Real-Time Workshop® Embedded Coder Release Notes

Contents

Summary by Version

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

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V4.6.1 (R2007a+)	No	No	Bug Reports Includes fixes	Printable Release Notes: PDF No changes to documentation
V4.6 (R2007a)	Yes Details	No	Bug Reports Includes fixes	Current product documentation
V4.5 (R2006b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.4.1 (R2006a+)	No	No	Bug Reports Includes fixes	No
V4.4 (R2006a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.3 (R14SP3)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.2.1 (R14SP2+)	No	No	Bug Reports Includes fixes	No
V4.2 (R14SP2)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V4.1 (R14SP1)	Yes Details	No	Fixed bugs	No
V4.0 (R14)	Yes Details	Yes Summary	Fixed bugs	No
V3.2.1 (R13SP2)	Yes Details	No	Fixed bugs	V3.2.1 product documentation

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
V3.2 (R13SP1+)	Yes Details	No	No bug fixes	No
V3.1 (R13SP1)	Yes Details	No	No bug fixes	No

About Release Notes

Use release notes when upgrading to a newer version to learn about new features and changes, and the potential impact on your existing files and practices. Release notes are also beneficial if you use or support multiple versions.

If you are not upgrading from the most recent previous version, review release notes for all interim versions, not just for the version you are installing. For example, when upgrading from V1.0 to V1.2, review the New Features and Changes, Version Compatibility Considerations, and Bug Reports for V1.1 and V1.2.

New Features and Changes

These include

- New functionality
- Changes to existing functionality
- Changes to system requirements (complete system requirements for the current version are at the MathWorks Web site)
- Any version compatibility considerations associated with each new feature or change

Version Compatibility Considerations

When a new feature or change introduces a known incompatibility between versions, its description includes a **Compatibility Considerations** subsection that details the impact. For a list of all new features and

changes that have compatibility impact, see the “Compatibility Summary for Real-Time Workshop Embedded Coder” on page 91.

Compatibility issues that become known after the product has been released are added to Bug Reports at the MathWorks Web site. Because bug fixes can sometimes result in incompatibilities, also review fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

MathWorks Bug Reports is a user-searchable database of known problems, workarounds, and fixes. The MathWorks updates the Bug Reports database as new problems and resolutions become known, so check it as needed for the latest information.

Access Bug Reports at the MathWorks Web site using your MathWorks Account. If you are not logged in to your MathWorks Account when you link to Bug Reports, you are prompted to log in or create an account. You then can view bug fixes and known problems for R14SP2 and more recent releases.

The Bug Reports database was introduced for R14SP2 and does not include information for prior releases. You can access a list of bug fixes made in prior versions via the links in the summary table.

Related Documentation at Web Site

Printable Release Notes (PDF). You can print release notes from the PDF version, located at the MathWorks Web site. The PDF version does not support links to other documents or to the Web site, such as to Bug Reports. Use the browser-based version of release notes for access to all information.

Product Documentation. At the MathWorks Web site, you can access complete product documentation for the current version and some previous versions, as noted in the summary table.

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

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

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	Printable Release Notes: PDF No changes to documentation: Current product documentation (V4.6)

Version 4.6 (R2007a) Real-Time Workshop Embedded Coder

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

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

New features and changes introduced in this version are

Controlling Step Function Prototypes for Models

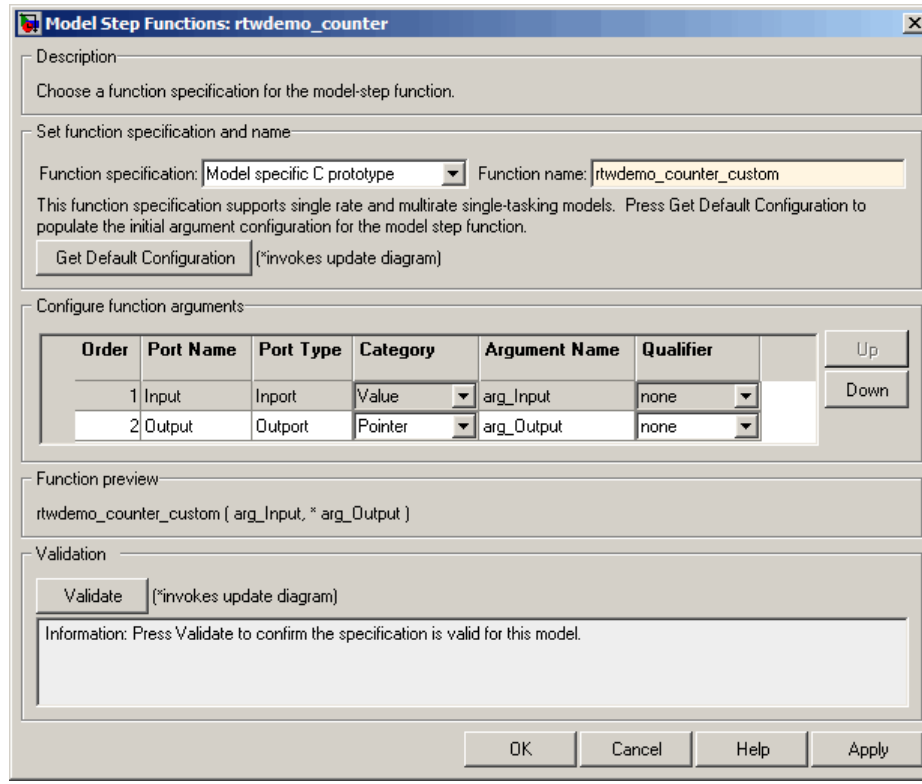
In previous releases, there were only limited ways to control the function prototype of an ERT-based model's generated *model*_step function. The default *model*_step function prototype resembles the following:

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

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

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

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



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

Alternatively, you can use the following function prototype control functions to programmatically control *model*_step function prototypes:

Function	Description
addArgConf	Add argument configuration information for Simulink model port to model-specific C function prototype
attachToModel	Attach model-specific C function prototype to loaded ERT-based Simulink model
getArgCategory	Get argument category for Simulink model port from model-specific C function prototype
getArgName	Get argument name for Simulink model port from model-specific C function prototype
getArgPosition	Get argument position for Simulink model port from model-specific C function prototype
getArgQualifier	Get argument type qualifier for Simulink model port from model-specific C function prototype
getDefaultConf	Get default configuration information for model-specific C function prototype from Simulink model to which it is attached
getFunctionName	Get function name from model-specific C function prototype
getNumArgs	Get number of function arguments from model-specific C function prototype
runValidation	Validate model-specific C function prototype against Simulink model to which it is attached
setArgCategory	Set argument category for Simulink model port in model-specific C function prototype
setArgName	Set argument name for Simulink model port in model-specific C function prototype
setArgPosition	Set argument position for Simulink model port in model-specific C function prototype
setArgQualifier	Set argument type qualifier for Simulink model port in model-specific C function prototype
setFunctionName	Set function name in model-specific C function prototype

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

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

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

For more information about controlling *`model`*`_step` function prototypes, see the sections and "Controlling model_step Function Prototypes" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "model_step Function Prototype Control Limitations" in the Real-Time Workshop Embedded Coder documentation.

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

New ModelStepFunctionPrototypeControlCompliant Target Configuration Parameter

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

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

To make a target compliant,

1. Use the `SelectCallback` function to set `ModelStepFunctionPrototypeControlCompliant` to `on`. This enables the feature infrastructure and user interface.

2. If your target uses a custom static main module, and if a nondefault function prototype control configuration is associated with a model, update the main module to call the function prototype controlled model step function. You can do this in either of the following ways:

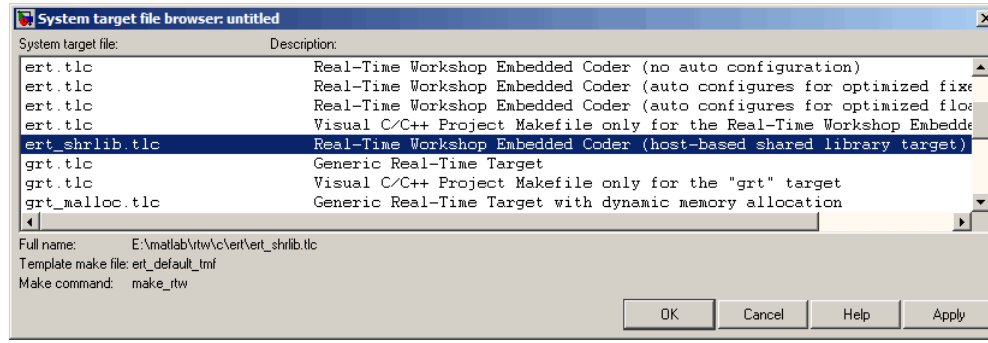
   a. Manually adapt your main module to declare appropriate model data and call the function prototype controlled model step function.

   b. Generate your main module using **Generate an example main program** on the **Templates** pane of the Configuration Parameters dialog box. This mechanism has been updated to declare model data and call the function prototype controlled model step function appropriately.

New ERT Target for Generating Host-Based Shared Libraries

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

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



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

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

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

#endif

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

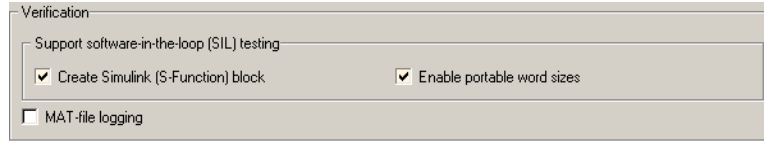
```
#else /* UNIX */
    handleLib = LOADLIB("./rtwdemo_counter.so", RTLD_LAZY);
#endif
#endif
...
    return(CLOSELIB(handleLib));
}
```

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

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

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

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



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

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

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

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

For more information about using portable word sizes for host-based SIL testing, see and “Validating ERT Production Code on the MATLAB Host Computer Using Portable Word Sizes” in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see “Portable Word Sizes Limitations” in the Real-Time Workshop Embedded Coder documentation.

New Code Style Options for Controlling Expression Optimizations in Generated Code

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

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

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

Enhanced MISRA-C Compliance

This release provides several enhancements to MISRA-C compliance, including

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

New and Enhanced Demos

The following demos have been added:

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

The following demo has been enhanced:

- `rtwdemo_sil`

Version 4.5 (R2006b) Real-Time Workshop Embedded Coder

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

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

New features and changes introduced in this version are

• "New Reference Documentation" on page 22

Efficiency Enhancements in Generated Code

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

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

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

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

- Prelookup
- Interpolation Using Prelookup

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

Ability to Control Use of Parentheses in Generated Code

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

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

Enhanced HTML Code Report Performance and Content

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

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

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

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

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

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

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

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

Note

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

New 'error' Hook Method for STF_make_rtw_hook.m

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

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

Maximum Length Enforced for Auto-Generated Identifiers in Generated Code

In previous releases, some auto-generated identifiers in generated code were allowed to exceed the **Maximum identifier length** specified on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box. Generated identifiers that exceeded the **Maximum identifier length** did not honor the user setting and potentially were inconsistent with ANSI-C or MISRA guidelines requiring identifiers to be unique within a prescribed length (31 characters).

