab appears whenever a reference is selected, allowing reference properties to be viewed and changed. A typical appearance is:



6 Use the **Name** field to enter a name for the new reference. The name must be unique in the importing package, but can duplicate the name in the source package.

7 Set **Refer to custom storage class in package** to specify the package that contains the custom storage class you want to reference.

8 Set **Custom storage class to reference** to specify the custom storage class to be referenced. Trying to create a circular reference generates an error and leaves the package unchanged.

9 Click **OK** or **Apply**.

If you had worked under the **Memory Section** tab rather than the **Custom Storage Class** tab, the sequence would have been essentially the same, with

appropriate differences in the dialog box labels and the set of items available to be chosen.

You can use Custom Storage Class Designer capabilities to copy, reorder, validate, and otherwise manage classes and sections that have been added to a class by reference. However, you cannot change the underlying definitions. You can change a custom storage class or memory section only in the package where it was originally defined.

For more information, see "Using Custom Storage Class References" and "Using Memory Section References".

Data Type Assistant Support for MPT Objects

Simulink now provides a standardized user interface, the **Data Type Assistant**, for specifying data types associated with Simulink blocks and data objects, as well as Stateflow data. See "Using the Data Type Assistant" for details.

The Data Type Assistant appears on the dialogs of a variety of blocks and data objects, including MPT data objects, which are specific to the Real-Time Workshop Embedded Coder software:

- `mpt.Parameter`
- `mpt.Signal`

Information about MPT objects appears in "Defining Data Representation and Storage for Code Generation".

New Target Configuration Parameter for Enabling Real-Time Workshop Compiler Optimization Level Control

V5.0 (R2007b) Real-Time Workshop Embedded Coder introduces a new target configuration parameter, `CompOptLevelCompliant`. This parameter indicates whether a target supports the new Real-Time Workshop configuration parameter **Compiler Optimization Level**. (**Compiler Optimization Level** controls the compiler optimization level for building generated code; for more information, see "Controlling Compiler Optimization Level and

Specifying Custom Optimization Settings" in the Real-Time Workshop documentation.)

When the `CompOptLevelCompliant` target configuration parameter is set to `on`, the **Compiler Optimization Level** parameter is displayed in the **Real-Time Workshop** pane of the Configuration Parameters dialog box for your model. If the `CompOptLevelCompliant` parameter is not set to `on`, the **Compiler Optimization Level** parameter does not appear.

By default, the `CompOptLevelCompliant` parameter is set to `off` for custom targets and `on` for targets provided by Real-Time Workshop and Real-Time Workshop Embedded Coder.

To make a target compliant, use the `SelectCallback` function to set `CompOptLevelCompliant` to `on`, and then modify the target makefile to honor the setting for **Compiler Optimization Level**, in the manner of the targets provided by Real-Time Workshop and Real-Time Workshop Embedded Coder.

New Interactive Guided Introduction Demo

An interactive demo of Real-Time Workshop Embedded Coder is available. This demo shows you how to apply MathWorks products to the basic steps that are common to most projects that design and implement a control algorithm. To view the demo:

1. In the MATLAB command window, enter the following command:

   ```
   >> rtwdemos
   ```

   The Real-Time Workshop Demos page is displayed.

2. Scroll down to **Step-by-Step Code Generation Process** under **Guided Tutorials**.

Help displays the names of the modules that comprise the **Step-by-Step Code Generation Process** demo. To begin viewing the demo, click the name of the first module, **Introduction**, then proceed through subsequent modules in order, or jump directly to any that are of particular interest.

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_pcgd_intro`	Apply MathWorks products to the basic steps that are common to most projects that design and implement a control algorithm. For more information, see "New Interactive Guided Introduction Demo" on page 78.
`rtwdemo_tfl_script`	Use Target Function Libraries (TFLs) to replace operators and functions in generated code. With each example model included in this demo, a separate Target Function Library is provided to illustrate the creation of operator and function replacements using a MATLAB based API, and how to register the replacements with Simulink.

The following demo has been enhanced to illustrate code traceability improvements in R2007b, including block-to-code highlighting and traceability report enhancements:

- `rtwdemo_hyperlinks`

Version 4.6.1 (R2007a+) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.6.1 (R2007a+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	No

Version 4.6 (R2007a) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.6 (R2007a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports Includes fixes	No

New features and changes introduced in this version are

Controlling Step Function Prototypes for Models

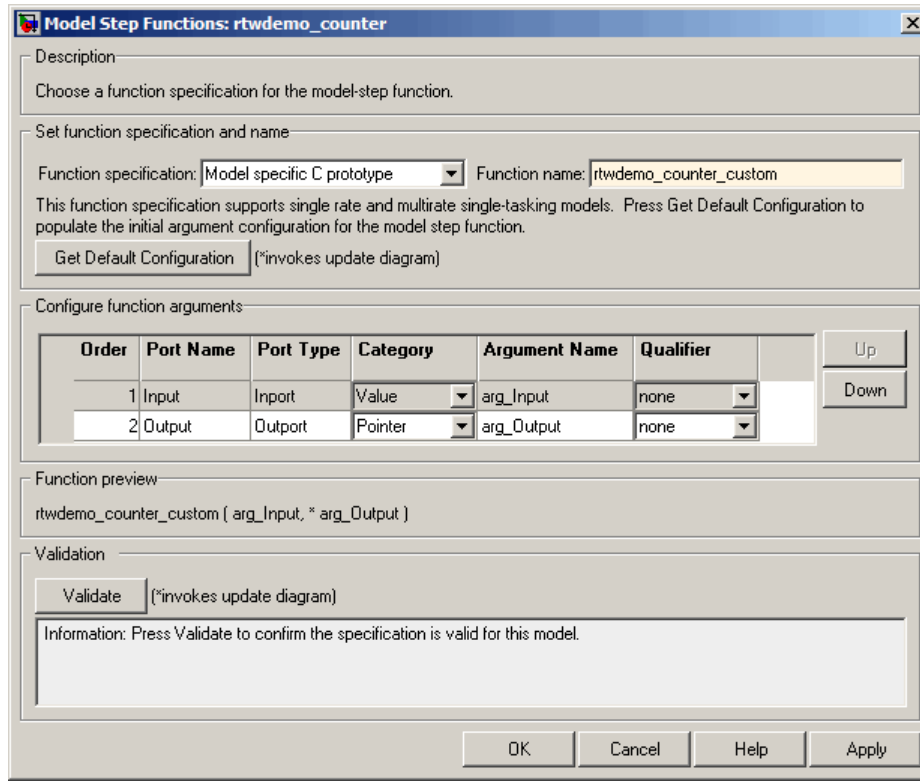
In previous releases, there were only limited ways to control the function prototype of an ERT-based model's generated *model*_step function. The default *model*_step function prototype resembles the following:

```
void model_step(void);
```

If you generate reusable, reentrant code for an ERT-based model, the model's root-level inputs and outputs, block states, parameters, and external outputs are passed in to *model*_step using a function prototype that resembles the following:

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

This release adds more flexible user control over the *`model`*`_step` function prototype that is generated for ERT-based Simulink models. From the **Interface** pane of the Configuration Parameters dialog box, you can click a new **Configure Functions** button that launches a Model Step Functions dialog box. Based on the **Function specification** value you select for your *`model`*`_step` function (supported values include `Default model-step function` and `Model specific C prototype`), you can preview and modify the function prototype. Here is a sample dialog box:



Once you validate and apply your changes, you can generate code based on your function prototype modifications.

Alternatively, you can use function prototype control functions to programmatically control *model*_step function prototypes. For more information, see "Configuring Model Function Prototypes Programmatically" in the Real-Time Workshop Embedded Coder documentation.

You can also control step function prototypes for nonvirtual subsystems, if you generate subsystem code using right-click build. To launch the Model Step Functions for subsystem dialog box, use the function `RTW.configSubsystemBuild`:

```
RTW.configSubsystemBuild('model/subsystem')
RTW.configSubsystemBuild(gcb)
```

Right-click building the subsystem will generate the step function according to the customizations you make.

For more information about controlling *model*_step function prototypes, see the sections "Configuring the Target Hardware Environment" and "Controlling Generation of Function Prototypes" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Model Function Prototype Control Limitations" in the Real-Time Workshop Embedded Coder documentation.

For more detailed information about the default calling interface for the *model*_step function, see the `model_step` reference page.

New ModelStepFunctionPrototypeControlCompliant Target Configuration Parameter

In conjunction with the function prototype control feature described in the previous section, this release introduces the `ModelStepFunctionPrototypeControlCompliant` target configuration parameter. This parameter is set in the `SelectCallback` function for a target to indicate whether the target supports the ability to control the function prototypes of step functions that are generated for a Simulink model. The default is `off` for custom and non-ERT targets and `on` for ERT (`ert.tlc`) targets.

When this parameter is set to `off` and you attempt to use function prototype control to modify a step function signature, Real-Time Workshop Embedded Coder ignores the modified function prototype control configuration.

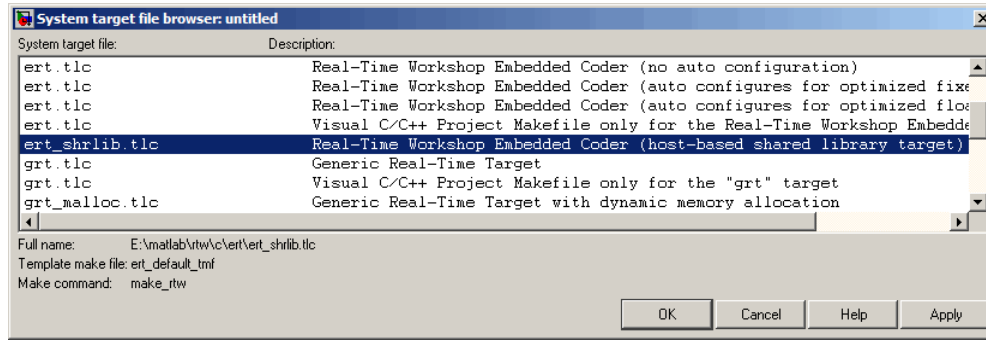
To make a target compliant,

1. Use the `SelectCallback` function to set `ModelStepFunctionPrototypeControlCompliant` to `on`. This enables the feature infrastructure and user interface.
2. If your target uses a custom static main module, and if a nondefault function prototype control configuration is associated with a model, update the main module to call the function prototype controlled model step function. You can do this in either of the following ways:
   a. Manually adapt your main module to declare appropriate model data and call the function prototype controlled model step function.
   b. Generate your main module using **Generate an example main program** on the **Templates** pane of the Configuration Parameters dialog box. This mechanism has been updated to declare model data and call the function prototype controlled model step function appropriately.

New ERT Target for Generating Host-Based Shared Libraries

This release adds a new ERT target, `ert_shrlib.tlc`, for generating a host-based shared library from your Simulink model. Selecting this target allows you to generate a shared library version of your model code that is appropriate for your host platform, either a Windows® dynamic link library (`.dll`) file or a UNIX® shared object (`.so`) file. This feature can be used to package your source code securely for easy distribution and shared use. The generated `.dll` or `.so` file is shareable among different applications and upgradeable without having to recompile the applications that use it.

To configure your model code for shared use by applications, you select the `ert_shrlib.tlc` target on the **Real-Time Workshop** pane of the Configuration Parameters dialog box.



The shared library generated from your model can be dynamically loaded from another application. For example, if you open the demo `rtwdemo_counter`, select the `ert_shrlib.tlc` target, and generate code, application code similar to the following could be used to dynamically load the generated library file:

```
#if (defined(_WIN32)||defined(_WIN64)) /* WINDOWS */
#include <windows.h>
#define GETSYMBOLADDR GetProcAddress
#define LOADLIB LoadLibrary
#define CLOSELIB FreeLibrary

#else /* UNIX */
#include <dlfcn.h>
#define GETSYMBOLADDR dlsym
#define LOADLIB dlopen
#define CLOSELIB dlclose

#endif

int main()
{
    void* handleLib;
...
#if defined(_WIN64)
    handleLib = LOADLIB("./rtwdemo_counter_win64.dll");
#else
#if defined(_WIN32)
    handleLib = LOADLIB("./rtwdemo_counter_win32.dll");
```

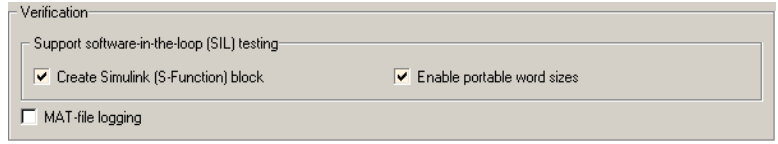
```
#else /* UNIX */
    handleLib = LOADLIB("./rtwdemo_counter.so", RTLD_LAZY);
#endif
#endif
...
    return(CLOSELIB(handleLib));
}
```

For more information about using the `ert_shrlib.tlc` target, see "Creating and Using Host-Based Shared Libraries" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Host-Based Shared Library Limitations" in the Real-Time Workshop Embedded Coder documentation.

Enhanced Software-in-the-loop (SIL) Testing with New Portable Word Sizes Option

This release adds a new model configuration option, **Enable portable word sizes**, that supports code generation for host-target configurations in which the processor word sizes differ between host and target platforms (for example, a 32-bit host and a 16-bit target). Selecting the **Enable portable word sizes** option allows you to generate code with conditional processing macros that allow the same generated source code files to be used both for software-in-the-loop (SIL) testing on the host platform and for production deployment on the target platform.

