Release Notes for Simulink® Coder™

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Release Notes for Simulink® Coder™

Trademarks

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

Patents

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

Contents

R2012b

R2012a

R2011b

R2011a

R2012b

Version: 8.3
New Features: Yes
Bug Fixes: Yes

Unified and simplified code interface for ERT and GRT targets

Compatibility Considerations: Yes

Previously, Simulink® Coder™ software provided a static main program for GRT-based targets, *matlabroot*/rtw/c/grt/grt_main.c, that was distinct from the static main program that Embedded Coder® software provided for ERT-based targets, *matlabroot*/rtw/c/ert/ert_main.c.

Beginning in R2012b, Simulink Coder software provides a unified static main program for both GRT- and ERT-based targets:

matlabroot/rtw/c/src/common/rt_main.c

Generated code for GRT-based models is simplified and more consistent with generated code for ERT-based models. As a result, GRT- and ERT-based models can now use a common static main program. The benefits for GRT-based models include:

- The generated `rtModel` structure has a minimal number of fields.
- Unused macros no longer appear in the generated code.
- Multitasking behavior is consistent between GRT and ERT generated code.

Note

- If you are using the pre-R2012a GRT call interface (by selecting the model option **Classic call interface**) with a static main program, use the static main program *matlabroot*/rtw/c/grt/classic_main.c as a reference point.
- The previous GRT and ERT static main program files, *matlabroot*/rtw/c/grt/grt_main.c and *matlabroot*/rtw/c/ert/ert_main.c, have been removed from the software and are replaced by the new simplified and classic static main program files, *matlabroot*/rtw/c/src/common/rt_main.c and *matlabroot*/rtw/c/grt/classic_main.c.
- The generated main program file for ERT targets is still named ert_main.c/cpp.
- If you have an Embedded Coder license, see also "External mode support for ERT targets with static main" in the Embedded Coder release notes.

Compatibility Considerations

If you use a GRT-based target with a static main program, and if you configure your models with the simplified call interface that was made available to GRT targets in R2012a (that is, you do not use the model option **Classic call interface**), you must update your static main program to be compatible with the R2012b static main changes. Use the code in *matlabroot*/rtw/c/src/common/rt_main.c as an example. The following sections outline some of the key changes to look for.

Error status handling

In R2012a, GRT targets using the simplified call interface handled stop simulation requests (during MAT-file logging or External mode simulation) differently from ERT targets using the simplified call interface:

- For ERT targets, a stop simulation request caused the error status to be set to Simulation finished. The main program (ert_main.c) treated

this error status as a non-error, while treating all other non-NULL status values as errors.

- For GRT targets, a stop simulation request did not cause the error status to be set (it remained NULL). The main program treated all non-NULL status values as errors.

Beginning in R2012b, the error status handling for GRT targets using the simplified call interface has been changed to match ERT targets using the simplified interface.

Unused macros

In R2012a, GRT targets using the simplified call interface generated macros differently from ERT targets using the simplified call interface:

- For ERT targets, the build process did not generate macros if they were not used in the generated code.
- For GRT targets, the build process unconditionally generated several macros that were not used in generated code.

Beginning in R2012b, the build process no longer unconditionally generates unused macros for GRT targets using the simplified call interface. The macros affected include:

- `rtm*` macros for accessing unused fields of the `rtModel` structure, such as `ModelPtrs`, `StepSize`, `ChildSfunction`, `TPtr`, and `TaskTime`
- `IsSampleHit`

Multitasking functions

In R2012a, GRT targets using the simplified call interface generated functions for multitasking differently from ERT targets using the simplified call interface:

- For ERT targets, the build process never generated the `rt_SimUpdateDiscreteEvents` function and, by default, never generated the `rate_monotonic_scheduler` function. (The `rate_monotonic_scheduler` function is for MathWorks® internal use only.)

- For GRT targets, the build process generated the functions `rt_SimUpdateDiscreteEvents` and `rate_monotonic_scheduler` for multitasking.

Beginning in R2012b, the build process no longer generates the multitasking functions `rt_SimUpdateDiscreteEvents` and `rate_monotonic_scheduler` for GRT targets using the simplified call interface.