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

For limitations that apply, see "Identifier Format Control Parameters Limitations" in the Real-Time Workshop Embedded Coder documentation. For upgrade and compatibility considerations, see "Compatibility Considerations" on page 19.

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

Compatibility Considerations

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

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

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

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

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

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

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

New Default Value for IncludeERTFirstTime Model Configuration Parameter

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

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

To include the `firstTime` argument in generated code, change the value of the `IncludeERTFirstTime` parameter to `on`. However, see the release note "Use of firstTime Argument to model_initialize Function to Be Discontinued" on page 21.

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

Compatibility Considerations

For ERT-based Simulink models newly created in R2006b, the code generated for the *`model`*`_initialize` function by default will no longer include the `firstTime` argument. As a result, existing custom static main programs that invoke *`model`*`_initialize` with the `firstTime` argument will need to be reconciled with the code generated for the *`model`*`_initialize` function. For example, you can

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

Use of firstTime Argument to model_initialize Function to Be Discontinued

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

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

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

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

New and Enhanced Demos

The following demos have been added:

Demo...	Shows How You Can...
`rtwdemo_importstruct`	Import externally defined parameters into Simulink. This model demonstrates how to generate code that accesses the fields of a data structure. The data structure is defined in legacy (hand-written) code and accessed via a pointer. This technique enables users to easily switch between complete sets of parameters at run time (for example, between reference and working versions).
`rtwdemo_osek`	Interface to the OSEK/VDX® real-time operating system. For more information, see “New General-Purpose OSEK/VDX Real-Time Operating System (RTOS) Example” on page 17.
`rtwdemo_parentheses`	Set the style of parenthesization in generated code to be Minimum (only parentheses required by C syntax), Nominal (parentheses added to optimize readability), or Maximum (parentheses obviate C precedence, as required by MISRA).

New Reference Documentation

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

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

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

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

Version 4.4 (R2006a) Real-Time Workshop Embedded Coder

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

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

New features and changes introduced in this version are

Nonvirtual Subsystem Option for Generating Modular Function Code

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

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

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

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

To be able to use this option,

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

To configure your subsystem for generating modular function code, you invoke the Subsystem Parameters dialog box and make a series of selections to

display and enable the **Function with separate data** option. For details, see "Nonvirtual Subsystem Modular Function Code Generation" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Nonvirtual Subsystem Modular Function Code Limitations" in the Real-Time Workshop Embedded Coder documentation.

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

Exporting Function-Call Subsystems

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

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

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

Identifier Format Control Parameters for Code Generation

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

In previous releases, the **Symbol format** parameter on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box allowed you to specify one macro string that affected naming for a range of

symbol types. In this release, several **Identifier format control** parameters allow you to exercise format control individually for

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

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

For a description of the new **Identifier format control** parameters and their use, see "Symbols Pane" and its subsection "Specifying Identifier Formats" in the Real-Time Workshop Embedded Coder documentation. For limitations that apply, see "Identifier Format Control Parameters Limitations" in the Real-Time Workshop Embedded Coder documentation. For upgrade and compatibility considerations, see "Compatibility Considerations" on page 27.

Compatibility Considerations

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

- Some identifiers that were allowed to exceed the **Maximum identifier length** (on the **Real-Time Workshop/Symbols** pane of the Configuration Parameters dialog box) in earlier releases are mangled in this release to conform to the maximum length. The types of identifiers that potentially are affected are Simulink global variables, Simulink global types, local variables, subsystem methods, and Simulink constant macros. The mangling is most likely to occur in models with long names.

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

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

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

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

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

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

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

New and Changed Memory Section Capabilities

This release provides new and changed memory section capabilities in Real-Time Workshop Embedded Coder. In previous releases, memory sections could be applied only to data objects defined in custom storage classes, and memory section pragmas could surround only a contiguous block of all data objects in that section. This release adds enhancements that

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

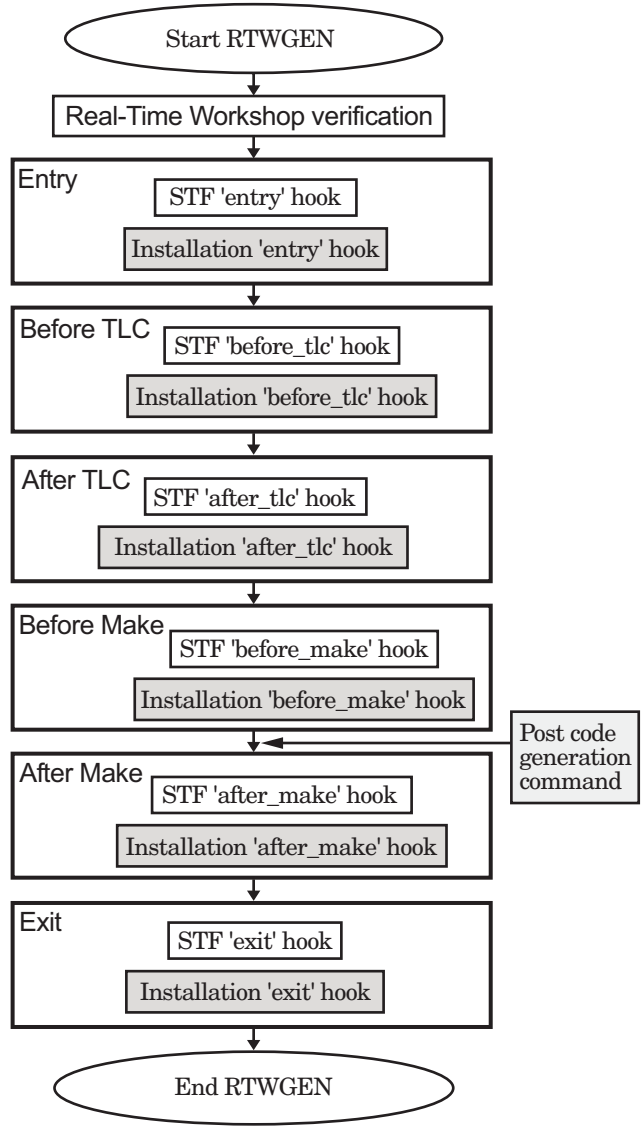
For detailed descriptions of the new memory section capabilities, see and the "Memory Sections" chapter in the Real-Time Workshop Embedded Coder documentation.