To use this feature, select both **Create Simulink (S-Function) block** and **Enable portable word sizes** on the **Interface** pane of the Configuration Parameters dialog box. Also, make sure that **Emulation hardware** is set to **None** on the **Hardware Implementation** pane.



When you generate code from your model, data type definitions are conditionalized such that `tmwtypes.h` is included to support SIL testing on the host platform and Real-Time Workshop types are used to support

deployment on the target platform. For example, in the generated code below, the first two lines define types for host-based SIL testing and the **bold** lines define types for target deployment:

```
#ifdef PORTABLE_WORDSIZES    /* PORTABLE_WORDSIZES defined */
# include "tmwtypes.h"
#else                        /* PORTABLE_WORDSIZES not defined */
#define __TMWTYPES__
#include <limits.h>
...
typedef signed char int8_T;
typedef unsigned char uint8_T;
typedef int int16_T;
typedef unsigned int uint16_T;
typedef long int32_T;
typedef unsigned long uint32_T;
typedef float real32_T;
typedef double real64_T;
...
#endif                       /* PORTABLE_WORDSIZES */
```

To build the generated code for SIL testing on the host platform, the definition `PORTABLE_WORDSIZES` should be passed to the compiler, for example by using the compiler option `-DPORTABLE_WORDSIZES`. To build the same code for target deployment, the code should be compiled without the `PORTABLE_WORDSIZES` definition.

For more information about using portable word sizes for host-based SIL testing, see "Configuring the Target Hardware Environment" and "Configuring Hardware Implementation Settings" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Portable Word Sizes Limitation" in the Real-Time Workshop Embedded Coder documentation.

New Code Style Options for Controlling Expression Optimizations in Generated Code

Two new options on the **Code Style** pane of the Configuration Parameters dialog box allow you to control specific optimizations in generated code:

Option	Description
Preserve operand order in expression	By default, Real-Time Workshop might reorder commutable operands to make an expression left-recursive. Selecting this option preserves the expression order you specify in the model.
Preserve condition expression in if statement	By default, Real-Time Workshop negates the condition expression in an `if` statement if the first statement branch is empty. Selecting this option preserves the condition expression you specify in the model.

For more information, see Code Style Pane in the Real-Time Workshop Embedded Coder documentation.

Enhanced MISRA C Compliance

This release provides several enhancements to MISRA C compliance, including

- Numerous improvements to source files in the *`matlabroot`*`/rtw/c/libsrc` directory
- Elimination of `goto` statements in Stateflow generated code (for more information, see the Stateflow and Stateflow Coder Release Notes)
- Simplified generated code for reusable enabled subsystems (for more information, see the Real-Time Workshop Release Notes)

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_fcnprotoctrl`	Control the generated function prototype for the model entry point function `model_step`
`rtwdemo_shrlib`	Use the ERT shared library target to generate a host-based shared library (`.dll` or `.so`) from a model and then load the shared library from another application

The following demo has been enhanced:

- `rtwdemo_sil`

Version 4.5 (R2006b) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.5 (R2006b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports Includes fixes	No

New features and changes introduced in this version are

• "New Reference Documentation" on page 97

Efficiency Enhancements in Generated Code

Real-Time Workshop Embedded Coder V4.5 (R2006b) provides the following efficiency enhancements in code generated from Simulink models:

- Element-by-element optimized code for vector operations
- Improved efficiency and readability of `for` loops generated for wide signals that have multiple sources
- Unnecessary temporary variables no longer generated for muxed signals

Fixed-Point Code Generation Support for Enhanced N-Dimensional Lookup Table Blocks

Real-Time Workshop Embedded Coder V4.5 (R2006b) supports fixed-point code generation for the following new Simulink N-dimensional lookup table blocks:

- Prelookup
- Interpolation Using Prelookup

The new blocks provide fixed-point arithmetic and more efficient code generation than the blocks they replace, PreLookup Index Search and Interpolation (n-D) Using PreLookup.

Ability to Control Use of Parentheses in Generated Code

The new **Code Style** pane in the Configuration Parameters dialog box allows you to control optional parentheses in generated code, including generating code that meets MISRA® requirements. The default behavior is to generate code similar to that of previous releases.

For more information, see Code Style Pane in the Real-Time Workshop Embedded Coder documentation.

Enhanced HTML Code Report Performance and Content

If you select the **Generate HTML report** check box on the **Real-Time Workshop** pane of the Configuration Parameters dialog box, Real-Time Workshop Embedded Coder automatically produces a code generation report in HTML format. In R2006b, the performance associated with producing the code generation report has improved significantly. In addition, the reports no longer display the names of hidden blocks, such as automatically inserted Rate Transition blocks, as hyperlinks.

New General-Purpose OSEK/VDX Real-Time Operating System (RTOS) Example

Real-Time Workshop Embedded Coder V4.5 (R2006b) adds a new demo, `rtwdemo_osek`, that illustrates techniques for interfacing to the OSEK/VDX® real-time operating system (RTOS). The demo model includes:

- Example Simulink block implementations of OSEK functions `SetAlarm` and `ActivateTask`
- Function-call subsystems that are generated as separate OSEK tasks, which can execute based on assigned priority using the OSEK scheduler

Additional files related to the demo are provided in *`matlabroot`*`/toolbox/rtw/rtwdemos/osektgt_demo`. These include:

- Real-Time Workshop Embedded Coder file customization template `osek_file_process.tlc`, which generates a generic OSEK main program and an OSEK Implementation Language (OIL) file
- OSEK library file `oseklib.tlc`, which contains functions called by `osek_file_process.tlc`
- C and C MEX-files for the S-functions `oseksetalarm` and `osektask`

After launching `rtwdemo_osek`, you can save the model file `rtwdemo_osek.mdl` to a work directory. You can use the model file and the related files in *`matlabroot`*`/toolbox/rtw/rtwdemos/osektgt_demo` as a starting point to target specific OSEK implementations. The demo model provides examples of implementing Simulink blocks for OSEK APIs, and you can modify

`oseklib.tlc` and `osek_file_process.tlc` to provide detailed information about your OSEK implementation.

Note

- The `rtwdemo_osek` demo runs only on 32-bit Windows. (You can run the demo if the MATLAB `computer` command returns the value `PCWIN` on your system.)
- The `rtwdemo_osek` demo incorporates a subset of the Embedded Target for OSEK/VDX functionality. With the introduction of R2006b, Embedded Target for OSEK/VDX will no longer be available for purchase as a separate product.

New 'error' Hook Method for STF_make_rtw_hook.m

As of V4.5 (R2006b), the `STF_make_rtw_hook.m` hook file, which you can use to customize the target build process, supports a new `'error'` hook method. If used, Real-Time Workshop calls the `'error'` hook method when an error occurs during code generation or the build process. For example, you might use the new hook method to clean up any static or global data used by the hook file after an error occurs. Valid arguments include the hook method and model name.

For more information about the `'error'` hook method or the `STF_make_rtw_hook.m` hook file, see "Customizing the Target Build Process with the STF_make_rtw Hook File" in the Real-Time Workshop documentation.

Maximum Length Enforced for Auto-Generated Identifiers in Generated Code

In previous releases, some autogenerated identifiers in generated code were allowed to exceed the **Maximum identifier length** specified on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box. Generated identifiers that exceeded the **Maximum identifier length** did not honor the user setting and potentially were inconsistent

with ANSI C or MISRA guidelines requiring identifiers to be unique within a prescribed length (31 characters).

In R2006b, the user-specified **Maximum identifier length** is more rigorously enforced for autogenerated identifiers in generated code.

For limitations that apply, see "Identifier Format Control Parameters Limitations" in the Real-Time Workshop Embedded Coder documentation. For upgrade and compatibility considerations, see "Compatibility Considerations" on page 94.

For more information about the Real-Time Workshop Embedded Coder parameters for **Identifier format control** and their use, see "Customizing Generated Identifiers" and its subsection "Specifying Identifier Formats" in the Real-Time Workshop Embedded Coder documentation.

Compatibility Considerations

The following considerations for identifier format control apply when upgrading a Simulink model from an earlier release to this release:

- Some identifiers that were allowed to exceed the **Maximum identifier length** (on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box) in earlier releases are mangled in this release to conform to the maximum length. The mangling is most likely to occur in models with long names.

  To preserve the identifiers, you can increase the value of the **Maximum identifier length** parameter for the model.

- For models that use model referencing, some models that built successfully in previous versions might get build warnings or errors in R2006b, due to potential collisions between truncated identifier names that are exported by sibling models. To avoid name clashes in models that use model referencing, do one of the following:

  - Increase the **Maximum identifier length** for top and referenced models until the warnings or errors disappear. In this case, uniqueness of model names ensures that the exported identifier names do not clash.

  - Define a unique identifier naming scheme for each model. For example, you might define the **Identifier format control** string `m1$R$N$M` for

the first model, `m2$R$N$M` for the second model, and so forth. In this case, uniqueness of **Identifier format control** strings ensures that the exported identifier names do not clash.

- The identifier format control enhancements in this release introduce some naming differences in the autogenerated identifiers for
  - Stateflow and Embedded MATLAB temporary variables
  - Subsystem and model reference global identifiers and types

New Default Value for IncludeERTFirstTime Model Configuration Parameter

In R2006a, Real-Time Workshop Embedded Coder introduced the `IncludeERTFirstTime` model configuration parameter, which specifies whether Real-Time Workshop Embedded Coder is to include the `firstTime` argument in the *`model`*`_initialize` function generated for an ERT-based Simulink model.

In R2006b, the default value of this parameter has changed from `on` (include the `firstTime` argument) to `off` (do not include the `firstTime` argument). As a result, for ERT-based Simulink models newly created in R2006b, the code generated for the *`model`*`_initialize` function by default will no longer include the `firstTime` argument.

To include the `firstTime` argument in generated code, change the value of the `IncludeERTFirstTime` parameter to `on`. However, see the release note "Use of firstTime Argument to model_initialize Function to Be Discontinued" on page 96.

Note In R2006b, it is no longer required that the setting for `IncludeERTFirstTime` must be consistent throughout a model reference hierarchy.

Compatibility Considerations

For ERT-based Simulink models newly created in R2006b, the code generated for the *`model`*`_initialize` function by default will no longer include the `firstTime` argument. As a result, existing custom static main programs

that invoke *model*_initialize with the firstTime argument will need to be reconciled with the code generated for the *model*_initialize function. For example, you can

- Modify the invoking main program to remove code related to the firstTime argument (recommended).
- Change the value of the IncludeERTFirstTime model configuration parameter to on and regenerate code for the Simulink model.
- Modify the invoking main program to conditionally include or suppress the firstTime argument for the Simulink model. In the generated header file autobuild.h, the macro INCLUDE_FIRST_TIME_ARG will be set to 0 if the IncludeERTFirstTime parameter is set to off or 1 if the parameter is set to on. Inside the static main program, make sure to #include autobuild.h and then conditionally compile declarations and calls to the *model*_initialize function, based on the value of the INCLUDE_FIRST_TIME_ARG macro.

Use of firstTime Argument to model_initialize Function to Be Discontinued

In a future release, Real-Time Workshop Embedded Coder will no longer use the firstTime argument in a model's generated *model*_initialize function. For more information about this change, use the form at http://www.mathworks.com/contact_TS.html to contact The MathWorks Technical Support.

"Source of initial values" Option for MPT Data Objects Removed

In R2006b, the **Source of initial values** option for MPT data objects has been removed from the **Data Placement** pane of the Configuration Parameters dialog box. Although this option was visible in R2006a, it was obsolete and the setting had no effect.

Use Simulink.Signal objects to initialize signal values, as explained in "Initializing Signals and Discrete States" in the Simulink documentation.

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_importstruct`	Import externally defined parameters into Simulink. This model demonstrates how to generate code that accesses the fields of a data structure. The data structure is defined in legacy (hand-written) code and accessed via a pointer. This technique enables users to easily switch between complete sets of parameters at run time (for example, between reference and working versions).
`rtwdemo_osek`	Interface to the OSEK/VDX real-time operating system. For more information, see "New General-Purpose OSEK/VDX Real-Time Operating System (RTOS) Example" on page 92.
`rtwdemo_parentheses`	Set the style of parenthesization in generated code to be Minimum (only parentheses required by C syntax), Nominal (parentheses added to optimize readability), or Maximum (parentheses obviate C precedence, as required by MISRA).

New Reference Documentation

R2006b adds HTML and PDF reference documentation for Real-Time Workshop Embedded Coder functions and blocks.

Version 4.4.1 (R2006a+) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.4.1 (R2006a+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	No

Version 4.4 (R2006a) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.4 (R2006a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports Includes fixes	No

New features and changes introduced in this version are

• "New and Enhanced Demos" on page 111

Nonvirtual Subsystem Option for Generating Modular Function Code

This release provides a new subsystem option, **Function with separate data**, that allows you to generate modular function code for nonvirtual subsystems, including atomic subsystems and conditionally executed subsystems.

In previous releases, the generated code for a nonvirtual subsystem did not separate a subsystem's internal data from the data of its parent Simulink model. This could make it difficult to trace and test the code, particularly for nonreusable subsystems. Also, in large models containing nonvirtual subsystems, data structures could become large and potentially difficult to compile.

