

Real-Time Workshop Embedded Coder Release Notes

The “Real-Time Workshop Embedded Coder 3.0 Release Notes” on page 1-1 describe the changes introduced in the latest version of the Real-Time Workshop Embedded Coder. The following topics are discussed in these Release Notes:

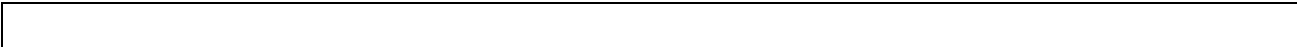
- “New Features” on page 1-2
- “Major Bug Fixes” on page 1-13
- “Known Software and Documentation Problems” on page 1-14
- “Upgrading from an Earlier Release” on page 1-15

If you are upgrading from a release earlier than Release 12.1, you should also see these sections:

- “Real-Time Workshop Embedded Coder 2.0 Release Notes” on page 2-1
- “Real-Time Workshop Embedded Coder 1.0 Release Notes” on page 3-1

Printing the Release Notes

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



Real-Time Workshop Embedded Coder 3.0 Release Notes

1

New Features	1-2
New User's Guide	1-2
Auto-generated Main Program	1-2
Code Generation Options	1-4
ECRobot Target Example	1-6
External Mode Support	1-7
GetSet Custom Storage Class for Data Store Memory	1-8
Hierarchical Parameter Structures	1-8
Real-Time Model Structure Replaces Real-Time Object and Logging Object	1-9
Reusable Code Generation	1-9
Revised Packaging of Generated Code Files	1-10
Template Makefile for Tornado	1-12
Major Bug Fixes	1-13
Fixed Incorrect Effects of Expression Folding for Action Subsystems	1-13
Removed License Restriction on Loading Objects with Custom Storage Classes	1-13
Known Software and Documentation Problems	1-14
Considerations for Use of Initialize Internal Data Option	1-14
Upgrading from an Earlier Release	1-15
Changes to Main Program Module (ert_main.c)	1-15
HTML Code Generation Report Changes	1-15
Include Model Name in Exported Structures Option Superseded	1-15
Replace Obsolete Header File #includes	1-16
Update Your Custom System Target Files	1-16

2

Real-Time Workshop Embedded Coder 2.0 Release Notes

Release Summary	2-2
New Features	2-3
Custom Storage Classes for Data Objects	2-3
Enhanced Code Generation Options	2-4
Virtualized Output Ports Optimization	2-4
Improved HTML Code Generation Report	2-4

3

Real-Time Workshop Embedded Coder 1.0 Release Notes

Introduction to the Real-Time Workshop Embedded Coder	3-2
Real-Time Workshop Embedded Coder Demos	3-3

Real-Time Workshop Embedded Coder

3.0 Release Notes

New Features	1-2
New User's Guide	1-2
Auto-generated Main Program	1-2
Code Generation Options	1-4
ECRobot Target Example	1-6
External Mode Support	1-7
GetSet Custom Storage Class for Data Store Memory	1-8
Hierarchical Parameter Structures	1-8
Real-Time Model Structure Replaces	
Real-Time Object and Logging Object	1-9
Reusable Code Generation	1-9
Revised Packaging of Generated Code Files	1-10
Template Makefile for Tornado	1-12
Major Bug Fixes	1-13
Fixed Incorrect Effects of Expression Folding for Action	
Subsystems	1-13
Removed License Restriction on Loading Objects with Custom Storage	
Classes	1-13
Known Software and Documentation Problems	1-14
Considerations for Use of Initialize Internal	
Data Option	1-14
Upgrading from an Earlier Release	1-15
Changes to Main Program Module (ert_main.c)	1-15
HTML Code Generation Report Changes	1-15
Include Model Name in Exported Structures	
Option Superseded	1-15
Replace Obsolete Header File #includes	1-16
Update Your Custom System Target Files	1-16

New Features

This section summarizes the new features and enhancements introduced in the Real-Time Workshop Embedded Coder 3.0.

If you are upgrading from a release earlier than Release 12.1, then you should see the Real-Time Workshop Embedded Coder 2.0 “Release Summary” on page 2-2. For an introduction to the Real-Time Workshop Embedded Coder and for information about demos for the product, see the “Product Overview” section of the Real-Time Workshop Embedded Coder User’s Guide.