Convenient packNGo dialog for packaging generated code and artifacts

R2012b adds model configuration parameters for packaging generated code and artifacts as part of a model build. The following new parameters are located on the **Code Generation** pane of the Configuration Parameters dialog box:

- **Package code and artifacts** (`PackageGeneratedCodeAndArtifacts`) — Specify whether to automatically package generated code and artifacts for relocation.
- **Zip file name** (`PackageName`) — Specify the name of the `.zip` file in which to package generated code and artifacts for relocation.

If you select **Package code and artifacts**, the build process runs the `packNGo` function after code generation to package generated code and artifacts for relocation. Selecting **Package code and artifacts** also enables the **Zip file name** parameter for specifying a `.zip` file name. The default file name is *`model`*`.zip`. (*`model`* represents the name of the top model for which code is being generated.)

For more information, see “Relocate Code to Another Development Environment”.

Reusable code for subsystems shared by referenced models

In R2012b, you can configure a subsystem that is shared across referenced models to generate code to the shared utilities folder. Code generation creates a standalone function in the shared utilities folder that can be called by the generated code of multiple referenced models.

To generate a single function for a reusable subsystem, the subsystem must be an active link to a subsystem in a library. For more information, see “Code Reuse For Subsystems Shared Across Referenced Models”.

Code generation for protected models for accelerated simulations and host targets

A protected model can include the generated code of the model. To create a protected model, right-click the referenced model and select **Subsystem & Model Reference > Create Protected Model for Selected Block** to open the Create Protected Model dialog box. You can select options that:

- Include the generated C code of the referenced model.
- Obfuscate the generated code.
- Create a protected model report.

You can then package the protected model, generated code, and protected model report for a third party to use for accelerated simulations and code generation. In R2012b, the file extension for protected models is `.slxp` (instead of the `.mdlp` extension in previous releases).

For more information, see “Protect a Referenced Model” and “Package a Protected Model”.

Reduction of data copies with buses and more efficient `for`-loops in generated code

Reduction of cyclomatic complexity with virtual bus expansion

In R2012b, code generation reduces cyclomatic complexity introduced by virtual bus expansion. This enhancement improves execution speed, reduces code size, and enables additional optimizations that reduce data copies and RAM consumption.

Simplifying `for` loop control statements

Improvements to `for` loops in the generated code include lifting invariance out of the `for` loop header and simplifying complex control statements in the `for` loop header. This enhancement improves execution speed and the readability of the generated code.

Unified rtiostream serial and TCP/IP target connectivity for all host platforms

Beginning in R2012b, Simulink Coder software provides unified `rtiostream` serial and TCP/IP target connectivity for all host platforms. Specifically, R2012b extends `rtiostream` serial connectivity to Linux® and Macintosh host platforms; previously, only Windows® host platforms were supported.

If you have implemented `rtiostream` serial connectivity for your embedded target environment, you can use `rtiostream` serial communication on any valid host to connect a Simulink model to your embedded target, using External mode or processor-in-the-loop (PIL) simulation.

Note Simulink Coder software provides `rtiostream` serial and TCP/IP target connectivity for all host platforms. If required, you can implement custom `rtiostream` connectivity—for example, to support a communication protocol other than serial or TCP/IP—for both the host platform and the embedded target environment.

Constant parameters generated as individual constants to shared location

Previously, constant parameters were generated to a model-specific structure, `rtConstP`, in the *`model`*`_data.c` file. If constant parameters are part of a model reference hierarchy or the model configuration parameter **Shared code placement** is set to `Shared location`, they are generated to a shared location. In R2012b, shared constant parameters are generated as individual constants to the `const_params.c` file in the`_sharedutils` folder. This code generation improvement generates less code and allows for subsystem code reuse across models. For more information, see “Shared Constant Parameters for Code Reuse”.

Code Efficiency Enhancements

The following code generation enhancements improve the efficiency of the generated code by:

- Removing a root-level outport data copy in the generated code when data is from a Stateflow® chart. This enhancement reduces RAM and ROM consumption and improves execution speed.
- Removing a data copy for masked subsystems when a parameter is a matrix data type. This enhancement reduces RAM and ROM consumption and improves execution speed.
- Removing a limitation where the joint presence of initial value and function prototype control prevent removal of the root-level outport data copy in the generated code. The outport data copy is removed when the initial value is zero. This enhancement reduces RAM and ROM consumption and improves execution speed.
- Removing an unnecessary global variable generated by a For Each Subsystem or as a result from the selected configuration parameter **Pack Boolean data into bitfields**. In R2012b, the variable is removed from the global block structure which reduces global RAM.