In this release, the Subsystem Parameters dialog box option **Function with separate data** allows you to generate subsystem function code in which the internal data for a nonvirtual subsystem is separated from its parent model and owned by the subsystem. As a result, the generated code for the subsystem is easier to trace and test. The data separation also tends to reduce the size of data structures throughout the model.

Note Selecting the **Function with separate data** option for a nonvirtual subsystem has no semantic effect on the parent Simulink model.

To be able to use this option,

- Your Simulink model must use an ERT-based system target file (requires a license for Real-Time Workshop Embedded Coder).
- Your subsystem must be configured to be atomic or conditionally executed (for more information, see "Systems and Subsystems" in the Simulink documentation).
- Your subsystem must use the `Function` setting for the **Real-Time Workshop system code** parameter.

To configure your subsystem for generating modular function code, you invoke the Subsystem Parameters dialog box and make a series of selections to display and enable the **Function with separate data** option. For details, see "Nonvirtual Subsystem Modular Function Code Generation" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Nonvirtual Subsystem Modular Function Code Limitations" in the Real-Time Workshop Embedded Coder documentation.

For more information about generating code for atomic subsystems, see the sections "Creating Subsystems" and "Generating Code and Executables from Subsystems" in the Real-Time Workshop documentation.

Exporting Function-Call Subsystems

This release adds new code generation capabilities for Simulink function-call subsystems. You can use these new capabilities to

- Automatically generate code for
  - A function-call subsystem that contains only blocks that support code generation
  - A virtual subsystem that contains only such subsystems and a few other types of blocks
- Optionally generate an ERT S-function wrapper for the generated code

For detailed descriptions of the new exporting capabilities, see "Exporting Function-Call Subsystems" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Function-Call Subsystems Export Limitations" in the Real-Time Workshop Embedded Coder documentation.

Identifier Format Control Parameters for Code Generation

This release adds several **Identifier format control** parameters that provide you finer control over the naming rules for identifiers created in generated code.

In previous releases, the **Symbol format** parameter on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box allowed you to specify one macro string that affected naming for a range of symbol types. In this release, several **Identifier format control** parameters allow you to exercise format control individually for

- Global variable names
- Global type names
- Field names of global types
- Subsystem method names
- Local temporary variable names
- Local block output variable names
- Constant macro names

To be able to use the new **Identifier format control** parameters, your Simulink model must use an ERT-based system target file (requires a license for Real-Time Workshop Embedded Coder).

For a description of the new **Identifier format control** parameters and their use, see "Customizing Generated Identifiers" and its subsection "Specifying Identifier Formats" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Identifier Format Control Parameters Limitations" in the Real-Time Workshop Embedded Coder documentation. For upgrade and compatibility considerations, see "Compatibility Considerations" on page 102.

Compatibility Considerations

The following considerations for identifier format control apply when upgrading a Simulink model from an earlier release to this release:

- Some identifiers that were allowed to exceed the **Maximum identifier length** (on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box) in earlier releases are mangled in this release to conform to the maximum length. The types of identifiers that potentially are affected are Simulink global variables, Simulink global types, local

variables, subsystem methods, and Simulink constant macros. The mangling is most likely to occur in models with long names.

To preserve the identifiers, you can increase the **Maximum identifier length**.

- By default, Simulink constant macro names are generated in a different format in this release than in earlier releases.

  To restore the earlier identifier format for Simulink constant macro names, you can specify the macro string `rtcP$N$M` for the **Constant macros** parameter on the **Real-Time Workshop/Symbols** pane. However, this setting causes Stateflow constant macros to be generated differently than in earlier releases.

- In earlier releases, symbols exported by referenced models were prefixed with the full model name to avoid name collisions between sibling models with similar long names. In this release, the **Maximum identifier length** is enforced for these exported identifiers, increasing the likelihood of a collision between truncated model names that did not occur in earlier releases.

  In this release, the software provides a warning when the current **Maximum identifier length** cannot accommodate the full model name in the exported identifiers. This warning indicates a potential name collision between sibling models.

  To avoid name clashes in models that use model referencing, do one of the following:

  - Increase the **Maximum identifier length** for top and referenced models until the warning disappears. In this case, uniqueness of model names ensures that the exported identifier names do not clash.
  - Define a unique identifier naming scheme for each model. For example, you might define the **Identifier format control** string `m1$R$N$M` for the first model, `m2$R$N$M` for the second model, and so forth. In this case, uniqueness of **Identifier format control** strings ensures that the exported identifier names do not clash.

New and Changed Memory Section Capabilities

This release provides new and changed memory section capabilities in Real-Time Workshop Embedded Coder. In previous releases, memory sections

could be applied only to data objects defined in custom storage classes, and memory section pragmas could surround only a contiguous block of all data objects in that section. This release adds enhancements that

- Provide an improved user interface for defining memory sections, including a new **Memory Sections** pane in the Configuration Parameters dialog box.
- Support memory sections for
  - Model-level functions
  - Model-level internal data
  - Subsystem functions
  - Subsystem internal data
- Allow pragmas to be applied separately to each function or data definition. The text of each pragma can contain the name of the definition to which it applies.

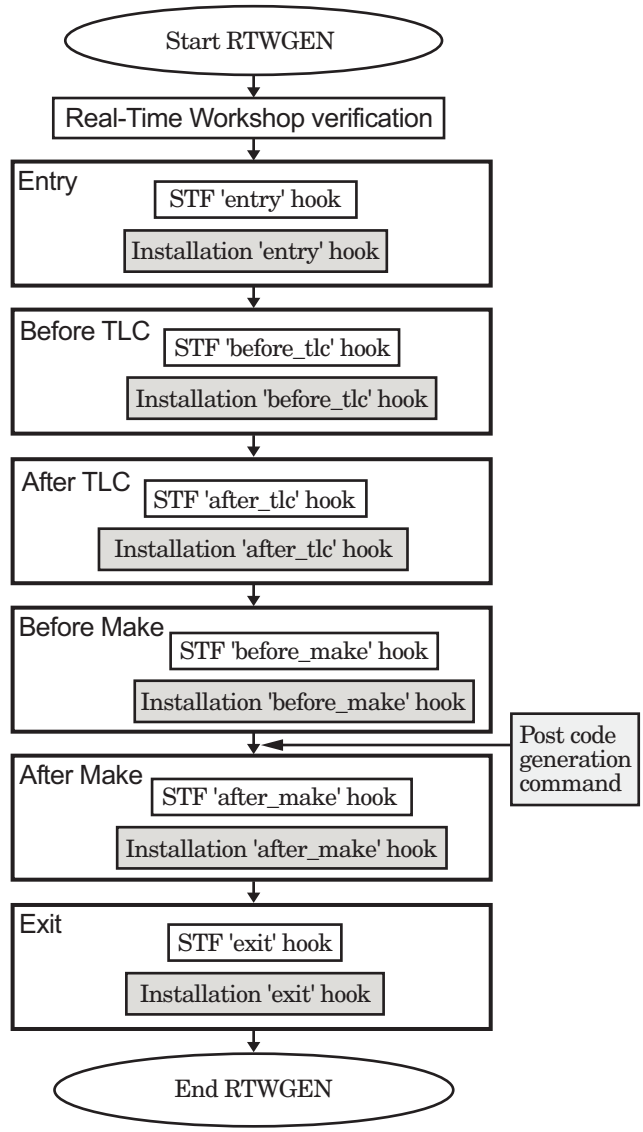
For detailed descriptions of the new memory section capabilities, see "Configuring Memory Sections" and the "Inserting Comments and Pragmas in Generated Code" chapter in the Real-Time Workshop Embedded Coder documentation.

New sl_customization API for Registering Real-Time Workshop Build Process Hooks

This release introduces an API, exercised through the Simulink customization file `sl_customization.m`, that allows you to register installation-specific hook functions to be invoked during the Real-Time Workshop build process.

The hook functions that you register through `sl_customization.m` complement System Target File (STF) hooks (described in "Customizing the Target Build Process with the STF_make_rtw Hook File") and post-code generation commands (described in "Customizing Post Code Generation Build Processing").

The following figure shows the relationship between installation-level hooks and the other available mechanisms for customizing the build process.



For details on the use of `sl_customization.m` to register build hook functions, see "Customizing the Target Build Process with sl_customization.m" in the Real-Time Workshop documentation. For more information on the Simulink

customization file `sl_customization.m`, see "Customizing the Simulink User Interface" in the Simulink documentation.

New sl_customization API for Customizing Data Objects

This release introduces an API, exercised through the Simulink customization file `sl_customization.m`, that allows you to register Simulink data object customizations, including

- User data types
- `mpt` user object types
- Data Object Wizard (DOW) user packages

This new registration mechanism replaces earlier mechanisms involving the files `custom_user_type_registration.m` (for creating user data types) and `custom_user_object_type_info.m` (for registering `mpt` user object types). A script is provided to convert existing instances of `custom_user_type_registration.m` and `custom_user_object_type_info.m` to `sl_customization.m` (see "Compatibility Considerations" on page 106).

For details on the use of `sl_customization.m` to customize Simulink data objects, see the following sections in the Real-Time Workshop Embedded Coder documentation:

- "Creating User Data Types"
- "Registering mpt User Object Types"
- "Customizing Data Object Wizard User Packages"

For more information on the Simulink customization file `sl_customization.m`, see "Customizing the Simulink User Interface" in the Simulink documentation.

Compatibility Considerations

In R2006a, data object customization mechanisms involving the files `custom_user_type_registration.m` and `custom_user_object_type_info.m` have been replaced by APIs exercised through the Simulink customization

file `sl_customization.m`. The older files and the mechanisms associated with them no longer have any effect.

If you have instances of `custom_user_type_registration.m` and `custom_user_object_type_info.m` that contain data object customizations that you want to preserve, you must convert the customizations to the new mechanism. You can use the MATLAB command `convert_custom_registration` to generate a corresponding `sl_customization.m` file.

When `convert_custom_registration` executes, it searches for `custom_user_type_registration.m` and `custom_user_object_type_info.m` on the MATLAB path, obtains custom registration information from the files, and generates `sl_customization.m` in the current work directory. If no custom registration information is found, `sl_customization.m` is not generated.

When you invoke `convert_custom_registration`, you optionally can provide an argument of 0, to specify that any existing `sl_customization.m` in the current work directory should be overwritten, or 1, to specify that any existing `sl_customization.m` in the current work directory should be renamed to `sl_customization_old.m`. The default action is to overwrite the existing file, if any. For example:

```
>> convert_custom_registration(1)  % Generate sl_customization.m without overwriting
```

mpt Signal Object Enhancements

Prior to R2006a, you could initialize a signal if you defined it as an `mpt` signal object. With the introduction of initial value support for Simulink signal objects, the support for signal object initialization for the two object types has merged. Initialization of `mpt` signal objects is semantically the same as it has been in previous releases. However, the merge has resulted in the following enhancements:

- You can initialize `mpt` signal objects for simulation and code generation. Prior to R2006a, you could initialize them for code generation only.
- Consistency checks are done to ensure that initial values you set match corresponding block parameters that specify initial conditions or values. Prior to R2006a, consistency checks were not performed.

- You can initialize signals that have an exported storage class. Prior to R2006a, you could initialize signals with an `mpt.Signal` class only.
- The **Source of initial values** option on the **Data Placement** pane of the Configuration Parameters dialog box is no longer needed. Although the option is visible, the setting has no effect.
- If you try to initialize an `mpt` signal object that represents a constant sample time, Simulink now ignores the initial value and generates a warning. Prior to R2006a, you could initialize such an `mpt` signal object without being notified of the error.

Note The code generated for `mpt` signal object initialization might vary slightly from code generated in previous releases.

For more information about the new initial value support for Simulink signal objects, see "Using Signal Objects to Initialize Signals and Discrete States" in the Simulink documentation and "Using Signal Objects to Initialize Signals and Discrete States" in the Real-Time Workshop documentation. For details on `mpt` data objects, see "Creating Simulink and mpt Data Objects" in the Real-Time Workshop Embedded Coder documentation. For information on options for controlling how signals in a model are stored and represented in generated code, see "Signal Considerations" in the Real-Time Workshop documentation.

New IncludeERTFirstTime Model Configuration Parameter

V4.4 (R2006a) Real-Time Workshop Embedded Coder introduces a new model configuration parameter, `IncludeERTFirstTime`. This parameter specifies whether Real-Time Workshop Embedded Coder is to include the `firstTime` argument in the *`model`*`_initialize` function generated for the model. By default, the parameter is set to `on` to include the argument.

Note The setting for `IncludeERTFirstTime` must be consistent throughout a model reference hierarchy.

New ERTFirstTimeCompliant Target Configuration Parameter

V4.4 (R2006a) Real-Time Workshop Embedded Coder introduces a new target configuration parameter, `ERTFirstTimeCompliant`. This parameter indicates whether a target supports the ability to use the `IncludeERTFirstTime` model configuration parameter to control inclusion of the `firstTime` argument in the model's *`model`*`_initialize` function. You set this parameter in the `SelectCallback` function.

By default, the parameter is set to `off` for custom and non-ERT targets, and `on` for the ERT target. (The ERT target has been enhanced to support conditional inclusion of `firstTime` in the *`model`*`_initialize` function.)

For more information about how to configure a custom embedded target to support `firstTime` argument control, see "Supporting firstTime Argument Control" in the Real-Time Workshop documentation.

firstTime Argument to model_initialize Function

In a future release, Real-Time Workshop Embedded Coder will no longer use the `firstTime` argument in a model's generated *`model`*`_initialize` function. For more information about this change, use the form at http://www.mathworks.com/contact_TS.html to contact The MathWorks Technical Support.