New User’s Guide

The Real-Time Workshop Embedded Coder is now documented in a separate manual, the Real-Time Workshop Embedded Coder User’s Guide. The new manual includes and updates the information about the Real-Time Workshop Embedded Coder that was formerly included in the Real-Time Workshop documentation.

The new manual also includes an appendix on generating ASAP2 files.

Auto-generated Main Program

The new **Generate an example main program** option lets you generate a fully commented, model-specific example main program module, `ert_main.c`. The code requires minimal modification for deployment in your environment. You can generate the example main program as either:

- A bare-board program designed to run under control of a real-time clock, without a real-time operating system
- An example showing how to deploy the code under the VxWorks real-time operating system

By default, **Generate an example main program** is on. Note that once you have generated and customized the main program, you should take care to avoid overwriting your customized version.

See the “Data Structures and Program Execution” section of the section of the Real-Time Workshop Embedded Coder User’s Guide for further information about the operation of the main program module.

The Real-Time Workshop Embedded Coder continues to provide a static (non-generated) version of `ert_main.c` as a template example for developing embedded applications. To use the static version of `ert_main.c` (or continue using a customized version):

- Turn **Generate an example main program** off.
- Read the next section, “Modifying the Static Main Program Module” and make changes to `ert_main.c` as required by your application.

Modifying the Static Main Program Module

This section describes modifications you may need to make to use the static version of `ert_main.c` (or continue using a customized version).

- The multi-rate scheduling code that was in `ert_main.c` is now generated with the model code. Therefore, if your model has multiple rates, note that multi-rate systems will not operate correctly unless:
 - The multirate scheduling code is removed. The relevant code is tagged with the keyword `REMOVE` in comments. (See also the Version 3.0 comments in `ert_main.c`.)
 - Use the `MODEL_SETEVENTS` macro (defined in `ert_main.c`) to set the event flags instead of accessing the flags directly. The relevant code is tagged with the keyword `REPLACE` in comments.
- Remove old `#include ertformat.h` directives. `ertformat.h` is no longer required, and will be obsoleted in a future release. The following macros, formerly defined in `ertformat.h`, are now defined within `ert_main.c`:
 - `EXPAND_CONCAT`
 - `CONCAT`
 - `MODEL_INITIALIZE`
 - `MODEL_STEP`
 - `MODEL_TERMINATE`
 - `MODEL_SETEVENTS`
 - `RT_OBJ`
- If applicable, follow comments in the code regarding where to add code for reading/writing model I/O and saving/restoring FPU context.
- When the **Generate code only** and **Generate an example main program** options are off, the Real-Time Workshop Embedded Coder generates the file `autobuild.h` to provide an interface between the main module and

generated model code. If you create your own static main program module, you would normally include `autobuild.h`.

Alternatively, you can suppress generation of `autobuild.h`, and include `model.h` directly in your main module. To suppress generation of `autobuild.h`, use the following statement in your system target file:

```
%assign AutoBuildProcedure = 0
```

- If you have cleared the **Terminate function required** option, remove or comment out the following in your production version of `ert_main.c`:
 - The `#if TERMFN...` compile-time error check
 - The call to `MODEL_TERMINATE`
- If you do *not* want to combine output and update functions, clear the **Single output/update function** option and make the following changes in your production version of `ert_main.c`:
 - Replace calls to `MODEL_STEP` with calls to `MODEL_OUTPUT` and `MODEL_UPDATE`.
 - Remove the `#if ONESTEPFN...` error check.
- The static `ert_main.c` module does not support the **Generate Reusable Code** option. Use this option only if you are generating a main program. The following error check will raise a compile-time error if **Generate Reusable Code** is used illegally:

```
#if MULTI_INSTANCE_CODE==1
```
- The static `ert_main.c` module does not support the **External Mode** option. Use this option only if you are generating a main program. The following error check will raise a compile-time error if **External Mode** is used illegally:

```
#ifdef EXT_MODE
```

Code Generation Options

This section describes new Real-Time Workshop Embedded Coder code generation options. These options are available via the **ERT code generation options** menus of the Real-Time Workshop page of the **Simulation Parameters** dialog box.

