Real-Time Workshop® Embedded Coder Release Notes

The Real-Time Workshop Embedded Coder 4.2.1 Release Notes describe the changes introduced in the Release 14 with Service Pack 2+ version of the Real-Time Workshop Embedded Coder. The following topics are discussed in these Release Notes:

The Real-Time Workshop Embedded Coder Release Notes also provide information about recent versions of the product, in case you are upgrading from an earlier version:

Contents

1 Real-Time Workshop Embedded Coder 4.2.1 Release Notes

2 Real-Time Workshop Embedded Coder 4.2 Release Notes

Real-Time Workshop Embedded Coder 4.1 Release Notes

3

Real-Time Workshop Embedded Coder 4.0 Release Notes

4

5 Real-Time Workshop Embedded Coder 3.2.1 Release Notes

6 Real-Time Workshop Embedded Coder 3.2 Release Notes

7 Real-Time Workshop Embedded Coder 3.1 Release Notes

1

Real-Time Workshop Embedded Coder 4.2.1 Release Notes

Major Bug Fixes

Real-Time Workshop Embedded Coder Version 4.2.1 includes important bug fixes made since Version 4.2. You can see a list of the major Version 4.2.1 bug fixes on the MathWorks Web site.

If you are viewing these Release Notes in PDF form, click the words "bug fixes" in the second sentence above to see the notes about major fixes.

Known Software and Documentation Problems

The MathWorks Web site includes a list of known software and documentation problems in Version 4.2.1.

If you are viewing these Release Notes in PDF form, click the word "problems" in the sentence above to see the notes about known problems.

2

Real-Time Workshop Embedded Coder 4.2 Release Notes

New Features and Enhancements

Real-Time Workshop Embedded Coder Version 4.2 introduces the following new features and enhancements.

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 and limitations, see the "Real-Time Workshop 6.2 Release Notes".

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 "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 Requirements to Configuration Options" 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 update/output 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** dialog box allowed you to select the **Alias override naming rule** check box for an `mpt` data object. As explained in the "Real-Time Workshop Embedded Coder Module Packaging Features" documentation, 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 the Real-Time Workshop Embedded Coder Module Packaging Features documentation. The revised material explains how to create Simulink data objects using the Data Object Wizard, and compares this with creating `mpt` data objects.

Major Bug Fixes

Real-Time Workshop Embedded Coder Version 4.2 includes important bug fixes made since Version 4.1. You can see a list of the major Version 4.2 bug fixes on the MathWorks Web site.

If you are viewing these Release Notes in PDF form, click the words "bug fixes" in the second sentence above to see the notes about major fixes.

3

Real-Time Workshop Embedded Coder 4.1 Release Notes

Major Bug Fixes

The Real-Time Workshop Embedded Coder 4.1 includes several bug fixes made since Version 4.0. This section describes the particularly important Version 4.1 bug fixes.

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

Upgrading from Version 4.0

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 the Module Packaging Features documentation
- Addition of data placement rules in the Appendix of the Module Packaging Features documentation

Upgrading from Version 3.2.1 or 3.2

This section discusses the following issues pertaining to upgrades from Real-Time Workshop Embedded Coder Version 3.2.1 or 3.2 to Version 4.1:

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 with 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, The MathWorks strongly recommends that you convert them to 4.0 CSCs. The conversion procedure is described in "Converting pre-Release 14 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 is available for backwards compatibility.

The "System Target Files" chapter of the Developing Embedded Targets document 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. See "Compatibility Issues for rtwoptions Callbacks" in the "System Target Files" chapter of the Developing Embedded Targets document.
- 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. See "Inheritance of Target Options" in the "System Target Files" chapter of the Developing Embedded Targets document.
- Display of target options: Your target options are displayed differently, and you might want to reorganize them. See "Appearance of Target Options in New Model Explorer Dialog View" in the "System Target Files" chapter of the Developing Embedded Targets document for information on how custom target options are displayed.

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. The "Supporting Model Referencing" chapter of the Developing Embedded Targets document provides 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 the "Supporting Model Referencing" chapter:

- 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-level 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 documentation.

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 singlerate code.

However, when the model 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 “The Static Main Program Module”.

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 the Main Program” and “Custom File Processing” 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 multi-rate 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 "Data Structures and Program Execution" 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 the Simulink Model Explorer, or by setting the `RTWInfo` properties via MATLAB commands. (See also the "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.