Data Object Wizard Script for Labeling Root I/O Signals Based on Inport/Outport Names

This release includes a Data Object Wizard script, `propagate_rootio_signal_names`, that labels a Simulink model's unlabeled root I/O signals based on the corresponding root Inport/Outport names. Signals that are already labeled are not affected.

The script takes the name of a Simulink model in the current working directory as an input argument. It returns

- `1` if it completed without error
- `0` and an error message if it failed to complete due to an error
- `-1` and an error message if it completed but found a naming inconsistency

The script locates unlabeled signals connected with root I/O ports in the specified model. Each unlabeled signal will be labeled using its port name, provided that the port name is a valid C identifier, is not a C keyword, and does not conflict with other signal and parameter names in the model. If the specified model is not already open, the script opens the model for viewing. You can examine the modifications and decide whether to save the model with the changes.

For a simple demonstration of this functionality, launch `rtwdemo_counter` and save the model to your current working directory as `rtwdemo_counter_test`. You can then run the `propagate_rootio_signal_names` script on the saved model using either of the following MATLAB commands:

```
propagate_rootio_signal_names('rtwdemo_counter_test')

[status,errMsg] =  propagate_rootio_signal_names('rtwdemo_counter_test')
```

In the resulting display of `rtwdemo_counter_test`, signal labels will have been added to the model diagram. You can relaunch the original `rtwdemo_counter` and visually compare `rtwdemo_counter` with `rtwdemo_counter_test`.

MISRA C Compliance Enhanced for Multiport Switch Block Code

R2006a improves the MISRA C compliance of generated code by generating a default switch case statement for the Multiport Switch block. For a description of this block, see Multiport Switch in the Simulink Reference.

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_export_functions`	Export function-call subsystems
`rtwdemo_memsec`	Insert pragmas for functions and data in generated code

The following demos have been enhanced:

- `rtwdemo_namerules`
- `rtwdemo_symbols`

Version 4.3 (R14SP3) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.3 (R14SP3):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports Includes fixes	No

New features and changes introduced in this version are

Data Type Replacement

This release provides the ability to replace built-in data type names with user-defined replacement data type names in the generated code for ERT target models.

As in previous releases, you can register user-defined data types and specify their associated header files using mechanisms described in "Managing Data Definitions and Declarations With the Data Dictionary" of the Real-Time Workshop Embedded Coder documentation. User-defined data types can be automatically created as `Simulink.AliasType` objects in the base workspace.

This release augments the existing mechanisms for registering user-defined data types by adding:

- The **Data Type Replacement** pane, a new subpane under the **Real-Time Workshop** pane of the Configuration Parameters dialog box. This pane provides an improved user interface for mapping built-in data types to user-defined replacement data types.
- Consistency checks to ensure that your specified data type replacements are consistent with your model's data types.
- Many-to-one data type replacement, the ability to map multiple built-in data types to one replacement data type in generated code. For example, built-in data types `uint8` and `boolean` could both be replaced in your generated code by a data type `U8` that you have previously defined.

Data type replacement is available for code generated using Real-Time Workshop Embedded Coder, whether from Simulink, Stateflow charts, or Embedded MATLAB blocks.

For details on specifying replacement data types for a Simulink model, see "Replacing Built-In Data Type Names in Generated Code" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Data Type Replacement Limitations" in the Real-Time Workshop Embedded Coder documentation.

HeaderFile Property Now Optional as Part of GetSet Data Object

In previous releases, a `Simulink.Signal` or `mpt.Signal` object of custom storage class `GetSet` was required to specify its `HeaderFile` property. The specified header file was then added as an `#include` in the generated code.

This release makes it optional to specify the `HeaderFile` property on data objects of the `GetSet` custom storage class. This accommodates users who prefer to use a model's custom code options to include header files.

Note If you omit the `HeaderFile` property for a `GetSet` data object, you must specify a header file by an alternative means, such as the **Header file** field of the **Real-Time Workshop/Custom Code** pane of the Configuration Parameters dialog box. Otherwise, the generated code might not compile or might function improperly.

Data Object Wizard Enhancements

The Data Object Wizard has been enhanced with new search options for including or omitting the following types of data objects for searches:

Alias types
Block outputs
Data stores
Parameters
Root inputs
Root outputs
States

For details on these enhancements, see "Data Object Wizard" in the Simulink documentation.

Global Data Stores Can Be Initialized Using mpt.Signal Object's RTWInfo.InitialValue Property

Global data stores may be defined in the base workspace using `mpt.Signal` objects (as well as `Simulink.Signal` objects or any of the subclasses of `Simulink.Signal`). In Release 14SP3, you can use the `RTWInfo.InitialValue` property of the `mpt.Signal` object to initialize a global data store.

If you set the `RTWInfo.InitialValue` property of the `mpt.Signal` object to a nonempty value, the value of that property becomes the initial condition of the global data store. If the `InitialValue` property of the object is empty (`[]`), the initial value of the global data store remains 0 (for example, false for Boolean data).

ERT Automatic Configuration Changes

If you generate code for ERT-based models that use the automatic model configuration feature, you should be aware of the following auto-configuration related changes in this release. If you supply your own script for ERT auto-configuration, you should consider modifying your code to take advantage of these changes.

- The `ert_config_opt` auto-configuration function that is invoked at the `'entry'` hook during code generation now additionally runs at target selection time (that is, when you use the **Real-Time Workshop** pane of the Configuration Parameters dialog box to select an auto-configuration target).
- To support this dual invocation, the `ert_config_opt` function now takes variable input arguments. The second argument still specifies `'optimized_fixed_point'` or `'optimized_floating_point'` as before, but the first argument now specifies either a model name, for `'entry'-hook` invocation, or a configuration set handle, for target-selection invocation. (The function is located in the file *`matlabroot`*`/toolbox/rtw/targets/ecoder/ert_config_opt.m`.)
- The `'entry'` hook in the example hook file `ert_make_rtw_hook.m` has added code to report changes in the configuration set caused by invoking `ert_config_opt` (via gateway routine `ert_auto_configuration`) during code generation. (The example `'entry'` hook is located in the file *`matlabroot`*`/toolbox/rtw/targets/ecoder/ert_make_rtw_hook.m`.)

Compatibility Considerations

If you supply your own auto-configuration script in place of the default version of `ert_config_opt`, your auto-configuration code will continue to be invoked and execute at the `'entry'` hook. However, to additionally run your code at target selection time, you must modify your script to support the variable input arguments in the manner shown in `ert_config_opt.m`.

Documentation Enhancements

The following areas of the Real-Time Workshop Embedded Coder documentation have been corrected or improved:

- 'Basic Tutorial' in the Module Packaging Features documentation
- 'Comparison of a Template and Its Generated File' in the Module Packaging Features documentation
- 'mpt Parameter and Signal Properties' in the Module Packaging Features documentation

Version 4.2.1 (R14SP2+) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.2.1 (R14SP2+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	No

Version 4.2 (R14SP2) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.2 (R14SP2):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports Includes fixes	No

New features and changes introduced in this version are

C++ Target Language Support

This release introduces support for generating C++ code. The primary use for this feature is to facilitate integration of generated code with legacy or custom user code written in C++. For detailed information about C++ code generation, see "Choosing and Configuring a Compiler" and "Integration Options" in the Real-Time Workshop documentation.

External Mode Support for ERT VxWorks Example Target

The ERT VxWorks® example target now includes full support for 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. With this release, you can generate code to support external mode communication between host (Simulink) and ERT VxWorks target systems. For more information, see "Using External Mode with the ERT Target" in the Real-Time Workshop Embedded Coder documentation.

Custom Storage Classes with ERT S-Functions

Custom storage classes (CSCs) now can be used with ERT S-functions. This capability was disabled in Version 4.0, Release 14, and is reenabled in this release.

For more information, see "Creating and Using Custom Storage Classes" in the Real-Time Workshop Embedded Coder documentation.

Consistency Checking for ERT Target Options

Pre-model-compilation consistency checking has been added to detect and warn against conflicting combinations of ERT target configuration options. (Configuration options that are available for the ERT target are described in "Mapping Application Objectives to Model Configuration Parameters" in the Real-Time Workshop Embedded Coder documentation.)

Error messages now are issued for the following conflicts:

- **GRT compatible call interface** (`GRTInterface`) is on and **Support floating-point numbers** (`!PurelyIntegerCode`) is off
- **MAT-file logging** (`MatFileLogging`) is on and **Support floating-point numbers** (`!PurelyIntegerCode`) is off
- **Support non-finite numbers** (`SupportNonFinite`) is off and **MAT-file logging** (`MatFileLogging`) is on

- **GRT compatible call interface** (`GRTInterface`) is on and **Single output/update function** (`CombineOutputUpdateFcns`) is on
- **MAT-file logging** (`MatFileLogging`) is on and **Terminate function required** (`IncludeMdlTerminateFcn`) is off
- **MAT-file logging** (`MatFileLogging`) is on and **Suppress error status in real-time model data structure** (`SuppressErrorStatus`) is on

Warning messages now are issued for the following conflicts:

- **Support non-finite numbers** (`SupportNonFinite`) is off and **Support non-inlined s-functions** (`SupportNonInlinedSFcns`) is on
- **Support non-finite numbers** (`SupportNonFinite`) is on and **Support floating-point numbers** (`!PurelyIntegerCode`) is off
- **Support non-inlined S-functions** (`SupportNonInlinedSFcns`) is on and **Support floating-point numbers** (`!PurelyIntegerCode`) is off

Model Explorer "Alias Override Naming Rule" Check Box Removed

Before this release, the Model Explorer allowed you to select the **Alias override naming rule** option for an `mpt` data object. As explained in "Defining Data Representation and Storage for Code Generation", this resulted in the name that you typed in the **Alias** field overriding the global naming rule for the selected data object. This only applied to `mpt` data objects, not to Simulink data objects.

This release removes the **Alias overrides naming rule** check box. Now, the override works the same way for `mpt` and for Simulink data objects: As explained in the documentation, if the **Alias** field is empty, the global naming rule (that you select on the Configuration Parameters dialog box) applies to all data objects. But if you do specify a name in the **Alias** field, this overrides the naming rule for that data object. There is no need for the check box.

Model Explorer Data Object Header File No Longer Generated If Header File Name Is Not Specified

Before this release, when you specified a **Definition file** name on the Model Explorer dialog box for a data object and did not specify a **Header file** name,

the code generator generated a header file in which it declared the data object. The code generator used the same name for the header file (for example, `data.h`) as you specified for the definition file (for example, `data.c`).

With this release, when you specify a **Definition file** name and do not specify a **Header file** name, the code generator does not generate a header file. The code generator declares the data object according to the global naming rule. In this case, the code generator assumes that you do not want it to generate the header file.

Enhanced MPF Documentation of Managing Data Dictionary

This release restructures the 'Managing the Data Dictionary' chapter in Module Packaging Features. The revised material explains how to create Simulink data objects using the Data Object Wizard, and compares this with creating `mpt` data objects.

File custom_user_type_registration.m No Longer Automatically Called During Code Generation

Beginning with Real-Time Workshop Embedded Coder 4.2 (R14SP2), the file `custom_user_type_registration.m`, which you provide if you want to register user-defined data types, no longer is called automatically during code generation. Instead, after modifying and saving your `custom_user_type_registration.m` file, you must create the `Simulink.AliasType` objects corresponding to your user-defined data types *before* generating code. For a description of the R14SP2 and R14SP3 procedure for using `custom_user_type_registration.m` to register user-defined data types, see 'Creating User Data Types' in the R14SP2 or R14SP3 Real-Time Workshop Embedded Coder Module Packaging Features document.

Compatibility Considerations

The following compatibility consideration applies if you are upgrading from V4.1 (R14SP1) to V4.2 (R14SP2), V4.2.1 (R14SP2+), or V4.3 (R14SP3). If you are upgrading to V4.4 (R2006a) from V4.1 or later, see the release notes for "Version 4.4 (R2006a) Real-Time Workshop® Embedded Coder Software" on page 99 .