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

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

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

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



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

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

New sl_customization API for Customizing Data Objects

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

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

This new registration mechanism replaces earlier mechanisms involving the files `custom_user_type_registration.m` (for creating user data types) and `custom_user_object_type_info.m` (for registering `mpt` user object types). A script is provided to convert existing instances of `custom_user_type_registration.m` and `custom_user_object_type_info.m` to `sl_customization.m` (see "Compatibility Considerations" on page 31).

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

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

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

Compatibility Considerations

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

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

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

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

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

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

mpt Signal Object Enhancements

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

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

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

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

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

New IncludeERTFirstTime Model Configuration Parameter

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

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

New ERTFirstTimeCompliant Target Configuration Parameter

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

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

When this parameter is set to `off` and you attempt to set the new model parameter `IncludeERTFirstTime` to `off`, Real-Time Workshop Embedded Coder ignores the request and issues a warning indicating that you need to make the target compliant.

To make a target compliant,

1. Use the `SelectCallback` function to set `ERTFirstTimeCompliant` to `on`.
2. If your target uses a custom static main program, update it to handle the inclusion and suppression of the `firstTime` argument for a given model. One way to do this is to

a Make sure the target TLC file assigns 1 to `AutoBuildProcedure` when using a static main program. For example,

```
%assign AutoBuildProcedure = !GenerateSampleERTMain
```

b In the generated header file `autobuild.h`, the macro INCLUDE_FIRST_TIME_ARG will be set to 0 if the `IncludeERTFirstTime` parameter is set to `off` or 1 if the parameter is set to `on`.

c Inside the static main program, make sure to `#include autobuild.h` and then conditionally compile declarations and calls to the *`model`*`_initialize` function, based on the value of the INCLUDE_FIRST_TIME_ARG macro.

firstTime Argument to model_initialize Function

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

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

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

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

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

The script locates unlabeled signals connected with root I/O ports in the specified model. Each unlabeled signal will be labeled using its port name,

provided that the port name is a valid C identifier, is not a C keyword, and does not conflict with other signal and parameter names in the model. If the specified model is not already open, the script opens the model for viewing. You can examine the modifications and decide whether to save the model with the changes.