Known Software and Documentation Problems

Real-Time Workshop Embedded Coder 4.1 provides a list of known software and documentation problems. If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site, and use the “problems” link provided.

4

Real-Time Workshop Embedded Coder 4.0 Release Notes

New Features

This section summarizes the new features and enhancements introduced in the Real-Time Workshop Embedded Coder 4.0. The new features include:

If you are upgrading from a version prior to Version 4.0, then also see "New Features" on page 6-2 in the Real-Time Workshop Embedded Coder 3.2 Release Notes.

Expanded Documentation Collection

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

Using Real-Time Workshop Embedded Coder	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 Workship 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. 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 the Model Explorer and the **Configuration Parameters** dialog. See Using Simulink and the Simulink 6.0 Release Notes for details.
- The general Real-Time Workshop code generation options and use of the System Target File Browser. See the Real-Time Workshop documentation and the “Real-Time Workshop 6.0 Release Notes” for details.

Some panes of the new **Configuration Parameters** dialog (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 3-6 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 supress 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.
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.

Pane and Subpane	Option	Usage
	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 M-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.
	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.

Pane and Subpane	Option	Usage
	#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.
	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.

Pane and Subpane	Option	Usage
	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, typedefs, functions, and so on.
	Generate an example main program	Control whether `ert_main.c` is generated.
	Target operating system	Generate a bare-board 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.

Pane and Subpane	Option	Usage
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.
	Source of initial values	Specify the source that initializes the model's signals in the generated code.

Pane and Subpane	Option	Usage
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.
	Use memset to intitialize 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.

Pane and Subpane	Option	Usage
	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 "Code Generation Options and Optimizations" chapter of the Real-Time Workshop Embedded Coder documentation.
- The Module Packaging Features documentation.

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 “Real-Time Workshop 6.0 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, 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. 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 3-8.

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.

Note that inlining S-functions is often advantageous in production code generation, for example in implementing device drivers. See "Tradeoffs in Device Driver Development" in the Developing Embedded Targets document for a discussion of the pros and cons.

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, multi-model control system application.s

The Module Packaging Features documentation 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 and M-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 a selected data object's property values as a comment in a generated file above that data object's identifier.
  - 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 ASAP2 Files" 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 "Advanced Code Generation Techniques" 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 "Custom Storage Classes" chapter of the Real-Time Workshop Embedded Coder User's Guide for a complete 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 3-5 for information on these changes.

More Efficient Multi-Rate Multitasking Code Generation

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

For multi-rate 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 sub-rate 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 multi-rate multitasking models, you must regenerate code, including the main program, `ert_main.c`.

See the "Data Structures and Program Execution" 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 multi-rate, 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 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 below, are fully described in the "System Target Files" chapter of Developing Embedded Targets.

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 or the Model Explorer. 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 the "Supporting Model Referencing" chapter of Developing Embedded Targets.

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
```

Major Bug Fixes

The Real-Time Workshop Embedded Coder 4.0 includes several bug fixes made since Version 3.2.1. This section describes the particularly important Version 4.0 bug fixes.

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

5

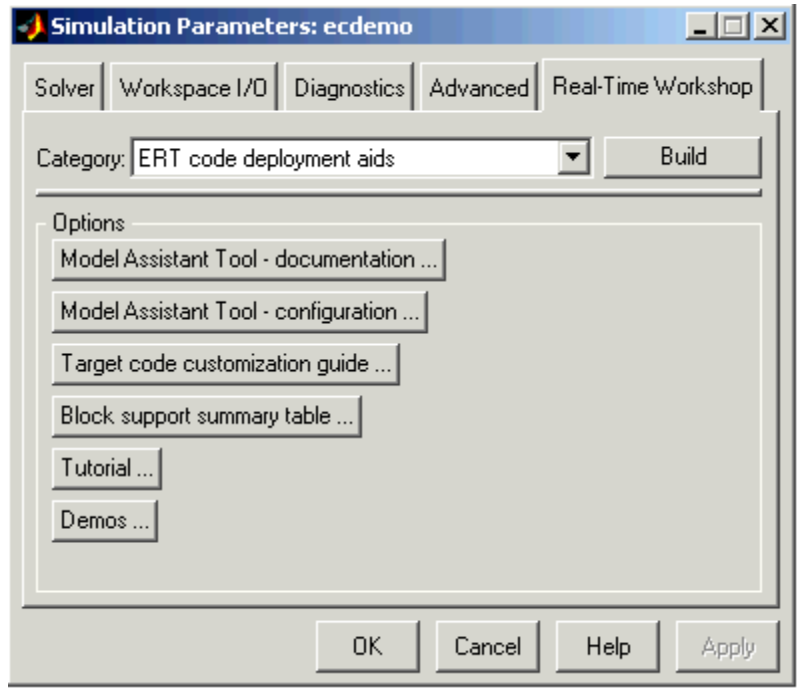
Real-Time Workshop Embedded Coder 3.2.1 Release Notes

New Features

This section summarizes the new features and enhancements introduced in the Real-Time Workshop Embedded Coder 3.2.1.

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 "Advanced Code Generation Features" 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 the Real-Time Workshop Embedded Coder 3.2 (see Chapter 6, “Real-Time Workshop Embedded Coder 3.2 Release Notes” 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
  ```