If you modified and saved `custom_user_type_registration.m` in V4.1 (R14SP1), you must now create the `Simulink.AliasType` objects corresponding to your user-defined data types before generating code for your model. For example, you can:

- Invoke the MATLAB function `ec_create_type_obj` to programmatically create all the required `Simulink.AliasType` objects
- Create `Simulink.AliasType` objects one at a time by selecting **Add > Simulink.AliasType** in the Model Explorer
- Create `Simulink.AliasType` objects one at a time by entering the MATLAB command *`userdatatype`* `= Simulink.AliasType`, where *`userdatatype`* is a user-defined data type registered in `custom_user_type_registration.m`

Version 4.1 (R14SP1) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.1 (R14SP1):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Fixed bugs	No

New features and changes introduced in this version are described here:

Significant Documentation Corrections

Documentation for Real-Time Workshop Embedded Coder in Version 4.1 corrects errors, omissions, and inconsistencies in the Version 4.0 documentation. Topics affected most significantly by these changes include the following:

- Discussion of data structures and code modules
- Description of the static main program module
- Discussion of the interaction between **Simulink block comments** and **Simulink block description** configuration parameters
- Custom storage classes
- Template makefile modifications for supporting model reference features
- Description of makefile variable `SYS_TARGET_FILE`
- Custom target configuration tutorial
- Interfacing an integrated development environment
- Tradeoffs for device driver development
- Writing a device driver C MEX S-function
- Single-model approach to using device drivers in simulation

- Addition of a basic tutorial to the “Getting Started” chapter of Module Packaging Features
- Addition of data placement rules in the “Referenced Tables” appendix of Module Packaging Features

Version 4.0 (R14) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 4.0 (R14):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—See "Upgrading from R13SP1+ or R13SP2" on page 142 and "Generating R13SP1+ or R13SP2 Code From ERT-Based Simulink Models Created In R14 or Later" on page 151. See also Summary.	Fixed bugs	No

New features and changes introduced in this version are

Expanded Documentation Collection

The Real-Time Workshop Embedded Coder documentation collection has been expanded and includes following documents:

User's Guide	Describes Embedded Real-Time (ERT) model execution, timing, and task management; the `rtModel` data structure; how to interface to and call model code; ERT code generation options and optimization tips; custom storage classes; and advanced code generation techniques.
"Module Packaging Features"	Describes features teams of engineers can apply to generate ANSI/ISO C production code and executables for large-scale, multimodel control system applications.
"Developing Embedded Targets"	Describes requirements and implementation details for creating custom embedded targets based on the ERT target. It includes detailed information on the structure and organization of target directories, system target files, and template makefiles; how to support the Real-Time Workshop model referencing feature; how to implement device drivers; and operation of the build process and how to customize it.

New ERT Target Options User Interface

You can configure ERT target code generation options in the new Simulink Model Explorer and Configuration Parameters dialog box. Before working with the ERT target in this new environment, you should become familiar with

- Configuration sets and how to view and edit them in Model Explorer and the Configuration Parameters dialog box. See *Simulink User's Guide* for details.
- The general Real-Time Workshop code generation options and use of the System Target File Browser. See the *Real-Time Workshop User's Guide* for details.

Some panes of the new Configuration Parameters dialog box (for example, the **Templates** and **Interface** panes) contain only ERT-specific options. Others, such as the **Real-Time Workshop** pane, display a combination of general Real-Time Workshop options and ERT target options.

Note If you have developed a custom target based on the ERT target (or any other Real-Time Workshop target) see "Defining and Displaying Custom Target Options" on page 144 for a discussion of compatibility issues that may affect the appearance and operation of your target.

The following table summarizes new and revised ERT target code generation options.

Pane and Subpane	Option	Usage
Real-Time Workshop	**Include hyperlinks to model**	Include or suppress hyperlinks from generated code to the source blocks in the model.
	Launch report after code generation completes	Automatically display the HTML report in a MATLAB web browser window after code generation.

Pane and Subpane	Option	Usage
Real-Time Workshop > Comments	**Simulink block descriptions**	Include text specified in the **Description** field of Block Properties dialogs as comments in the generated code for the corresponding blocks.
	Stateflow object descriptions	Include text specified in the **Description** field of state object Properties dialogs as comments in the generated code for the corresponding objects.
	Simulink data object descriptions	Include text specified in the **Description** field of object properties defined in the Simulink Model Explorer as comments in the generated code for the corresponding objects.
	Custom comments (mpt objects only)	Include comments just above signals and parameter identifiers in the generated code as specified in an MATLAB code or TLC function.
Real-Time Workshop > Symbols	**Symbol format**	Customize generated symbols for signals, parameters, and other objects in a model based on a macro string that specifies whether and in what order substrings are to be included in the symbols.

Pane and Subpane	Option	Usage
	Minimum mangle length	Specify the minimum number of characters to be used for name mangling strings generated and applied to symbols to avoid name collisions.
	Maximum identifier length	Specify the maximum number of characters that can be used in generated function, `typedef`, and variable names.
	Generate scalar inline parameters as	Control how scalar inlined parameter values are expressed in generated code.
	#define naming	Define rules that change the names of a model's parameters that have a storage class of `Define`.
	Parameter naming	Define rules that change the names of all of a model's parameters.
	Signal naming	Define rules that change the names of a model's signals.
Real-Time Workshop > Interface	**Target floating-point math environment**	Specify the math library to be used. Support for the GNU C math library was added as an option.
	Support floating-point numbers	Enable or suppress the generation of floating-point numbers. To generate pure integer code, clear this option.
	Support complex numbers	Enable or suppress the generation of complex numbers.

Pane and Subpane	Option	Usage
	Support non-finite numbers	Enable or suppress the generation of nonfinite numbers.
	Support absolute time	Generate integer counters that provide absolute or elapsed time values for blocks in the model.
	Support continuous time	Generate code for continuous time blocks.
	Pass root-level I/O as	Control how input and output values at the root level of the model are passed to the *model*_step function. Enable only if you select **Generate reusable code**.
	GRT compatible call interface	Use ERT features with a GRT-based custom target that has a main program based on grt_main.c.
	Data Exchange: Interface	Generate external mode support code, ASAP2 data files, or C API code for monitoring signals and parameters.
Real-Time Workshop > Templates	**Source file (*.c) template**	Create or edit a code template.
	Source file (*.h) template	Create or edit a data template.
	File customization template	Specify a custom file processing (CFP) template, which organizes generated code into sections -- includes, typedef S-functions, and so on.

Pane and Subpane	Option	Usage
	Generate an example main program	Control whether `ert_main.c` is generated.
	Target operating system	Generate a bareboard main program designed to run under control of a real-time clock without a real-time operation system or a fully commented example showing how to deploy the code under the VxWorks real-time operating system.
Real-Time Workshop > Data Placement	**Data definition**	Specify whether data is to be defined in the generated source file or in a single separate header file.
	Data reference	Specify whether data is to be declared in the generated source file or in a single separate header file
	Module naming	Name the generated module using the same name as the model or a user-specified name.
	Signal display level	Specify whether to declare signal data objects as global data in the generated code.
	Parameter tune level	Declare a parameter data object as tunable global data in the generated code.
	#include file delimiter	Specify the #include file delimiter to be used in generated files that contain the #include preprocessor directive for MPF data objects.

Pane and Subpane	Option	Usage
	Source of initial values	Specify the source that initializes the model's signals in the generated code.
Optimization	**Application lifespan (days)**	Minimize the allocation of memory for absolute and elapsed time counters generated for blocks that require an absolute or elapsed time value. The word size of the counters is allocated optimally to accommodate the maximum value that you specify for this parameter.
	Remove root-level I/O zero initialization	Specify whether initialization code for root-level inports and outports with a value of zero are to be generated. Previously labeled **Initialize external data**. Default is now cleared rather than set.
	Remove internal state zero initialization	Specify whether initialization code for work structures, such as block states and block outputs, are to be generated. Previously labeled **Initialize internal data**. Default is now cleared rather than set.

Pane and Subpane	Option	Usage
	Use memset to initialize floats and doubles	Specify whether internal storage, regardless of type, is to be cleared to the integer bit pattern 0 or the `memset` function is to set float and double storage to 0.0. Previously labeled **Initialize Floats and Doubles** to 0.0. Default is now cleared rather than set.
	Optimize initialization code for memory reference	Specify whether a model contains an enabled subsystem and will be referred to by another model with a Model block. If these conditions exist, the option should be cleared.
	Remove code that protects against division arithmetic exceptions	Suppress generation of code that guards against fixed-point division by zero exceptions.

Note The **Symbol format** option supports all functions previously implemented by the **Prefix model name to global identifiers**, **Include system Hierarchy Number in Identifiers**, and **Include data type acronym in identifier** options in a more compact form. The **Symbol format** option replaces all these options. However, existing models will continue to generate code that respects the settings of the previous options.

Detailed descriptions of options specific to the ERT target are provided in:

- The "Preparing Models for Code Generation" chapter of the Real-Time Workshop Embedded Coder documentation.
- The "Defining Data Representation and Storage for Code Generation" document.

GRT and ERT Target Unification

Release 14 introduced Generic Real-Time (GRT) and Embedded Real-Time (ERT) target unification enhancements. The enhancements include the following changes to the underlying technology for Real-Time Workshop and Real-Time Workshop Embedded Coder.

- Both products use a common format for backend generated code.
- The feature list common to both products is expanded.
- Some features and efficiencies formerly exclusive to the ERT target are now available to the GRT target. Conversely, the ERT target now supports some features that were previously available only with the GRT target.
- Conversion from GRT-based targets to ERT-based targets is greatly simplified.

See the "Version 6.0 (R14) Real-Time Workshop Software" Release Notes for a high-level overview and comparison of feature enhancements and compatibility issues that result from target unification in Real-Time Workshop 6.0 and Real-Time Workshop Embedded Coder 4.0.

Support for Continuous Time Blocks

The ERT target now supports code generation for continuous time blocks. If you select the **Support continuous time** option in the **Interface** subpane under **Real-Time Workshop** on the Configuration Parameters dialog box, you can use any such blocks in your models, without restriction.

Note that use of certain continuous time blocks is not recommended for production code generation for embedded systems. The Simulink Block Data Type Support table summarizes characteristics of blocks in the Simulink and Fixed-Point block libraries, including whether or not they are recommended for use in production code generation. To view this table, execute the following MATLAB command:

```
showblockdatatypetable
```

Then, refer to the "Recommended for Production Code?" column of the table.

Support for Continuous Solvers

The ERT target now supports continuous solvers. You can select any solver from the **Solver** menu on the **Solver** pane of the Configuration Parameters dialog box. However, note that the solver **Type** must be `fixed-step` for use with the ERT target, as in previous releases.

Note Custom targets must be modified to support continuous time. The required modifications are described in "Supporting Continuous Time in Custom Targets" on page 146.

Support for Noninlined S-Functions

In previous releases, the ERT target required that all S-functions in a model be inlined with a corresponding TLC file for code generation. This restriction has been removed. Models can now include noninlined S-functions.

To enable support for noninlined S-functions, select the **Support non-inlined S-functions** option in the **Interface** subpane under **Real-Time Workshop** on the Configuration Parameters dialog box.

Note that inlining S-functions is often advantageous in production code generation, for example in implementing device drivers.

Module Packaging Features

Module Packaging Features (MPF) are a major subcomponent of the Real-Time Workshop Embedded Coder. These features enable teams of engineers to apply the Real-Time Workshop Embedded Coder for generating ANSI/ISO production code and executables for large-scale, multimodel control system applications.

"Defining Data Representation and Storage for Code Generation" describes these features in detail. This note summarizes the capabilities of MPF.

Introduction

With MPF, you can

- Package the generated code into the desired number of `.c` and `.h` files.
- Control the *internal* organization of each of the generated files. For example, for readability, your company may have software standards that define where to place comments and sections of code within files.
- Control whether or not the generated files contain definitions for a model's global identifiers. If such definitions exist, you determine the files in which the code generator places them. Also, you can specify the generated files where the code generator places global data (`extern`) declarations.

In addition to meeting the preceding packaging needs, you can use MPF to

- Register user-defined data types.
- Customize comments.
- Locate variables in target memory where desired.

You implement these features with available dialogs, user-definable templates, and MATLAB scripts.

MPF Feature Summary

This section summarizes the module packaging features introduced in Real-Time Workshop Embedded Coder Version 4.0. MPF allows you to

- Select or define MPF template files. You can generate the desired `.c` and `.h` files and organize them the way you want. Also, these templates include template symbols whose locations in a template file determine where comments and code is located *in* the individual generated files.
- Manage the code generation data dictionary. This allows
  - Registering user-defined data types
  - Importing data objects into the code generation data dictionary from a `.mat` file of a previous Simulink session or from an external data dictionary (such as an Excel® file)
  - Adding Simulink data objects using the **Data Object Wizard**
  - Changing the alphabetical case and spellings that identifier names have in the generated code

- Select additional miscellaneous and advanced options. These include
  - Instructing the code generator to use the angle-bracket delimiter (for multiple data objects), instead of the double-quotation delimiter.
  - Selecting the source that initializes each of the model's signals in the generated code.
  - Adding property values of a selected data object as a comment in a generated file above the identifier of the data object.
  - Adding a comment to the model using the Simulink DocBlock so that this comment appears in the generated file where desired.
- Manage file placement of data declarations. You can determine whether or not the generated files contain defining declarations for a model's global identifiers. If defining declarations exist, you can determine the files in which the code generator places them. Also, you can determine the files where the code generator places global data reference (`extern`) declarations.

ASAP2 File Generation Changes

ASAP2 file generation is now available to all Real-Time Workshop targets. The documentation for this feature has been relocated to "Generating an ASAP2 File" in the Real-Time Workshop documentation.

Code Generation with User-Defined Data Types

Real-Time Workshop Embedded Coder now supports user-defined data type objects in code generation. Supported objects include objects of the following classes:

- `Simulink.NumericType`
- `Simulink.StructType`
- `Simulink.Bus`
- `Simulink.Aliastype`

In code generation, you can use user-defined data type objects to map your own data type definitions to Simulink built-in data types, and to generate

`#include` directives specifying your own header files, containing your data type definitions.

See the "Optimizing Generated Code" chapter of the Real-Time Workshop Embedded Coder documentation for details.