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

```
propagate_rootio_signal_names('rtwdemo_counter_test')

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

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

New and Enhanced Demos

The following demos have been added:

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

The following demos have been enhanced:

- `rtwdemo_namerules`
- `rtwdemo_symbols`

Version 4.3 (R14SP3) Real-Time Workshop Embedded Coder

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

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

New features and changes introduced in this version are

Data Type Replacement

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

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

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

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

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

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

HeaderFile Property Now Optional as Part of GetSet Data Object

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

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

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

Data Object Wizard Enhancements

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

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

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

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

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

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

ERT Automatic Configuration Changes

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

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

Compatibility Considerations

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

Documentation Enhancements

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

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

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

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

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

Version 4.2 (R14SP2) Real-Time Workshop Embedded Coder

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

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

New features and changes introduced in this version are

C++ Target Language Support

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

External Mode Support for ERT VxWorks Example Target

The ERT VxWorks example target now includes full support for Simulink® external mode. External mode lets you use your Simulink block diagram as a front end for a target program that runs on external hardware or in a separate process on your host computer, and allows you to tune parameters and view or log signals as the target program executes. With this release, you can generate code to support external mode communication between host (Simulink) and ERT VxWorks target systems. For more information, see "Using External Mode with the ERT Target" in the Real-Time Workshop Embedded Coder documentation.

Custom Storage Classes with ERT S-Functions

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

For more information, see "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 allowed you to select the **Alias override naming rule** option for an `mpt` data object. As explained in the Module Packaging Features document, this resulted in the name that you typed in the **Alias** field overriding the global naming rule for the selected data object. This only applied to `mpt` data objects, not to Simulink data objects.

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

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

Before this release, when you specified a **Definition file** name on the Model Explorer dialog box for a data object and did not specify a **Header file** name, the code generator generated a header file in which it declared the data object.

The code generator used the same name for the header file (for example, `data.h`) as you specified for the definition file (for example, `data.c`).

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

Enhanced MPF Documentation of Managing Data Dictionary

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

File custom_user_type_registration.m No Longer Automatically Called During Code Generation

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

Compatibility Considerations

The following compatibility consideration applies if you are upgrading from V4.1 (R14SP1) to V4.2 (R14SP2), V4.2.1 (R14SP2+), or V4.3 (R14SP3). If you are upgrading to V4.4 (R2006a) from V4.1 or later, see the release notes for "Version 4.4 (R2006a) Real-Time Workshop Embedded Coder" on page 24 .

If you modified and saved `custom_user_type_registration.m` in V4.1 (R14SP1), you must now create the `Simulink.AliasType` objects

corresponding to your user-defined data types before generating code for your model. For example, you can:

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

Version 4.1 (R14SP1) Real-Time Workshop Embedded Coder

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

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

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

Significant Documentation Corrections

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

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

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

Version 4.0 (R14) Real-Time Workshop Embedded Coder

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

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—See "Upgrading from R13SP1+ or R13SP2" on page 67 and "Generating R13SP1+ or R13SP2 Code From ERT-Based Simulink Models Created In R14 or Later" on page 76. See also Summary.	Fixed bugs	No

New features and changes introduced in this version are

Expanded Documentation Collection

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

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

New ERT Target Options User Interface

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

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

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

Note If you have developed a custom target based on the ERT target (or any other Real-Time Workshop target) see "Defining and Displaying Custom Target Options" on page 69 for a discussion of compatibility issues that may affect the appearance and operation of your target.

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

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

Pane and Subpane	Option	Usage
Real-Time Workshop: Comments	Simulink block descriptions	Include text specified in the **Description** field of Block Properties dialogs as comments in the generated code for the corresponding blocks.
	Stateflow object descriptions	Include text specified in the **Description** field of state object Properties dialogs as comments in the generated code for the corresponding objects.
	Simulink data object descriptions	Include text specified in the **Description** field of object properties defined in the Simulink Model Explorer as comments in the generated code for the corresponding objects.
	Custom comments (`mpt` objects only)	Include comments just above signals and parameter identifiers in the generated code as specified in an 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.

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

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

Pane and Subpane	Option	Usage
	Target operating system	Generate a bareboard main program designed to run under control of a real-time clock without a real-time operation system or a fully commented example showing how to deploy the code under the VxWorks real-time operating system.
Real-Time Workshop: Data Placement	Data definition	Specify whether data is to be defined in the generated source file or in a single separate header file.
	Data reference	Specify whether data is to be declared in the generated source file or in a single separate header file
	Module naming	Name the generated module using the same name as the model or a user-specified name.
	Signal display level	Specify whether to declare signal data objects as global data in the generated code.
	Parameter tune level	Declare a parameter data object as tunable global data in the generated code.
	#include file delimiter	Specify the #include file delimiter to be used in generated files that contain the #include preprocessor directive for MPF data objects.
	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 initialize floats and doubles	Specify whether internal storage, regardless of type, is to be cleared to the integer bit pattern 0 or the `memset` function is to set float and double storage to 0.0. Previously labeled **Initialize Floats and Doubles** to 0.0. Default is now cleared rather than set.

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 document.

GRT and ERT Target Unification

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

- Both products use a common format for backend generated code.
- The feature list common to both products is expanded.

- Some features and efficiencies formerly exclusive to the ERT target are now available to the GRT target. Conversely, the ERT target now supports some features that were previously available only with the GRT target.
- Conversion from GRT-based targets to ERT-based targets is greatly simplified.

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

Support for Continuous Time Blocks

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

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

```
showblockdatatypetable
```

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

Support for Continuous Solvers

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

Note Custom targets must be modified to support continuous time. The required modifications are described in "Supporting Continuous Time in Custom Targets" on page 71.

Support for Noninlined S-Functions

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

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

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

The Module Packaging Features document 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 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 an ASAP2 File" in the Real-Time Workshop documentation.

Code Generation with User-Defined Data Types

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

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

In code generation, you can use user-defined data type objects to map your own data type definitions to Simulink built-in data types, and to generate `#include` directives specifying your own header files, containing your data type definitions.

See the "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 detailed description of CSCs and the Custom Storage Class Designer.

Compatibility with Previous CSCs

In prior releases, CSCs were implemented via special `Simulink.CustomSignal` and `Simulink.CustomParameter` classes. We recommend that you consider replacing `Simulink.CustomSignal` and `Simulink.CustomParameter` objects

in your models with equivalent `Simulink.Signal` and `Simulink.Parameter` objects.

Minor changes have been made in the `Simulink.CustomSignal` and `Simulink.CustomParameter` classes. See "Custom Storage Class Compatibility Issues" on page 68 for information on these changes.

More Efficient Multi-Rate Multitasking Code Generation

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

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

```
model_stepN
```

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

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

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

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

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

See the "Data Structures, Code Modules, 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 multirate, multitasking model, thereby improving performance of the generated code.

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

```
%assign SuppressSetEventsForThisBaseRateFcn = 1
```

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

New Callbacks Defined for System Target Files

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

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

Note If you have developed a custom target and you want it to be compatible with model referencing, you must implement a `SelectCallback` function to declare model reference compatibility. See 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
```

Upgrading from R13SP1+ or R13SP2

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

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

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

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

For example:

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

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

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

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

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

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

Custom Storage Class Compatibility Issues

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

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

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

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

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

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

Defining and Displaying Custom Target Options

For Release 14, extensive improvements and revisions have been made in the appearance and layout of code generation options and other target-specific options for Real-Time Workshop targets. If you have developed a custom target, you should take advantage of the Model Explorer and Configuration Parameters dialogs to present target options to end users. If you choose not to, a mechanism for using the old-style Simulation Parameters dialog box is available for backwards compatibility.

The “System Target Files” chapter of Developing Embedded Targets 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 “Using rtwoptions Callbacks in Release 14 or Later” in the “System Target Files” chapter of Developing Embedded Targets.
- 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 “Target Options Inheritance in Release 14 or Later” in the “System Target Files” chapter of Developing Embedded Targets.
- Display of target options: Your target options are displayed differently, and you might want to reorganize them. See “Target Options Display in Release 14 or Later” in the “System Target Files” chapter of Developing Embedded Targets 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 Developing Embedded Targets 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 single-rate code.