External Mode

Select this option to generate external mode communications support code in the target program. See also “External Mode Support” on page 1-7.

Generate an Example Main Program

This option lets you generate a model-specific example main program module. See “Auto-generated Main Program” on page 1-2.

Generate Reusable Code

When this option is selected, the Real-Time Workshop Embedded Coder generates reusable, reentrant code for the model. See “Reusable Code Generation” on page 1-9.

Parameter Structure

See “Hierarchical Parameter Structures” on page 1-8

Suppress Error Status in Real-Time Model Data Structure

If you do not need to log or monitor error status in your application, select this option.

By default, the real-time model data structure (rtM) includes an error status field (data type string). This field lets you log and monitor error messages via macros provided for this purpose (see *model.h*). The error status field is initialized to NULL. If **Suppress error status in real-time model data structure** is selected, the error status field is not included in rtM. Selecting this option may also cause the real-time model data structure to disappear completely from the generated code.

When generating code for multiple models that will be integrated together, make sure that the **Suppress error status in real-time model data structure** option is set the same for all of the models. Otherwise, the integrated application may exhibit unexpected behavior. For example, if the option is selected in one model but not in another, the error status may or may not be registered by the integrated application.

Do not select **Suppress error status in real-time model data structure** if the **MAT-file logging** option is also selected. The two options are incompatible.

Target Floating Point Math Environment

The Target Floating Point Math Environment popup menu provides two options. If you select the ANSI_C option (the default), the Real-Time Workshop Embedded Coder generates calls to the ANSI C (ANSI X3.159-1989) math library for floating point functions. If you select the ISO_C option, Real-Time Workshop Embedded Coder generates calls to the ISO C (ISO/IEC 9899:1999) math library wherever possible.

If your target compiler supports the ISO C (ISO/IEC 9899:1999) math library, we recommend selecting the ISO_C option and setting your compiler's ISO C option. This will generate calls to the ISO C functions wherever possible (for example, `sqrtf()` instead of `sqrt()` for single precision data) and ensure that you obtain the best performance your target compiler offers.

If your target compiler does not support ISO C math library functions, use the ANSI_C option.

ECRobot Target Example

The ECRobot (Embedded Coder Robot) target is a simple example of a custom target based on the Real-Time Workshop Embedded Coder. The ECRobot target was originally developed as a training example for use in classes offered to Real-Time Workshop Embedded Coder users. In this release, the ECRobot target is available to all Real-Time Workshop Embedded Coder users as an example and demonstration.

The ECRobot target files are automatically installed with the Real-Time Workshop Embedded Coder. Source code files, control files, demonstration models, and documentation for the target are installed in the directory

```
matlabroot/toolbox/rtw/targets/ECRobot
```

Note The ECRobot target requires an operating system kernel, a cross-compiler and support utilities that are available on the Web. For instructions on how to obtain and install these utilities, see the file `readme.html` in the `ECRobot/documentation` directory.

Note that the ECRobot target uses a Windows-based cross-compiler and other utilities; it is therefore hosted on Windows 2000 or Windows XP. A UNIX configuration is not planned.

Programs generated by the ECRobot target run on the Command System (RCX™) module of the LEGO® MINDSTORMS™ Robotics Invention System2.0™. This platform affords an inexpensive and simple way to study concepts and techniques essential to developing a custom embedded target, and to develop, run and observe generated programs.

The files included with the target illustrate typical approaches to problems encountered in custom target development, including:

- Interfacing a Real-Time Workshop Embedded Coder generated program to an external real-time operating system (RTOS) or kernel
- Implementing device drivers, via wrapper S-functions, for use in simulation and inlined code generation
- Customizing a system target file by adding code generation options and adding the target to the System Target File Browser
- Customizing a template makefile to use a target specific cross compiler and download generated code to the target hardware

External Mode Support

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

The **External mode** option is available via the **ERT code generation options (2)** category of the Real-Time Workshop page of the **Simulation Parameters** dialog box.

See the “External Mode” section of the Real-Time Workshop User’s Guide for further information.