Optimized Code Generation of Delay Block

In R2011b, a new Delay block replaced the Integer Delay block. The Delay block now supports optimized code generation.

Search improvements in code generation report

Searching text in the code generation report highlights all results and then scrolls to the first result. Press **Enter** to scroll through the subsequent search results. If the search returns no results, the background of the search box is highlighted red.

GRT template makefile change for MAT-file logging support

Compatibility Considerations: Yes

In R2012b, the template makefiles (TMFs) for GRT-based targets have been updated to better support the **MAT-file logging** (`MatFileLogging`) model option, which was added to the **Interface** pane of the Configuration Parameters dialog box for GRT targets in R2010b.

Compatibility Considerations

If you authored a TMF for a GRT-based target, you should update your TMF to better support the **MAT-file logging** option. If **MAT-file logging** is selected for a GRT model, your existing TMF will continue to work. But if **MAT-file logging** is cleared, compilation of the model code will fail unless your TMF is updated.

To update your TMF, do the following:

1. Add a makefile variable token for MAT-file logging to the TMF:

```
MAT_FILE        = |>MAT_FILE<|
```

2. Use this variable to create a `-D` define that is part of the compiler invocation. For example

```
CPP_REQ_DEFINES = -DMODEL=$(MODEL) -DRT -DNUMST=$(NUMST) \
                        -DTID01EQ=$(TID01EQ) -DNCSTATES=$(NCSTATES) -DUNIX \
                        -DMT=$(MULTITASKING) -DHAVESTDIO -DMAT_FILE=$(MAT_FILE)
```

For examples of this update, see the GRT-based TMFs provided with Simulink Coder, located at *`matlabroot`*`/rtw/c/grt/grt_*.tmf`.

Change for blocks that use TLC custom code functions in multirate subsystems

Compatibility Considerations: Yes

In earlier releases, blocks could use the TLC functions `LibSystem*CustomCode` to register custom code to be placed inside the `gcd` rate of a multirate subsystem. Beginning in R2012b, blocks that register custom code for this purpose must additionally register use of custom code with the Simulink software, using the `SimStruct` macro `ssSetUsingTLCCustomCodeFunctions`. Registering allows the Simulink engine to perform necessary adjustments to handle multiple rates for subsystems with custom code. Code generation will generate an error if all of the following conditions are true:

- An S-function uses `LibSystem*CustomCode` functions without registering their use to Simulink.
- The S-function is placed in a multirate subsystem.
- No nonvirtual block in the subsystem has a sample time equal to the `gcd` of the sample times in the system.

Compatibility Considerations

If you authored a block that uses any of the TLC `LibSystem*CustomCode` functions to register custom code to be placed inside multirate subsystem functions, the block now must register custom code use with the Simulink software. Modify the `mdlInitializeSizes` code in the block to call the `ssSetUsingTLCCustomCodeFunctions` macro, as shown below:

```
ssSetUsingTLCCustomCodeFunctions (S, 1);
```

Model rtwdemo_f14 removed from software

Compatibility Considerations: Yes

In R2012b, the example model `rtwdemo_f14` has been removed from the Simulink Coder software.

Compatibility Considerations

If you need an example model with similar content, open the Simulink example model `sldemo_f14` and configure it with a fixed-step solver. If you need an example GRT model that is configured for code generation, see the Simulink Coder models in the `rtwdemos` list.

Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

Search R2012b Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2012b&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2012b&product=RT`

R2012a

Version: 8.2
New Features: Yes
Bug Fixes: Yes

Simplified Call Interface for Generated Code

In previous releases, GRT and GRT-based targets generated code with a GRT-specific call interface, using the model entry functions `model`, `MdlInitializeSizes`, `MdlInitializeSampleTimes`, `MdlStart`, `MdlOutputs`, `MdlUpdate`, and `MdlTerminate`. ERT and ERT derived targets, by default, generated code with a simplified call interface, using the model entry functions `model_initialize`, `model_step`, and `model_terminate`. (Additionally, model options could be applied to customize the simplified call interface, such as clearing **Single output/update function** or **Terminate function required**.)