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

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

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

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

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

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

rtwtypes.h Replaces tmwtypes.h

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

Updating Customized Static Main Program Modules

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

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

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

   with

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

3. Search for `tmwtypes.h`. This search brings you to the "Includes" section.
4. Add the following include statement.

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

5. Delete the following include statements.

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

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

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

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

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

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

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

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

See the "Data Structures, Code Modules, 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 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.

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

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

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

```
Error executing build command: Error using ==> make_rtw
Error using ==> tlc_c
Error using ==> tlc_c (InvokeTLC)
Error:  Real-Time Workshop Error: Unable to locate ERT header file banner template:
ert_code_template.cgt.
```

To work around this problem in R13SP1+ or R13SP2, download and install a replacement version of the file *`matlabroot`*`/rtw/c/tlc/mw/setuplib.tlc`, as follows:

1 Go to the directory *matlabroot*/rtw/c/tlc/mw and rename the file setuplib.tlc to setuplib.tlc.old.

2 Click the appropriate link below to download a replacement version of setuplib.tlc. Place the file in the same directory as the file you renamed.

- setuplib_sp2.tlc
- setuplib_sp1plus.tlc

Note If you are not logged in to your MathWorks Account when you click the link, you will prompted to log in or create an account.

If you are reading the PDF or hard copy documentation, go to the MATLAB Help browser or The MathWorks web documentation to use the link.

---

3 Rename the downloaded file to setuplib.tlc.

Version 3.2.1 (R13SP2) Real-Time Workshop Embedded Coder

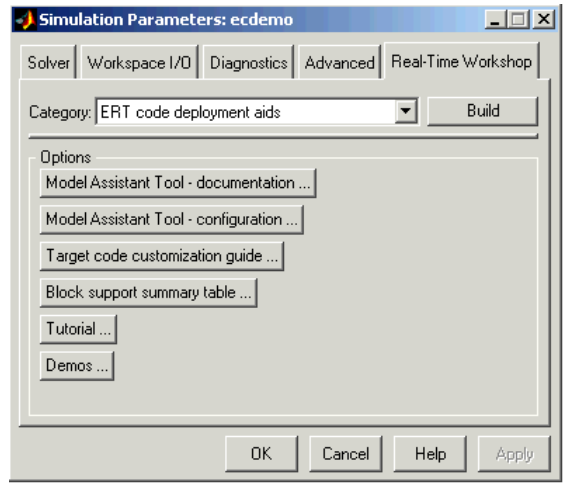
This table summarizes what's new in Version 3.2.1 (R13SP2):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Fixed bugs	V3.2.1 product documentation

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

ERT Code Deployment Aids Added to GUI

A new group of buttons has been added to the Embedded Real-Time (ERT) target options in the **Real-Time Workshop** pane of the Simulation Parameters dialog box. To access these buttons, select `ERT code deployment aids` from **Category** menu, as shown in the figure below.



The `ERT code deployment aids` buttons provide quick access to features and information that can help you to optimize your generated code. The buttons are:

- **Model Assistant Tool - documentation**: Click this button to view online help for the Model Assistant Tool in the MATLAB Help browser. You can also view this help by typing the MATLAB command

  ```
  modelassistant('help')
  ```

- **Model Assistant Tool - configuration**: Click this button to open the Model Assistant Tool for configuration of options.
- **Target code customization guide**: Click this button to view the "Advanced Code Generation Techniques" chapter of the Real-Time Workshop Embedded Coder documentation. The chapter documents useful code generation, optimization, and customization techniques for the ERT target. Most of the features described were introduced in Real-Time Workshop Embedded Coder 3.2 (see the release notes for"Version 3.2 (R13SP1+) Real-Time Workshop Embedded Coder" on page 80 for a summary).
- **Block summary support table**: Click this button to view the Simulink Block Data Type Support Table in the MATLAB Help Browser. The table describes the data types that are supported by the blocks in the main Simulink and Fixed-Point libraries. The table also identifies blocks that are suitable for production code generation. You can also view the table by typing the MATLAB command

  ```
  showblockdatatypetable
  ```

- **Tutorial**: Click this button to open an interactive Real-Time Workshop Embedded Coder tutorial demo in the in the MATLAB Help Browser. You can also view the tutorial demo by typing the MATLAB command

  ```
  ecodertutorial
  ```

- **Demos**: Click this button to open the Real-Time Workshop Embedded Coder demo suite. You can also view the demos by typing the MATLAB command

  ```
  ecoderdemos
  ```

Version 3.2 (R13SP1+) Real-Time Workshop Embedded Coder

This table summarizes what's new in Version 3.2 (R13SP1+):

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

New features and changes introduced in this version are:

Advanced Code Generation Techniques Documented

A new chapter, "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

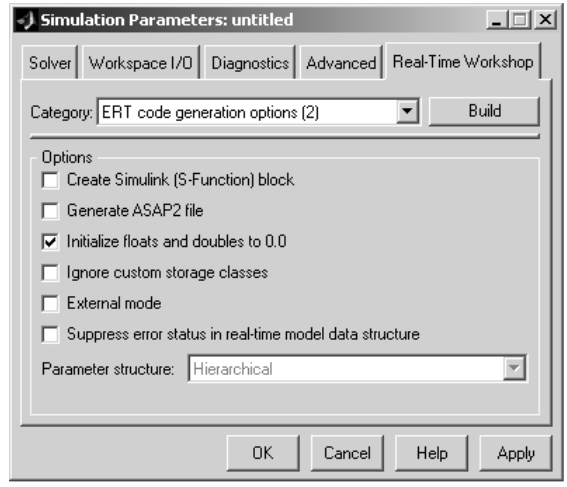
- A general hook file mechanism for adding target-specific customizations to the build process