GetSet Custom Storage Class for Data Store Memory

A new custom storage class, `GetSet`, has been added to provide accessor methods for memory associated with Data Store Memory blocks. This custom storage class is used in conjunction with Data Store Memory blocks. The purpose of the `GetSet` class is to generate code that reads (gets) and writes (sets) data via functions.

See the “Custom Storage Classes” section of the Real-Time Workshop Embedded Coder User’s Guide for detailed documentation of the `GetSet` class.

Hierarchical Parameter Structures

The **Parameter structure** menu lets you control how parameter data is generated for reusable subsystems. (If you are not familiar with reusable subsystem code generation, see “Nonvirtual Subsystem Code Generation Options” in the Real-Time Workshop User’s Guide for further information.)

The **Parameter structure** menu is available via the **ERT code generation options (3)** menu item of the Real-Time Workshop page of the **Simulation Parameters** dialog box.

The **Parameter structure** menu is enabled when the **Inline parameters** option is on. The menu lets you select the following options:

- **Hierarchical:** This option is the default. When the Hierarchical option is selected, the Real-Time Workshop Embedded Coder generates a separate header file, defining an independent parameter structure, for each subsystem that meets the following conditions:
 - The **Reusable function** option is selected in the subsystem’s **RTW system code** pop-up menu, and the subsystem meets all conditions for generation of reusable subsystem code.
 - The subsystem does not access any parameters other than its own (such as parameters of the root-level model).

When the Hierarchical option is selected, each generated subsystem parameter structure is referenced as a substructure of the root-level parameter data structure, which is therefore called a hierarchical data structure.

- **Non-hierarchical:** When this option is selected, the Real-Time Workshop Embedded Coder generates a single parameter data structure. This is a flat data structure; subsystem parameters are defined as fields within this structure.

Real-Time Model Structure Replaces Real-Time Object and Logging Object

The Real-Time Workshop Embedded Coder now encapsulates information about the root model in the *real-time model* data structure. We refer to the real-time model data structure as `rtM`.

`rtM` replaces the real-time object (`RT_OBJ`) and the logging object, which were used in previous releases. If your code accesses these objects through the macros provided with previous releases, it will continue to work.

See the “Data Structures and Code Modules” section of the Real-Time Workshop Embedded Coder User’s Guide for further information.

Reusable Code Generation

The **Generate reusable code** option lets you generate reusable, reentrant code from a model or subsystem. When this option is selected, data structures such as block states, parameters, external outputs, etc. are passed in (by reference) as arguments to *model_step* and other generated model functions. These data structures are also exported via *model.h*.

In some cases, the Real-Time Workshop Embedded Coder may generate code that will compile but is not reentrant (although it may still be acceptable for your application). For example, if a signal or parameter has a storage class other than `Auto`, the generated code is not reentrant, because the code must access the global data directly. To handle such cases, the **Reusable code error diagnostic** menu is enabled when **Generate reusable code** is selected. This menu offers a choice of three severity levels for diagnostics to be displayed in such cases:

- **None:** build proceeds without displaying a diagnostic message
- **Warn:** build proceeds after displaying a warning message
- **Error:** build aborts after displaying an error message

In some cases, the Real-Time Workshop Embedded Coder is unable to generate valid and compilable code. For example, if the model contains any of the following, the code generated would be invalid.

- A Stateflow chart that outputs function-call events
- An S-Function that is not code-reuse compliant
- A subsystem triggered by a wide function call trigger

In these cases, the build will terminate after reporting the problem.

When **Generate reusable code** option is not selected (the default), model data structures are statically allocated and accessed directly in the model code. Therefore the model code is neither reusable nor reentrant.

Revised Packaging of Generated Code Files

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

Note The file packaging of the Real-Time Workshop Embedded Coder differs slightly (but significantly) from the file packaging employed by the GRT, GRT malloc, and other non-embedded targets. See the Real-Time Workshop User's Guide for further information.