Major Bug Fixes

Real-Time Workshop Embedded Coder 3.2.1 includes important bug fixes made since Version 3.2.

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

Known Software and Documentation Problems

This section includes a link to a description of known software and documentation problems in Version 3.2.1 and prior.

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

6

Real-Time Workshop Embedded Coder 3.2 Release Notes

New Features

This section summarizes the new features and enhancements introduced in the Real-Time Workshop Embedded Coder 3.2. The new features include:

Advanced Code Generation Techniques Documented

A new chapter, "Advanced Code Generation Techniques," 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

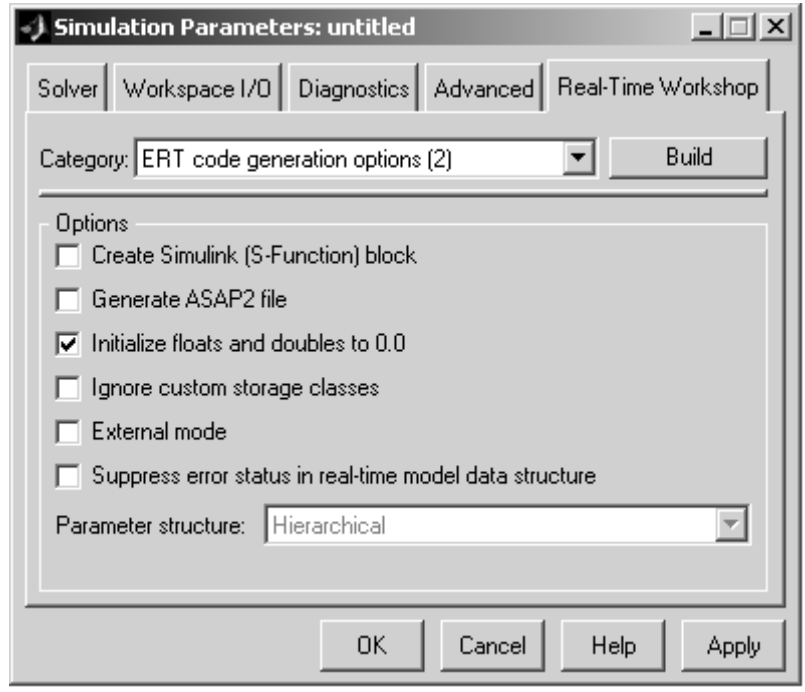
- How to specify target characteristics (such as word sizes for C data types) for the build process, so that generated code is correct for deployment on target hardware
- A general hook file mechanism for adding target-specific customizations to the build process