New Code Generation Options

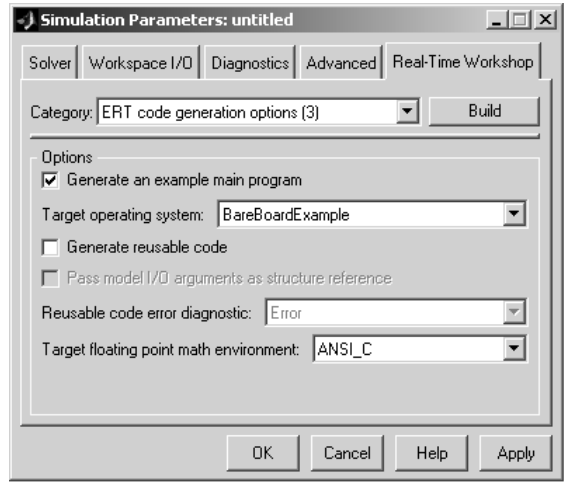
Several new code generation options have been added, and some changes have been made to the layout of Embedded Real-Time (ERT) target code generation options in the **Real-Time Workshop** pane of the Simulation Parameters dialog box.

Options Layout Changes and Additions

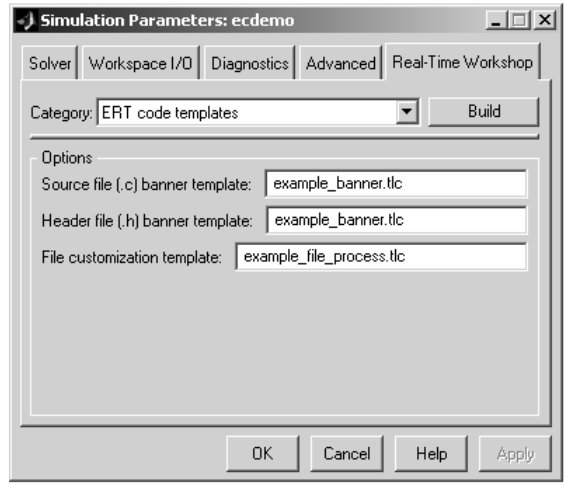
The **Suppress error status in real-time model data structure** option has been relocated to the `ERT code generation options (2)` category, as shown in this figure.



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 85.



A new group of options supporting use of *code templates*, a powerful and simple technique for customizing generated code, has been added. These options are available in the `ERT code templates` category of the **Real-Time Workshop** pane of the Simulation Parameters dialog box (see the figure below). Code templates are summarized in "Code Templates for Customizing Generated Code" on page 84.



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 options the user chooses manually are not disturbed.

The automatic configuration process, and utilities provided to support auto-configuration, are described in the "Advanced Code Generation Techniques" 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 Techniques" 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.

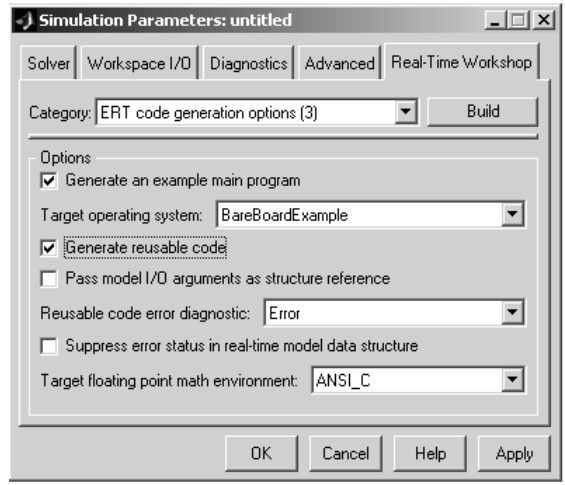
CFP templates are described in the "Advanced Code Generation Techniques" chapter of the Real-Time Workshop Embedded Coder User's Guide.

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 Techniques" 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 for further details.

Version 3.1 (R13SP1) Real-Time Workshop Embedded Coder

This table summarizes what's new in Version 3.1 (R13SP1):

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

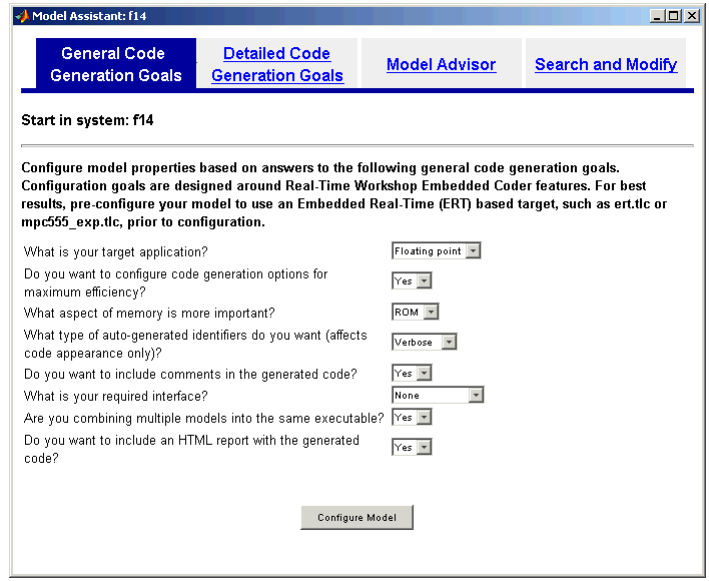
New features and changes introduced in this version are described here:

Model Assistant Tool

The Model Assistant Tool is a utility that lets you configure a model for code generation quickly. The Model Assistant Tool also helps you to identify aspects of your model that impede production deployment or limit code efficiency. You can use the Model Assistant Tool at any point in your design cycle, as it is completely independent from the code generation process.