Table 1-1: Real-Time Workshop Embedded Coder File Packaging

File	Description
<i>model.c</i>	Contains entry points for all code implementing the model algorithm (<i>model_step</i> , <i>model_initialize</i> , <i>model_terminate</i> , <i>model_SetEventsForThisBaseStep</i>).
<i>model_private.h</i>	Contains local defines and local data that are required by the model and subsystems. This file is included by the generated source files in the model. You do not need to include <i>model_private.h</i> when interfacing hand-written code to a model.

Table 1-1: Real-Time Workshop Embedded Coder File Packaging (Continued)

File	Description
<i>model.h</i>	<p>Defines model data structures and a public interface to the model entry points and data structures. Also provides an interface to the real-time model data structure (<i>model_rtM</i>) via accessor macros. <i>model.h</i> is included by subsystem .c files in the model.</p> <p>If you are interfacing your hand-written code to generated code for one or more models, you should include <i>model.h</i> for each model to which you want to interface.</p>
<i>model_data.c</i> (conditional)	<i>model_data.c</i> is conditionally generated. It contains the declarations for the parameters data structure and the constant block I/O data structure. If these data structures are not used in the model, <i>model_data.c</i> is not generated. Note that these structures are declared extern in <i>model.h</i> .
<i>model_types.h</i>	Provides forward declarations for the real-time model data structure and the parameters data structure. These may be needed by function declarations of reusable functions. <i>model_types.h</i> is included by all the generated header files in the model.
ert_main.c (optional)	This file is generated only if the Generate an example main program option is on. (This option is on by default). See “Auto-generated Main Program” on page 1-2.
autobuild.h (optional)	<p>This file is generated only if the Generate code only and Generate an example main program options are off. See “Auto-generated Main Program” on page 1-2.</p> <p>autobuild.h contains #include directives required by the static version of the ert_main.c main program module. Since the static ert_main.c is not created at code generation time, it includes autobuild.h to access model-specific data structures and entry points.</p>

Table 1-1: Real-Time Workshop Embedded Coder File Packaging (Continued)

File	Description
<code>model_pt.c</code> (optional)	Provides data structures that enable a running program to access model parameters without use of external mode. To learn how to generate and use the <code>model_pt.c</code> file, see “C API for Parameter Tuning” in the Real-Time Workshop documentation.
<code>model_bio.c</code> (optional)	Provides data structures that enable your code to access block outputs. To learn how to generate and use the <code>model_bio.c</code> file, see “Signal Monitoring via Block Outputs” in the Real-Time Workshop documentation.

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

- `model_common.h` (replaced by `model_types.h` and `model_private.h`)
- `model_export.h` (replaced by `model.h`)
- `model_prm.h` (replaced by `model_data.c`)
- `model_reg.h` (subsumed by `model_.c`)

See also “Code Modules” in the Real-Time Workshop Embedded Coder documentation.

Template Makefile for Tornado

We have provided a simplified version of the Tornado target template makefile support deployment of Real-Time Workshop Embedded Coder code on the VxWorks operating system. See `matlabroot/rtw/c/ert/ert_tornado.tmf` for details.

Major Bug Fixes

Fixed Incorrect Effects of Expression Folding for Action Subsystems

With expression folding enabled, models containing action subsystems (such as For Iterator Subsystems or While Iterator Subsystems) could generate invalid or inefficient code. This problem has been fixed.

Removed License Restriction on Loading Objects with Custom Storage Classes

In Release 12, the Real-Time Workshop Embedded Coder license was checked when loading Simulink data objects that contained custom storage classes. This license dependency has been removed; the license is now checked only when you generate code with data containing custom storage classes.

If you have created your own subclasses of Simulink data objects that contain custom storage classes, you should reload your classes into the Simulink Data Class Designer and regenerate them to remove the license dependency.

Known Software and Documentation Problems

This section describes a known documentation problem in Version 1.0.1.

Considerations for Use of Initialize Internal Data Option

This note supplements the discussion of the **Initialize internal data** option in the “Basic Code Generation Options” section of the Real-Time Workshop Embedded Coder User’s Guide. This information was not included in previous editions of that document.

Consider carefully the effect of the **Initialize internal data** option when generating code from a model or generating a Real-Time Workshop Embedded Coder wrapper S-function. By default, the **Initialize internal data** option is turned on, guaranteeing that memory will be in a known state each time the code begins execution.