New Code Generation Options

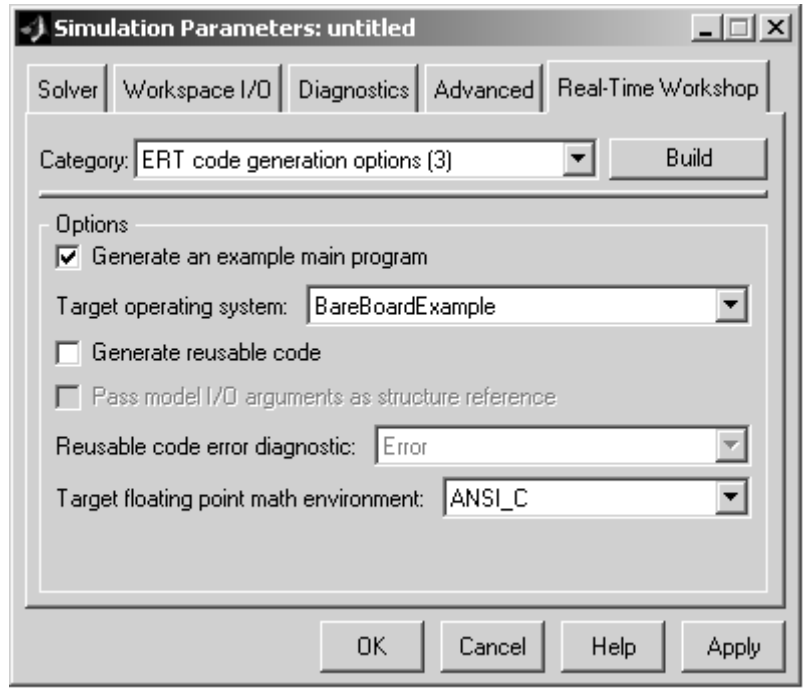
Several new code generation options have been added, and some changes have been made to the layout of Embedded Real-Time (ERT) target code generation options in the **Real-Time Workshop** pane of the **Simulation Parameters** dialog box.

Options Layout Changes and Additions

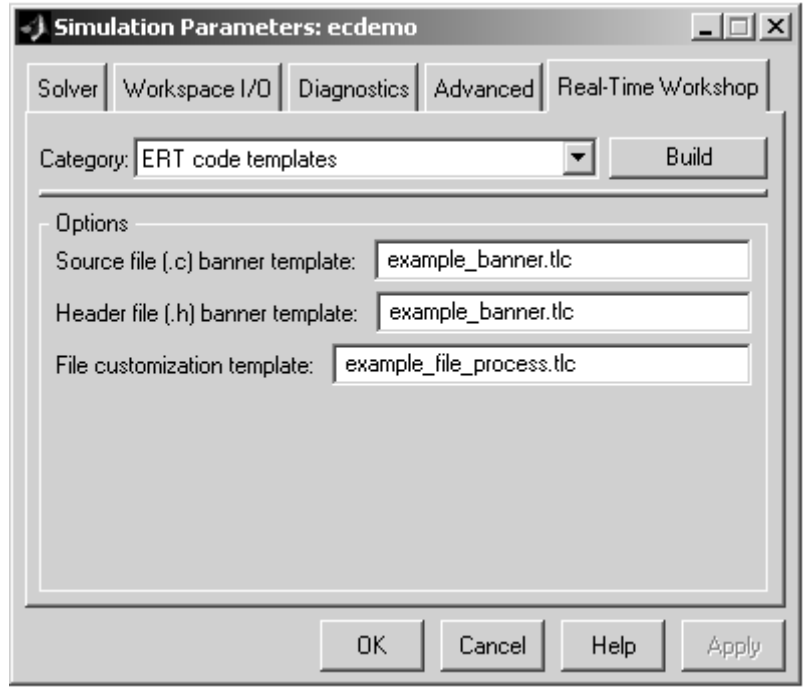
The **Suppress error status in real-time model data structure** option has been relocated to the `ERT code generation options (2)` category, as shown in this figure.



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 6-7.



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 (see the figure below). Code templates are summarized in "Code Templates for Customizing Generated Code" on page 6-6.



Auto-Configuration of Models for Code Generation

The 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 an 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 the user's manually chosen options are not disturbed.

The automatic configuration process, and utilities provided to support auto-configuration, are described in the "Advanced Code Generation Features" chapter of the Real-Time Workshop Embedded Coder User's Guide.

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.

The use of these targets is detailed in the "Advanced Code Generation Features" chapter of the Real-Time Workshop Embedded Coder User's Guide.

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.