The Model Assistant Tool is designed primarily for use with Real-Time Workshop Embedded Coder. It works most effectively with the Embedded Real-Time (ERT) target and with ERT-based targets (such as the Embedded Target for Motorola MPC555). It will also operate with other targets.

The figure below shows the top-level window of the Model Assistant Tool.



Four main components of the Model Assistant Tool provide a powerful and centralized interface for configuring settings for Simulink blocks, Stateflow® charts, models and subsystems. You select these components via the four buttons at the top of the Model Assistant display:

- **General Code Generation Goals**
- **Detailed Code Generation Goals**
- **Model Advisor**
- **Search and Modify**

These components are summarized in the next sections.

General Code Generation Goals

This component lets you quickly configure code generation settings based on specific goals, such as whether to optimize for RAM or ROM usage. Once you have decided the overall optimization and tradeoffs for your application, the Model Assistant Tool will select the model settings that best suit your goals.

Detailed Code Generation Goals

This component presents a centralized interface to the available code generation options. Options are grouped by category, and are applied across products.

Model Advisor

The Model Advisor component is particularly useful early in the design cycle. It provides an analysis of your model to ensure that you best utilize Real-Time Workshop Embedded Coder. You can check selected aspects of your model settings (for example, to identify possible inefficiencies such as blocks that generate saturation and rounding code) or choose **Select All** for a comprehensive analysis.

Search and Modify

This component is a powerful model search and modify engine. It reduces the effort of configuring a model block by block. The search feature helps you find attributes of blocks, lines, input ports, output ports, and annotations quickly. The modify feature lets you perform rapid batch operations on the search results. Frequently performed tasks are packaged conveniently into a single button click.

The **Search and Modify** component includes the following features:

- The **Frequent tasks** page lets you quickly perform common actions.
- The **Simulink object search** page lets you specify a general Simulink object search and modify action. This search mechanism is useful when you know the specific names of underlying attributes.
- The **Stateflow object search** page lets you quickly configure the Stateflow data in your model. This is particularly useful for converting data from floating point to fixed-point types.
- The **Search and replace Simulink text** page lets you quickly modify text for objects in Simulink. For example, you can change all occurrences of `'K1'` to `'K2'`. The semantics of the search and replace are the same as for the Stateflow search and replace tool that ships with Stateflow.
- Two **Parameter name search** mechanisms are provided:

- Search and modify parameters using prompt strings. This search mechanism is useful when you know the parameter by its dialog prompt string, but you don't know the name of the underlying attribute.
- "Fuzzy" search using property and/or value pairs. This search mechanism is useful for isolating the name of an underlying attribute.

Using the Model Assistant Tool

You run Model Assistant Tool from the MATLAB command line, via the `modelassistant` command. Before invoking the Model Assistant Tool, make sure that the desired target (such as the ERT target) is selected in the **Target Configuration** section of the **Real-Time Workshop** pane of the Simulation Parameters dialog box.

The following examples illustrate the `modelassistant` command syntax and its possible arguments.

To obtain detailed help on the Model Assistant Tool, type

```
modelassistant('help')
```

To invoke the Model Assistant Tool for the root system of a model, type

```
modelassistant('model')
```

where *`model`* is the name of the model.

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

```
modelassistant('subsystem')
```

where *`subsystem`* is the name of the subsystem.

You can also invoke the Model Assistant Tool for models and systems using the built-in Simulink `bdroot`, `gcb`, and `gcs` commands. For example:

```
modelassistant(gcs)
```

Further Help and Demos

The above sections have summarized the main features of the Model Assistant Tool. To obtain full online documentation on the Model Assistant Tool, type

```
modelassistant('help')
```

There are also three demo models available for the Model Assistant Tool: `advisordemo1`, `advisordemo2`, and `advisordemo3`.

Compatibility Summary for Real-Time Workshop Embedded Coder

This table summarizes new features and changes that might cause incompatibilities when you upgrade from an earlier version, or when you use files on multiple versions. Details are provided in the description of the new feature or change.

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version V4.6.1 (R2007a+)	None
V4.6 (R2007a)	None
V4.5 (R2006b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Maximum Length Enforced for Auto-Generated Identifiers in Generated Code" on page 18 • "New Default Value for IncludeERTFirstTime Model Configuration Parameter" on page 20
V4.4.1 (R2006a+)	None
V4.4 (R2006a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Identifier Format Control Parameters for Code Generation" on page 26 • "New sl_customization API for Customizing Data Objects" on page 31

Version (Release)	New Features and Changes with Version Compatibility Impact
V4.3 (R14SP3)	See the **Compatibility Considerations** subheading for this new feature or change: • "ERT Automatic Configuration Changes" on page 40
V4.2.1 (R14SP2+)	None
V4.2 (R14SP2)	See the **Compatibility Considerations** subheading for this new feature or change: • "File custom_user_type_registration.m No Longer Automatically Called During Code Generation" on page 46
V4.1 (R14SP1)	None
V4.0 (R14)	See: • "Upgrading from R13SP1+ or R13SP2" on page 67 • "Generating R13SP1+ or R13SP2 Code From ERT-Based Simulink Models Created In R14 or Later" on page 76
V3.2.1 (R13SP2)	None
V3.2 (R13SP1+)	None
V3.1 (R13SP1)	None