If you turn the option off, running a model (or a generated S-function) multiple times can result in different answers for each run. This behavior is sometimes desirable. For example, you can turn off **Initialize internal data** if you want to test the behavior of your design during a warm boot (i.e., a restart without full system reinitialization).

In cases where you have turned off **Initialize internal data** but still want to get the same answer on every run from a Real-Time Workshop Embedded Coder generated S-function, you can use either of the following MATLAB commands before each run:

```
clear <SFcnName> (where * is the name of the S-function)
```

or

```
clear mex
```

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from the Real-Time Workshop Embedded Coder 2.0 to Version 3.0.

Changes to Main Program Module (`ert_main.c`)

In this release, the Real-Time Workshop Embedded Coder generates a model-specific example main program module by default. For backwards compatibility, we continue to provide a static (non-generated) version of `ert_main.c`.

To use the static version of `ert_main.c`, (or continue using a previously customized version), note that certain modifications to `ert_main.c` may be required. See “Modifying the Static Main Program Module” on page 1-3 for modification guidelines.

HTML Code Generation Report Changes

In prior releases, the **Generate HTML report** option was available only for the Real-Time Workshop Embedded Coder. In the current release, a limited report is available for all targets (except the S-Function target and the Rapid Simulation target), while the Real-Time Workshop Embedded Coder continues to generate a more extensive report.

The **Generate HTML report** option is now located in the **General code generation options** category of the Real-Time Workshop page of the **Simulation Parameters** dialog box. The option is on by default.

See “Generating a Code Generation Report” in the Real-Time Workshop Embedded Coder User’s Guide for further information.

Include Model Name in Exported Structures Option Superseded

The **Include model name in exported structures** option has been superseded by the more general **Prefix model name to global identifiers** option. When this option is on, the Real-Time Workshop prefixes subsystem function names with the name of the model (`model_`). In addition, the model name is prefixed to the names of functions and data structures at the model level. This is useful in cases where you want to compile and link code from two or more models into a single executable, without name clashes.

Prefix model name to global identifiers is on by default. The option is located in the **General code appearance options** category of the Real-Time Workshop page of the **Simulation Parameters** dialog box.

Replace Obsolete Header File #includes

Generated code is packaged into different files in this release (see “Revised Packaging of Generated Code Files” on page 1-10). If you have interfaced your hand-written code to code generated by previous releases of Real-Time Workshop Embedded Coder, you may need to remove dependencies on header files that are no longer generated.

Use `#include model.h` directives, and remove `#include` directives referencing any of the following:

- `model_common.h`
- `model_export.h`
- `model_prm.h`
- `model_reg.h`

Update Your Custom System Target Files

The Real-Time Workshop Embedded Coder template makefile, `ert.tmf`, now contains additional identifiers that were not defined in release 12.1 or earlier releases. If you are using an older system target file (such as a custom target file based on an earlier version of `ert.tlc`) with the new `ert.tmf` file, you should update the system target file to make sure that no make options are missing.

All missing makefile options are reported on the MATLAB window during the build process, so that you can easily update your custom system target files.

Real-Time Workshop Embedded Coder

2.0 Release Notes

Release Summary	2-2
New Features	2-3
Custom Storage Classes for Data Objects	2-3
Enhanced Code Generation Options	2-4
Virtualized Output Ports Optimization	2-4
Improved HTML Code Generation Report	2-4

Release Summary

Release 2.0 of the Real-Time Workshop Embedded Coder is a major upgrade, incorporating

- Significant improvements in efficiency and readability of generated code. Many improvements in Real-Time Workshop code generation technology are especially applicable to embedded systems development. These include expression folding and buffer optimizations. For further information on these features, see the Real-Time Workshop 4.1 Release Notes.
- Custom storage classes for signal, state, and parameter objects, for embedded systems development.
- Additional and enhanced code generation options.
- Improved HTML code generation report.

New Features

This section introduces the new features and enhancements added in the Real-Time Workshop Embedded Coder 2.0, since the Real-Time Workshop Embedded Coder 1.0.