Enhanced Custom Storage Classes

The Real-Time Workshop Embedded Coder has extended the built-in storage classes provided by Real-Time Workshop. The Real-Time Workshop Embedded Coder now includes:

- A set of *custom storage classes* (CSCs). CSCs are designed to be useful in code generation for embedded systems development. The new enhanced and expanded CSC functionality has been incorporated into the `Simulink.Signal` and `Simulink.Parameter` classes. This simplifies code generation with CSCs, since you can use familiar signal and parameter objects for this purpose.
- The new Custom Storage Class Designer (`cscdesigner`) tool. The Custom Storage Class Designer lets you define additional CSCs that are tailored to your code generation requirements. The Custom Storage Class Designer provides a graphical user interface that lets you implement CSCs quickly and easily. You can use your CSCs in code generation immediately, without any TLC or other programming.

CSCs give you extended control over the constructs required to represent data in an embedded algorithm. For example, you can use CSCs to

- Define structures for storage of parameter or signal data.
- Conserve memory by storing Boolean data in bit fields.
- Integrate generated code with legacy software whose interfaces cannot be modified.
- Generate data structures and definitions that comply with your organization's software engineering guidelines for safety-critical code.

See the "Creating and Using Custom Storage Classes" chapter of the Real-Time Workshop Embedded Coder User's Guide for a detailed description of CSCs and the Custom Storage Class Designer.

Compatibility with Previous CSCs

In prior releases, CSCs were implemented via special `Simulink.CustomSignal` and `Simulink.CustomParameter` classes. We recommend that you consider replacing `Simulink.CustomSignal` and `Simulink.CustomParameter` objects in your models with equivalent `Simulink.Signal` and `Simulink.Parameter` objects.

Minor changes have been made in the `Simulink.CustomSignal` and `Simulink.CustomParameter` classes. See "Custom Storage Class Compatibility Issues" on page 143 for information on these changes.

More Efficient Multi-Rate Multitasking Code Generation

Real-Time Workshop Embedded Coder now generates significantly faster code for multirate multitasking models.

For multirate multitasking models, Real-Time Workshop Embedded Coder uses a strategy called *rate grouping*. Rate grouping generates separate *`model`*`_step` functions for the base rate task and each subrate task in the model. The function naming convention for these functions is

```
model_stepN
```

where *`N`* is a task identifier. For example, for a model named `my_model` that has three rates, the following functions are generated:

```
void my_model_step0 (void);
void my_model_step1 (void);
void my_model_step2 (void);
```

Each *`model`*`_step`*`N`* function executes all blocks sharing `tid` *`N`*; in other words, all block code that executes within task *`N`* is grouped into the associated *`model`*`_step`*`N`* function.

For other cases, Real-Time Workshop Embedded Coder generates a single *`model`*`_step` function. This *`model`*`_step` function uses the same scheduling technique (called *rate guarding*) as in previous versions of the product. When rate guarding is used, a task identifier is passed in to the *`model`*`_step` function.

To take advantage of rate grouping for existing multirate multitasking models, you must regenerate code, including the main program, `ert_main.c`.

See the "Developing Models for Code Generation" chapter of the Real-Time Workshop Embedded Coder documentation for a complete discussion of rate grouping.

More Efficient Task Scheduling for RTOS Targets

Using a new `rtmStepTask` macro, targets that employ the task management mechanisms of an RTOS can eliminate certain redundant scheduling calls during the execution of tasks in a multirate, multitasking model, thereby improving performance of the generated code.

The redundant scheduling calls are still generated by default for backward compatibility. However, you can suppress them by adding the following TLC variable `definition` to your system target file before the `%include "codegenentry.tlc"` statement:

```
%assign SuppressSetEventsForThisBaseRateFcn = 1
```

For more details on this feature, see "Optimizing Task Scheduling for Multirate Multitasking Models on RTOS Targets" in the Real-Time Workshop Embedded Coder documentation.

New Callbacks Defined for System Target Files

The Release 14 API for system target file callbacks provides three new callback functions for use in system target files. Unlike `rtwoptions` callbacks, these functions are associated with the target, not with its individual options. The callbacks are installed as fields in the `rtwgensettings` structure of the system target file. The callbacks, summarized in the next table, are fully described in "Customizing System Target Files" in the Real-Time Workshop documentation.

Callback Function...	Is Triggered...
`rtwgensettings.SelectCallback`	During model loading and when you select a target with the System Target File browser.
`rtwgensettings.ActivateCallback`	When the active configuration set of the model changes. This could happen during model loading and when you change the active configuration set.
`rtwgensettings.postapplyCallback`	When you click **Apply** or **OK** after editing options in the Configuration Parameters dialog box. The function is called after the changes have been applied to the configuration set.

Note If you have developed a custom target and you want it to be compatible with model referencing, you must implement a `SelectCallback` function to declare model reference compatibility. See "Supporting Model Referencing" in the Real-Time Workshop documentation.

New Option to Control Template Makefile Output Display

A new template makefile option lets you control whether or not template makefile output is displayed during the build process. To enable makefile output display at all times (regardless of the setting of the **Verbose build** option in the Real-Time Workshop **Debugging** pane) add the following macro to your template makefile:

```
VERBOSE_BUILD_OFF_TREATMENT = PRINT_OUTPUT_ALWAYS
```

When you configure your template makefile this way, the **Verbose build** option controls the display of other build process output (such as TLC messages), but template makefile output is always displayed.

You should add this macro in the template makefile section that includes other macros, such as `BUILD_SUCCESS`.

Demo Updates

This release includes a major update and reorganization of the Real-Time Workshop and Real-Time Workshop Embedded Coder demo collection. If you are reading this document online in the MATLAB Help browser, you can open the demo suite by clicking this link: rtwdemos.

Alternatively, you can access the demo suite by typing the name of the demo library at the MATLAB command prompt:

```
rtwdemos
```

Upgrading from R13SP1+ or R13SP2

This section discusses the following issues pertaining to upgrades from Real-Time Workshop Embedded Coder V3.2 (R13SP1+) or V3.2.1 (R13SP2) to V4.0 (R14):

TMF File Update Required for Use with Release 14 or Higher If Supporting ERT S-Function Generation

To use a Release 13 based TMF that supports ERT S-function generation with Release 14 or higher, you must update the TMF to include the following definitions:

```
LIBFIXPT=$(MATLAB_ROOT)\extern\lib\win32\microsoft\msvc50\libfixedpoint.lib
LIBS = $(LIBS) $(LIBFIXPT)
```

For example:

1. Search for an `if` statement similar to the following:

   ```
   !if $(B_ERTSFCN) == 1
   ERT_SFUN     = ..\$(MODEL)_sf.$(MEXEXT)
   ERT_SFUN_SRC = $(MODEL)_sf.c
   MEX          = $(MATLAB_BIN)\mex
   !endif
   ```

   The lines of code in the `if` statement may vary slightly depending on the `make` utility you are using.

2. Add the `LIBFIXPT` and `LIBS` definitions between the `MEX` definition and the `!endif` as follows:

   ```
   !if $(B_ERTSFCN) == 1
   ERT_SFUN     = ..\$(MODEL)_sf.$(MEXEXT)
   ERT_SFUN_SRC = $(MODEL)_sf.c
   MEX          = $(MATLAB_BIN)\mex
   LIBFIXPT =$(MATLAB_ROOT)\extern\lib\win32\microsoft\msvc50\libfixedpoint.lib
   LIBS     = $(LIBS) $(LIBFIXPT)
   !endif
   ```

For more examples, see the supplied Real-Time Workshop TMFs.

Custom Storage Class Compatibility Issues

Prior to 4.0, custom storage classes were implemented with special `Simulink.CustomSignal` and `Simulink.CustomParameter` classes.

In 4.0 and higher, the full functionality of the `Simulink.CustomSignal` and `Simulink.CustomParameter` classes is included in the `Simulink.Signal` and `Simulink.Parameter` classes. Consider replacing `Simulink.CustomSignal` and `Simulink.CustomParameter` objects in your models with equivalent `Simulink.Signal` and `Simulink.Parameter` objects.

If you prefer, you can continue to use the `Simulink.CustomSignal` and `Simulink.CustomParameter` classes in the current release. However, note that the following changes have been implemented in these classes:

- The `Internal` storage class has been removed from the enumerated values of the `RTWInfo.CustomStorageClass` property. `Internal` storage class is no longer supported.
- For the `ExportToFile` and `ImportFromFile` storage classes, the `RTWInfo.CustomAttributes.FileName` and `RTWInfo.CustomAttributes.IncludeDelimeter` properties have been combined into a single property, `RTWInfo.CustomAttributes.HeaderFile`. When specifying a header file, include both the filename and the required delimiter as you want them to appear in generated code, as in the following example:

  ```
  myobj.RTWInfo.CustomAttributes.HeaderFile = '<myheader.h>';
  ```

- Prior to 4.0, you created user-defined CSCs by designing custom packages that included the CSC definitions (as described in the `cscdesignintro` tutorial demo). This technique for creating CSCs is obsolete. For a description of the current procedure, which is much simpler, see “Creating Packages that Support CSC Definitions” in the "Custom Storage Classes" chapter of the Real-Time Workshop Embedded Coder documentation.

  If you designed your own custom packages containing CSCs prior to 4.0, MathWorks strongly recommends that you convert them to 4.0 CSCs. The conversion procedure is described in “Converting Older Packages to Use CSC Registration Files” in the "Custom Storage Classes" chapter of the Real-Time Workshop Embedded Coder documentation.

Defining and Displaying Custom Target Options

For Release 14, extensive improvements and revisions have been made in the appearance and layout of code generation options and other target-specific options for Real-Time Workshop targets. If you have developed a custom

target, you should take advantage of the Model Explorer and Configuration Parameters dialogs to present target options to end users. If you choose not to, a mechanism for using the old-style Simulation Parameters dialog box is available for backwards compatibility.

"Customizing System Target Files" in the Real-Time Workshop documentation discusses compatibility issues and solutions related to the definition and display of target-specific options for custom targets.

- Callback compatibility: If the `rtwoptions` array in your custom system target file contains callbacks, you must convert your callbacks to use the callback compatibility API provided in this release.
- Target options inheritance: If your custom target is derived from another target and inherits options, you need change your system target file to use a new inheritance mechanism.
- Display of target options: Your target options are displayed differently, and you might want to reorganize them.

Supporting Model Referencing in Custom Targets

Existing custom targets require a number of modifications for code generation compatibility with the model reference features introduced in Release 14. See "Supporting Model Referencing" in the Real-Time Workshop documentation for the information you need to adapt your target to support model referencing. Most of the guidelines concern required modifications to the system target file and template makefile.

The list below summarizes general requirements and issues for model reference compatibility that are discussed in "Supporting Model Referencing":

- A model reference compatible target must be derived from the ERT or GRT targets.
- Your system target file must declare model reference compatibility.
- Your template makefile must define a number of makefile tokens, variables and rules specifically for model referencing support.
- To support model reference builds, your template makefile must support use of the shared utilities directory.

- When generating code from a model that references another model, both the top model and the referenced models must be configured for the same code generation target.
- Note that the **External mode** option is not supported in model reference Real-Time Workshop target builds. If the user has selected this option, it is ignored during code generation.

For general information about model referencing, see the *Real-Time Workshop User's Guide*.

Supporting Continuous Time in Custom Targets

As of Release 14, the ERT target supports continuous time. If you want your custom ERT-based target to take advantage of this feature, you must update your template makefile (TMF) and the static main program module (for example, `mytarget_main.c`) for your target.

Template Makefile Modifications. Add the `NCSTATES` token expansion after the `NUMST` token expansion, as follows:

```
NUMST = |>NUMST<|
NCSTATES = |>NCSTATES<|
```

In addition, add `NCSTATES` to the `CPP_REQ_DEFINES` macro, as in the following example:

```
CPP_REQ_DEFINES = -DMODEL=$(MODEL) -DNUMST=$(NUMST) -DNCSTATES=$(NCSTATES) \
-DMAT_FILE=$(MAT_FILE)
-DINTEGER_CODE=$(INTEGER_CODE) \
-DONESTEPFCN=$(ONESTEPFCN) -DTERMFCN=$(TERMFCN) \
-DHAVESTDIO
-DMULTI_INSTANCE_CODE=$(MULTI_INSTANCE_CODE) \
-DADD_MDL_NAME_TO_GLOBALS=$(ADD_MDL_NAME_TO_GLOBALS)
```

Modifications to Main Program Module. The main program module defines a static main function that manages task scheduling for all supported tasking modes of single- and multiple-rate models. `NUMST` (the number of sample times in the model) determines whether the main function calls multirate or single-rate code.

However, when the model has continuous time, it is incorrect to rely on NUMST directly.

When the model has continuous time and the flag TID01EQ is true, both continuous time and the fastest discrete time are treated as one rate in generated code. The code associated with the fastest discrete rate is guarded by a major time step check. When the model has only two rates, and TID01EQ is true, the generated code has a single-rate call interface.

To support models that have continuous time, update the static main module to take TID01EQ into account, as follows:

1. Before NUMST is referenced in the file, add the following code:

```
#if defined(TID01EQ) && TID01EQ == 1 && NCSTATES == 0
#define DISC_NUMST (NUMST - 1)
#else
#define DISC_NUMST NUMST
#endif
```

2. Replace all instances of NUMST in the file by DISC_NUMST.

rtwtypes.h Replaces tmwtypes.h

The ERT target now generates an optimized rtwtypes.h header file, which includes only the necessary definitions required by the target. Most generated code modules require these definitions. This header file replaces the static tmwtypes.h header file. Note that non-ERT targets still use the tmwtypes.h header file.