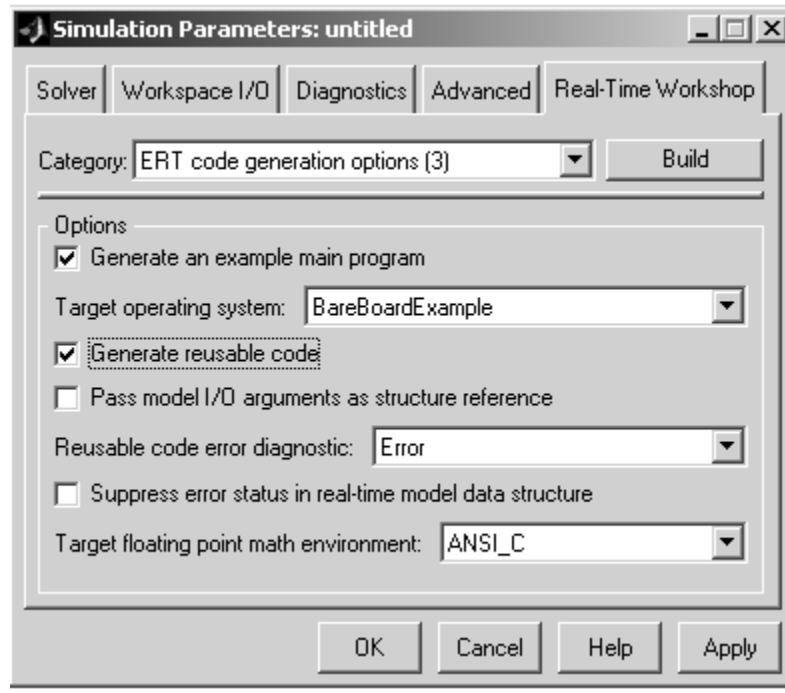
CFP templates are described in the "Advanced Code Generation Features" chapter of the Real-Time Workshop Embedded Coder User's Guide.

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 "Advanced Code Generation Features" 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 "Code Generation Options and Optimizations" chapter of the Real-Time Workshop Embedded Coder User's Guide documentation for further details.

7

Real-Time Workshop Embedded Coder 3.1 Release Notes

New Features

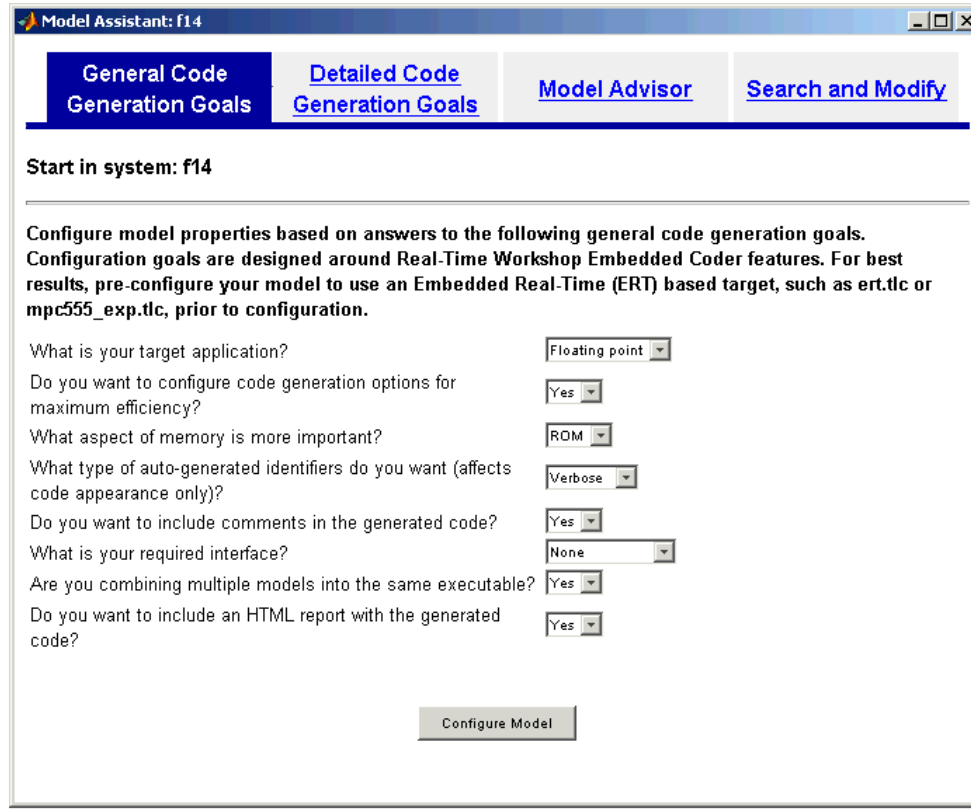
This section summarizes the new features and enhancements introduced in the Real-Time Workshop Embedded Coder 3.1.

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 trade-offs 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 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_name')
```

where `model_name` is the name of the model.

To invoke the Model Assistant Tool for a particular system in a model, type

```
modelassistant('system_name')
```

where `system_name` is the name of the system.

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`.