For information about Real-Time Workshop Embedded Coder features that are incorporated from Version 1.0, see “Introduction to the Real-Time Workshop Embedded Coder” on page 3-2.

Custom Storage Classes for Data Objects

The Real-Time Workshop Embedded Coder 2.0 implements a number of predefined custom storage classes that are useful in embedded systems development. These classes extend the built-in storage classes provided by the Real-Time Workshop. The built-in classes provide limited control over the form of the code generated for references signals, parameters, and other types of data. These storage classes are suitable for a simulation or rapid prototyping environment, but embedded system designers often require greater control over the representation of data.

Using Real-Time Workshop Embedded Coder custom storage classes, you can define and generate constructs such as bit fields or structs from your model, and easily interface data structures to externally written code.

See “Custom Storage Classes” in the *Real-Time Workshop Embedded Coder User’s Guide* for further information.

Note To create Simulink data objects with custom storage classes from M-code, you must have a Real-Time Workshop Embedded Coder license. To share data objects of this type with users who do not have the required license, save the objects in a MAT-file, which the users can load into the MATLAB workspace.

Enhanced Code Generation Options

Initialize Floats and Doubles to 0.0

This option lets you control how internal storage for floats and doubles is initialized. You can initialize floats and doubles to the integer bit pattern 0 (all bits off) or set float and double storage explicitly to the value 0.0.

See “Initialize Floats and Doubles to 0.0” in the *Real-Time Workshop Embedded Coder User’s Guide* for further information.

MAT-File Logging Off by Default

In prior releases, the **MAT-file logging** option was on by default. In Real-Time Workshop Embedded Coder 2.0, the **MAT-file logging** option is off by default. We recommend this setting because it eliminates the extra code and memory usage required to maintain logging variables.

Virtualized Output Ports Optimization

The virtualized output ports optimization lets you eliminate code and data storage associated with root output ports under certain conditions.

See “Virtualized Output Ports Optimization” in the *Real-Time Workshop Embedded Coder User’s Guide* for further information.

Improved HTML Code Generation Report

The format of the Real-Time Workshop Embedded Coder code generation report has been enhanced for readability. The “Optimizations” section lists additional options that will better optimize your code. Links from the report to the online Real-Time Workshop Embedded Coder documentation have been expanded.

See “Generating a Code Generation Report” in the *Real-Time Workshop Embedded Coder User’s Guide* for further information.

Real-Time Workshop Embedded Coder

1.0 Release Notes

Introduction to the Real-Time Workshop

Embedded Coder	3-2
Real-Time Workshop Embedded Coder Demos	3-3

Introduction to the Real-Time Workshop Embedded Coder

The Real-Time Workshop Embedded Coder is an add-on product for use with the Real-Time Workshop. The Real-Time Workshop Embedded Coder is designed for generation of highly optimized code for embedded systems.

The Real-Time Workshop Embedded Coder replaces and enhances the Embedded Real-Time (ERT) target. It is 100% compatible with the ERT target.

In addition to supporting all previous functions of the ERT target, the Real-Time Workshop Embedded Coder includes the following enhancements:

- Support for singletasking multirate models.
- The **Create Simulink (S-Function) block** option lets you build a noninlined C MEX S-function wrapper that calls the generated code in one automated step. The option also builds a model containing the generated S-Function block.
- The **Generate HTML report** option writes a Web-viewable code generation report that describes code modules and helps to identify code generation optimizations relevant to your program. The report is automatically opened into the MATLAB Help Browser.
- The **Insert block descriptions in code** option lets you insert text from the **Description** field of blocks in your model as comments in generated code.
- The **Generate ASAP2 file** code generation option lets you export an ASAP2 file containing information about your model during the code generation process.

Real-Time Workshop Embedded Coder Demos

We have provided a number of demos to help you become familiar with features of the Real-Time Workshop Embedded Coder and to inspect generated code. To run the demos:

- 1 Type following command at the MATLAB prompt.

```
ecoderdemos
```

- 2 The Real-Time Workshop Embedded Coder demo window opens.
- 3 Double-click on any of the blue **Open** boxes to run the desired demo.
- 4 A Simulink window, containing a demonstration model and instructions, opens. Follow the instructions for the demo.