Updating Customized Static Main Program Modules

If you are upgrading and your application uses a customized version of the static main program module ert_main.c, open the module and make the following changes:

1. Search for RT_MDL. This search brings you to the "Required defines" section.

2. Replace

```
#define RT_MDL                  CONCAT(MODEL,_rt0)
```

with

```
#define RT_MDL                CONCAT(MODEL,_M)
```

3. Search for `tmwtypes.h`. This search brings you to the "Includes" section.

4. Add the following include statement.

```
#include "rtwtypes.h"
```

5. Delete the following include statements.

```
#include "tmwtypes.h"
#include "simstruc_types.h"
```

6. Just below the `#include` section, add the following preprocessor conditional code, which determines whether to set up multitasking mode. Previously, this code resided in `simstruc_types.h`.

```
/*=========================*
 * Setup for multitasking *
 *=========================*/
#if defined(MT)
# if MT == 0
#  undef MT
# else
#  define MULTITASKING 1
# endif
#endif
```

For more information about `ert_main.c`, see “Static Main Program Module” in the Real-Time Workshop Embedded Coder documentation.

The MathWorks recommends that you generate a target-specific main program module rather than use a customized version of the static module, `ert_main.c`. For details, see “Generating a Standalone Program” and “Configuring Templates for Customizing Code Organization and Format” in the Real-Time Workshop Embedded Coder documentation.

Integer Code Only Option Replaced

The **Support floating-point numbers** option replaces, and inverts the logic of, the **Integer code only** option that was supported in previous releases. To generate pure integer code in new models, deselect the **Support floating-point numbers** option.

Note that for compatibility, models that were configured for **Integer code only** prior to Release 14 are automatically configured with **Support floating-point numbers** deselected, and generate pure integer code.

Rate Grouping Compatibility Issues

To take full advantage of the efficiency of rate grouping:

- Your multirate inlined S-functions must be upgraded to be fully rate grouping compliant. Existing S-functions continue to operate correctly without change, but we strongly recommend that you upgrade your TLC S-function implementations. See “Rate Grouping Compliance and Compatibility Issues” in the "Data Structures and Program Execution" chapter of the Real-Time Workshop Embedded Coder documentation.
- If you have previously generated and modified `ert_main.c` (as is typical of many ERT-based custom targets) take care to preserve your modifications and make equivalent changes to the regenerated `ert_main.c`. After you have done so, set the TLC variable `RateBasedStepFcn` to `1`, as described in “Rate Grouping and the Static Main Program” in the "Data Structures and Program Execution" chapter of the Real-Time Workshop Embedded Coder documentation.

Real-Time Object Structure Obsoleted by Real-Time Model Structure

In MATLAB Release 13, the real-time model (*`model`*`_M`) data structure replaced the real-time object (*`model`*`_rtO`) data structure. However, use of use of the older structure was still supported for backward compatibility.

Real-Time Workshop Embedded Coder 4.0 requires use of the real-time model data structure. If you have developed a custom target that references *`model`*`_rtO` (for example, in a customized`ert_main.c` module) you must replace them with references to *`model`*`_M`.

See the "Developing Models for Code Generation" chapter of the Real-Time Workshop Embedded Coder documentation for further information about the real-time model data structure.

rtmIsSampleHit and rtmIsSpecialSampleHit Macros Obsolete

The following macros are now obsolete and should not be used with the ERT target:

- `rtmIsSampleHit`
- `rtmIsSpecialSampleHit`

This does not cause a problem unless you have coded these macros directly into your TLC files. The recommended practice is to use the following TLC library functions:

- `%<LibIsSFcnSampleHit(tid)>`
- `%<LibIsSFcnSpecialSampleHit(tid)>`

If you have used these functions, they operate transparently.

RTWInfo Properties Assignment Warning Message

This note describes a minor change in behavior when the `RTWInfo` properties of a data object are assigned incorrectly.

You can assign a custom storage class to a data object either by using Simulink Model Explorer, or by setting the `RTWInfo` properties via MATLAB commands. (See also the "Creating and Using Custom Storage Classes" chapter in the Real-Time Workshop Embedded Coder documentation.) If you use MATLAB commands to assign a custom storage class, you must set both the `RTWInfo.CustomStorageClass` and `RTWInfo.StorageClass` fields. Make sure that the `RTWInfo.StorageClass` property is set to `'Custom'`, as in the following example.

```
aa = Simulink.Signal;
aa.RTWInfo.StorageClass = 'Custom';
aa.RTWInfo.CustomStorageClass = 'Struct';
aa.RTWInfo.CustomAttributes.StructName = 'mySignals';
```

If the `RTWInfo.StorageClass` is not set correctly as shown above, the assigned custom storage class (`RTWInfo.CustomStorageClass`) are ignored during code generation. In such cases, a warning is displayed at the time `RTWInfo.CustomStorageClass` is assigned, for example

```
foo = Simulink.Signal
foo.RTWInfo.CustomStorageClass = 'Struct'

Warning: The 'CustomStorageClass' property of RTWInfo will have
no effect unless the 'StorageClass' property is set to 'Custom'.
```

Previously, the warning was displayed at the time `RTWInfo.StorageClass` was assigned.

Generating R13SP1+ or R13SP2 Code From ERT-Based Simulink Models Created In R14 or Later

Due to a design change made in V4.0 (R14) Real-Time Workshop Embedded Coder, a Real-Time Workshop error occurs when generating R13SP1+ or R13SP2 code from an ERT-based Simulink model created in R14 or later.

If you use R14 or later to save an ERT-based Simulink model into R13SP1 format (using Simulink **File > Save As > Save as type**), and then try to generate code for the model under R13SP1+ or R13SP2, the following error is displayed:

```
Error executing build command: Error using ==> make_rtw
Error using ==> tlc_c
Error using ==> tlc_c (InvokeTLC)
Error:  Real-Time Workshop Error: Unable to locate ERT header file banner template:
ert_code_template.cgt.
```

To work around this problem in R13SP1+ or R13SP2, download and install a replacement version of the file *`matlabroot`*`/rtw/c/tlc/mw/setuplib.tlc`, as follows:

1. Go to the directory *`matlabroot`*`/rtw/c/tlc/mw` and rename the file `setuplib.tlc` to `setuplib.tlc.old`.
2. Click the appropriate link below to download a replacement version of `setuplib.tlc`. Place the file in the same directory as the file you renamed.

- `setuplib_sp2.tlc`
- `setuplib_sp1plus.tlc`

Note If you are not logged in to your MathWorks Account when you click the link, you will prompted to log in or create an account.

If you are reading the PDF or hard copy documentation, go to the MATLAB Help browser or The MathWorks web documentation to use the link.

---

3 Rename the downloaded file to `setuplib.tlc`.

Version 3.2.1 (R13SP2) Real-Time Workshop Embedded Coder Software

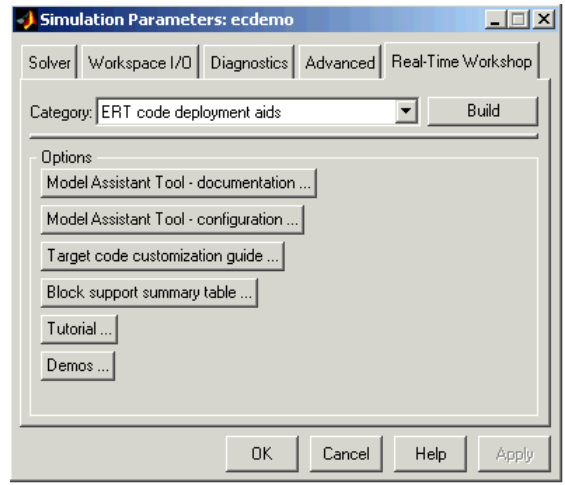
This table summarizes what's new in Version 3.2.1 (R13SP2):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Fixed bugs	V3.2.1 product documentation

New features and changes introduced in this version are described here:

ERT Code Deployment Aids Added to GUI

A new group of buttons has been added to the Embedded Real-Time (ERT) target options in the **Real-Time Workshop** pane of the Simulation Parameters dialog box. To access these buttons, select `ERT code deployment aids` from **Category** menu, as shown in the figure below.



The `ERT code deployment aids` buttons provide quick access to features and information that can help you to optimize your generated code. The buttons are:

- **Model Assistant Tool - documentation**: Click this button to view online help for the Model Assistant Tool in the MATLAB Help browser. You can also view this help by typing the MATLAB command

  ```
  modelassistant('help')
  ```

- **Model Assistant Tool - configuration**: Click this button to open the Model Assistant Tool for configuration of options.
- **Target code customization guide**: Click this button to view the "Optimizing Generated Code" chapter of the Real-Time Workshop Embedded Coder documentation. The chapter documents useful code generation, optimization, and customization techniques for the ERT target. Most of the features described were introduced in Real-Time Workshop Embedded Coder 3.2 (see the release notes for"Version 3.2 (R13SP1+) Real-Time Workshop® Embedded Coder Software" on page 155 for a summary).
- **Block summary support table**: Click this button to view the Simulink Block Data Type Support Table in the MATLAB Help Browser. The table describes the data types that are supported by the blocks in the main Simulink and Fixed-Point libraries. The table also identifies blocks that are suitable for production code generation. You can also view the table by typing the MATLAB command

  ```
  showblockdatatypetable
  ```

- **Tutorial**: Click this button to open an interactive Real-Time Workshop Embedded Coder tutorial demo in the in the MATLAB Help Browser. You can also view the tutorial demo by typing the MATLAB command

  ```
  ecodertutorial
  ```

- **Demos**: Click this button to open the Real-Time Workshop Embedded Coder demo suite. You can also view the demos by typing the MATLAB command

  ```
  ecoderdemos
  ```

Version 3.2 (R13SP1+) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 3.2 (R13SP1+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	No bug fixes	No

New features and changes introduced in this version are:

Advanced Code Generation Techniques Documented

A new chapter, "Optimizing Generated Code", has been added to the Real-Time Workshop Embedded Coder User's Guide. This chapter contains complete information on the new features that are summarized in these release notes. In addition, the chapter documents useful code generation, optimization, and customization techniques that have not received wide exposure in previous releases. These include

- How to specify target characteristics (such as word sizes for C data types) for the build process, so that generated code is correct for deployment on target hardware

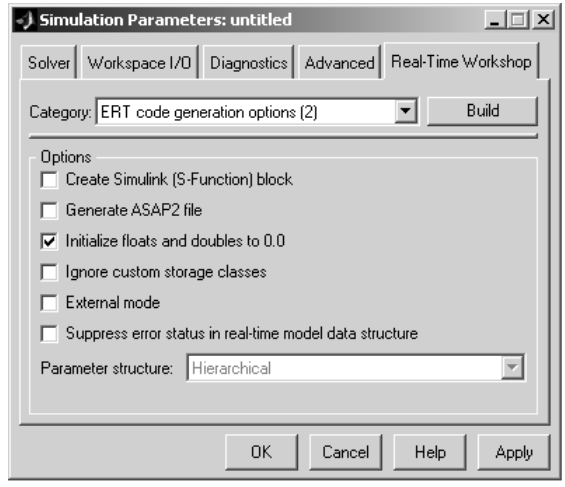
- A general hook file mechanism for adding target-specific customizations to the build process

New Code Generation Options

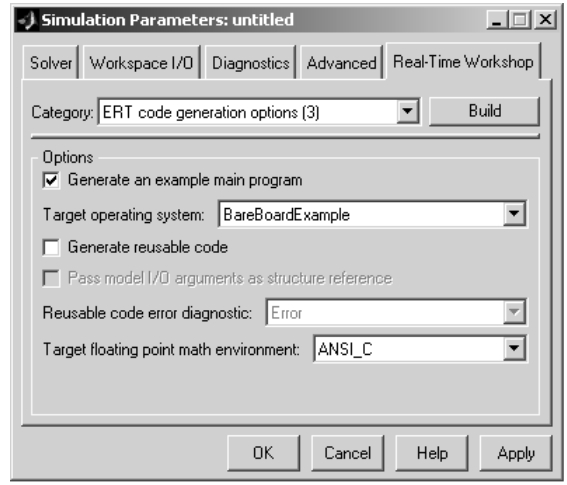
Several new code generation options have been added, and some changes have been made to the layout of Embedded Real-Time (ERT) target code generation options in the **Real-Time Workshop** pane of the Simulation Parameters dialog box.

Options Layout Changes and Additions

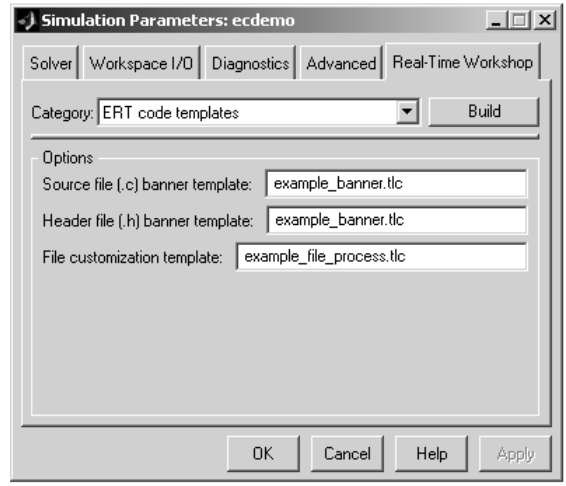
The **Suppress error status in real-time model data structure** option has been relocated to the `ERT code generation options (2)` category, as shown in this figure.



A new code generation option, **Pass model I/O arguments as structure reference**, is now available in the `ERT code generation options (3)` category, as shown below. This option is described in "Passing Model I/O Arguments to the model_step Function" on page 159.



A new group of options supporting use of *code templates*, a powerful and simple technique for customizing generated code, has been added. These options are available in the `ERT code templates` category of the **Real-Time Workshop** pane of the Simulation Parameters dialog box (see the figure below). Code templates are summarized in "Code Templates for Customizing Generated Code" on page 158.



Auto-Configuration of Models for Code Generation

Real-Time Workshop Embedded Coder now supports automated configuration of all (or selected) model parameters during the code generation process. By automatically configuring a model in this way, you can avoid manually configuring models. This saves time and eliminates potential errors.

Auto-configuration is performed by executing a `.m` file (referred to as a *hook file*) that is executed as part of the target build process. Therefore, auto-configuration becomes a function of the target that invokes the hook file. You can direct the automatic configuration process to save existing model settings before code generation and restore them afterwards, so that options the user chooses manually are not disturbed.

Optimized ERT Targets for Fixed-Point and Floating-Point Code Generation

To make it easier for you to customize a hook file that is optimized for your target hardware, Real-Time Workshop Embedded Coder provides two variants of the ERT target:

- `RTW Embedded Coder (auto configures for optimized fixed-point code)`: To optimize for fixed-point code generation, select this target from the System Target File Browser.
- `RTW Embedded Coder (auto configures for optimized floating-point code)`: To optimize for floating-point code generation, select this target from the System Target File Browser.

Code Templates for Customizing Generated Code

The ERT target now supports use of *custom file processing templates* (CFP templates).

A CFP template is a Target Language Compiler (TLC) file that calls a high-level applications programming interface (API), referred to as the *code template* API. The code template API simplifies generation of custom source code by letting you

- Generate virtually any type of source (`.c`) or header (`.h`) file. A CFP template can emit code to the standard generated model files (e.g., `model.c`, `model.h`, etc.) or generate files that are independent of model code.
- Organize generated code into sections (such as includes, `typedefs`, functions, and more). Your CFP template can emit code (e.g., functions), directives (such as `#define` or `#include` statements), or comments into each section as required.
- Generate code to call model functions such as *`model`*`_initialize`, *`model`*`_step`, etc.
- Generate code to read and write model inputs and outputs.
- Generate a main program module.
- Obtain information about the model and the files being generated from it.

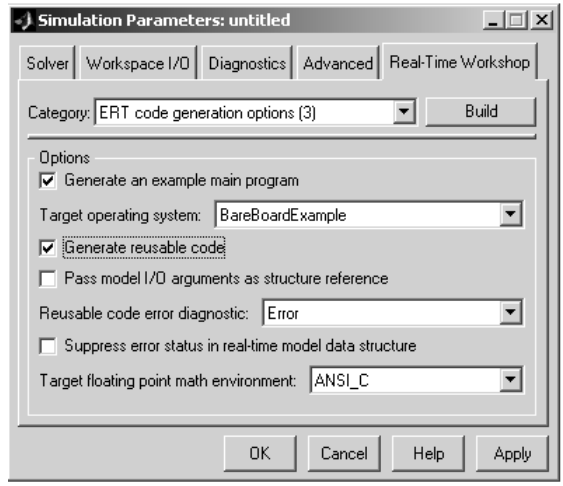
CFP templates are described in the "Optimizing Generated Code" chapter of the Real-Time Workshop Embedded Coder User's Guide.

Custom File Banner Generation

The ERT target now supports use of *banner templates* during code generation. A banner template is a TLC file that specifies banner and trailer comments that are emitted to generated source (`.c`) and header (`.h`) files. Banner templates are described in the "Optimizing Generated Code" chapter of the Real-Time Workshop Embedded Coder User's Guide.

Passing Model I/O Arguments to the model_step Function

A new code generation option, **Pass model I/O arguments as structure reference**, lets you control how model inputs and outputs at the root level of the model are passed in to the *`model`*`_step` function. This option is available in the `ERT code generation options (3)` category of the **Real-Time Workshop** pane of the Simulation Parameters dialog box. When **Generate reusable code** is selected, **Pass model I/O arguments as structure reference** is enabled, as shown in this figure.



When **Pass model I/O arguments as structure reference** is deselected (the default), each root-level model input and output is passed to *`model`*`_step` as a separate argument. When this option is selected, all root-level inputs are packed into a `struct` that is passed to *`model`*`_step` as an argument. Likewise, all root-level outputs are packed into a `struct` that is also passed to *`model`*`_step` as an argument. Selecting **Pass model I/O arguments as structure reference** can reduce the number of arguments passed in to *`model`*`_step`.

See the "Preparing Models for Code Generation" chapter of the Real-Time Workshop Embedded Coder User's Guide for further details.

Version 3.1 (R13SP1) Real-Time Workshop Embedded Coder Software

This table summarizes what's new in Version 3.1 (R13SP1):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	No bug fixes	No

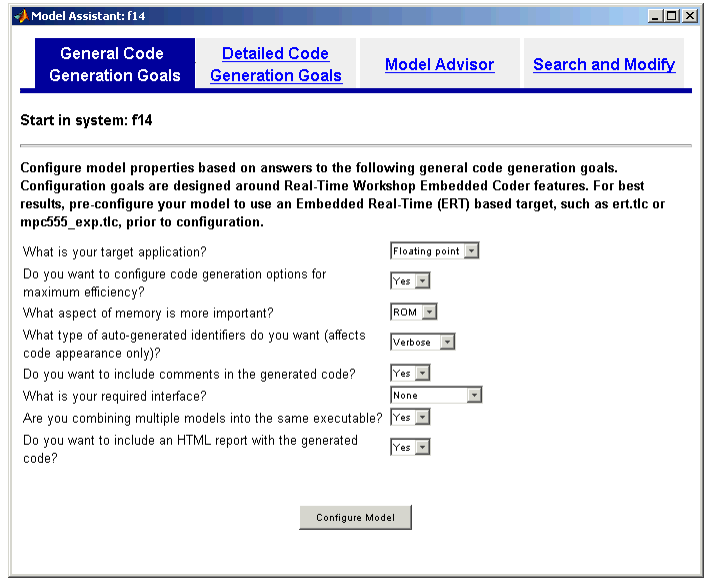
New features and changes introduced in this version are described here:

Model Assistant Tool

The Model Assistant Tool is a utility that lets you configure a model for code generation quickly. The Model Assistant Tool also helps you to identify aspects of your model that impede production deployment or limit code efficiency. You can use the Model Assistant Tool at any point in your design cycle, as it is completely independent from the code generation process.

The Model Assistant Tool is designed primarily for use with Real-Time Workshop Embedded Coder. It works most effectively with the Embedded Real-Time (ERT) target and with ERT-based targets (such as the Embedded Target for Motorola® MPC555). It will also operate with other targets.

The figure below shows the top-level window of the Model Assistant Tool.



Four main components of the Model Assistant Tool provide a powerful and centralized interface for configuring settings for Simulink blocks, Stateflow charts, models and subsystems. You select these components via the four buttons at the top of the Model Assistant display:

- **General Code Generation Goals**
- **Detailed Code Generation Goals**
- **Model Advisor**
- **Search and Modify**

These components are summarized in the next sections.

General Code Generation Goals

This component lets you quickly configure code generation settings based on specific goals, such as whether to optimize for RAM or ROM usage. Once you have decided the overall optimization and tradeoffs for your application, the Model Assistant Tool will select the model settings that best suit your goals.

Detailed Code Generation Goals

This component presents a centralized interface to the available code generation options. Options are grouped by category, and are applied across products.

Model Advisor

The Model Advisor component is particularly useful early in the design cycle. It provides an analysis of your model to ensure that you best utilize Real-Time Workshop Embedded Coder. You can check selected aspects of your model settings (for example, to identify possible inefficiencies such as blocks that generate saturation and rounding code) or choose **Select All** for a comprehensive analysis.

Search and Modify

This component is a powerful model search and modify engine. It reduces the effort of configuring a model block by block. The search feature helps you find attributes of blocks, lines, input ports, output ports, and annotations quickly. The modify feature lets you perform rapid batch operations on the search results. Frequently performed tasks are packaged conveniently into a single button click.

The **Search and Modify** component includes the following features:

- The **Frequent tasks** page lets you quickly perform common actions.
- The **Simulink object search** page lets you specify a general Simulink object search and modify action. This search mechanism is useful when you know the specific names of underlying attributes.
- The **Stateflow object search** page lets you quickly configure the Stateflow data in your model. This is particularly useful for converting data from floating point to fixed-point types.
- The **Search and replace Simulink text** page lets you quickly modify text for objects in Simulink. For example, you can change all occurrences of `'K1'` to `'K2'`. The semantics of the search and replace are the same as for the Stateflow search and replace tool that ships with Stateflow.
- Two **Parameter name search** mechanisms are provided:

- Search and modify parameters using prompt strings. This search mechanism is useful when you know the parameter by its dialog prompt string, but you don't know the name of the underlying attribute.
- "Fuzzy" search using property and/or value pairs. This search mechanism is useful for isolating the name of an underlying attribute.

Using the Model Assistant Tool

You run Model Assistant Tool from the MATLAB command line, via the `modelassistant` command. Before invoking the Model Assistant Tool, make sure that the desired target (such as the ERT target) is selected in the **Target Configuration** section of the **Real-Time Workshop** pane of the Simulation Parameters dialog box.

The following examples illustrate the `modelassistant` command syntax and its possible arguments.

To obtain detailed help on the Model Assistant Tool, type

```
modelassistant('help')
```

To invoke the Model Assistant Tool for the root system of a model, type

```
modelassistant('model')
```

where *`model`* is the name of the model.

To invoke the Model Assistant Tool for a particular subsystem in a model, type

```
modelassistant('subsystem')
```

where *`subsystem`* is the name of the subsystem.

You can also invoke the Model Assistant Tool for models and systems using the built-in Simulink `bdroot`, `gcb`, and `gcs` commands. For example:

```
modelassistant(gcs)
```

Further Help and Demos

The above sections have summarized the main features of the Model Assistant Tool. To obtain full online documentation on the Model Assistant Tool, type

```
modelassistant('help')
```

There are also three demo models available for the Model Assistant Tool: `advisordemo1`, `advisordemo2`, and `advisordemo3`.

Compatibility Summary for Real-Time Workshop Embedded Coder Software

This table summarizes new features and changes that might cause incompatibilities when you upgrade from an earlier version, or when you use files on multiple versions. Details are provided in the description of the new feature or change.

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version **V5.5 (R2010a)**	See the **Compatibility Considerations** subheading for each of these new features or changes: • "AUTOSAR Compiler Abstraction Macro Generation" on page 6 • "Software-in-the-Loop Simulation Mode" on page 7 • "Efficiency Objective for Code Generation Separated into Execution, ROM, and RAM" on page 8 • "Obsolete Header File rtw/c/ert/ertformat.h Removed" on page 14
V5.4 (R2009b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Control Visibility and Access for Data Members In C++ Model Class Using New Configuration Parameters" on page 23 • "Function Subsystems and Charts Now Generated as Private Member Functions in C++ Encapsulation Model Class" on page 24 • "Mismatched System Code Options for Nested Subsystems or Charts Now Error

Version (Release)	New Features and Changes with Version Compatibility Impact
	Out During C++ Encapsulation Code Generation” on page 25
V5.3 (R2009a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • “Optimized ERT-Based Targets Deprecated” on page 40 • “Right-Click Builds No Longer Generate Unused Functions in Production Code” on page 40
V5.2 (R2008b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • “Scheduling Code Now Separated from Model Application Code” on page 51 • “Model Initialization Function Prototype Control” on page 53
V5.1.1 (R2008a+)	None
V5.1 (R2008a)	See the **Compatibility Considerations** subheading for this new feature or change: • “Function Prototype Control Enhancements” on page 62 • “Custom Storage Class File Type Changed” on page 62
V5.0.1 (R2007b+)	None
V5.0 (R2007b)	None
V4.6.1 (R2007a+)	None
V4.6 (R2007a)	None

Version (Release)	New Features and Changes with Version Compatibility Impact
V4.5 (R2006b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Maximum Length Enforced for Auto-Generated Identifiers in Generated Code" on page 93 • "New Default Value for IncludeERTFirstTime Model Configuration Parameter" on page 95
V4.4.1 (R2006a+)	None
V4.4 (R2006a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Identifier Format Control Parameters for Code Generation" on page 101 • "New sl_customization API for Customizing Data Objects" on page 106
V4.3 (R14SP3)	See the **Compatibility Considerations** subheading for this new feature or change: • "ERT Automatic Configuration Changes" on page 115
V4.2.1 (R14SP2+)	None
V4.2 (R14SP2)	See the **Compatibility Considerations** subheading for this new feature or change: • "File custom_user_type_registration.m No Longer Automatically Called During Code Generation" on page 121
V4.1 (R14SP1)	None

Version (Release)	New Features and Changes with Version Compatibility Impact
V4.0 (R14)	See: • "Upgrading from R13SP1+ or R13SP2" on page 142 • "Generating R13SP1+ or R13SP2 Code From ERT-Based Simulink Models Created In R14 or Later" on page 151
V3.2.1 (R13SP2)	None
V3.2 (R13SP1+)	None
V3.1 (R13SP1)	None