In R2012a, GRT targets can now generate code with the same simplified call interface as ERT targets. This simplifies the task of interacting with the generated code. Target authors can author simpler `main.c` or `.cpp` programs for GRT targets. Also, it is no longer required to author different main programs for GRT and ERT targets.

To preserve compatibility with models, GRT-based custom targets, and GRT main modules created in earlier releases, R2012a provides the model option **Classic call interface** (`GRTInterface`), which is located on the **Code Generation > Interface** pane of the Configuration Parameters dialog box. If you select **Classic call interface**, code generation generates model function calls compatible with the main program module of the GRT target in models created before R2012a. If you clear the **Classic call interface** option, code generation generates the simplified call interface.

Note

- The **Classic call interface** (`GRTInterface`) option is available for both GRT-based and ERT-based models. For Embedded Coder users, it replaces the ERT model option **GRT-compatible call interface** (`GRTInterface`).
- For new GRT and ERT models, the **Classic call interface** option is cleared by default. New models use the simplified call interface.
- For GRT models created before R2012a, **Classic call interface** is selected by default. Existing GRT models can continue to use the pre-R2012a GRT-specific call interface.

Incremental Code Generation for Top-Level Models

R2012a provides the ability to omit unnecessary code regeneration from top model builds, allowing top models to be built incrementally. This can significantly reduce model build times.

Previously, each model build fully regenerated and compiled the top model code. Beginning in R2012a, the build process checks the structural checksum of the model to determine whether changes to the model require code regeneration. If code regeneration is required, the build process fully regenerates and compiles the model code, in the manner of earlier releases. However, if the generated code is found to be current with respect to the model, the build process does the following:

1. Skips model code regeneration.
2. Still calls build process hooks, including `STF_make_rtw_hook` functions and the post code generation command.
3. Reruns the makefile to make sure external dependencies are recompiled and relinked.

Additionally, command-line options exist for controlling or overriding the new build behavior. For more information, see Control Regeneration of Top Model Code.

Minimal Header File Dependencies with packNGo Function

The `packNGo` function, which packages model code files in a zip file for relocation, now by default includes only the minimal header files required in the zip file. The `packNGo` function now runs a preprocessor to determine the minimal header files required to build the code. Previously, `packNGo` included all header files found on the include path.

To revert to the behavior of previous releases, you can use the following form of the function:

```
>> packNGo(buildInfo,{'minimalHeaders',false})
```

ASAP2 Enhancements for Model Referencing and Structured Data

Ability to Merge ASAP2 Files Generated for Top and Referenced Models

R2012a provides the ability to merge ASAP2 files generated for top and referenced models into a single ASAP2 file. To merge ASAP2 files for a given model, use the function `rtw.asap2MergeMdlRefs`, which has the following syntax:

```
[status,info]=rtw.asap2MergeMdlRefs(topModelName,asap2FileName)
```

For more information, see Merge ASAP2 Files Generated for Top and Referenced Models

ASAP2 File Generation for Test Pointed Signals and States

ASAP2 file generation has been enhanced to generate ASAP2 MEASUREMENT records for the following data, without the need to resolve them to Simulink data objects:

- Test-pointed Simulink signals, usable inside reusable subsystems
- Test pointed Stateflow states, allowing you to monitor which state is active during real-time testing
- Test-pointed Stateflow local data
- Root-level inports and outports

Options to control ASAP2 record generation for structured data are defined in *matlabroot*`/toolbox/rtw/targets/asap2/asap2/user/asap2setup.tlc`:

- `ASAP2EnableTestPoints` enables or disables record generation for test pointed Simulink signals, test pointed Stateflow states, and test-pointed Stateflow local data (enabled by default)
- `ASAP2EnableRootLevelIO` enables or disables record generation for root-level inports and outports (disabled by default)

For more information, see Customize an ASAP2 File.

ASAP2 File Generation for Tunable Structure Parameters

ASAP2 file generation has been enhanced to generate ASAP2 CHARACTERISTIC records for tunable structure parameters. This allows you to tune structure parameters with ASAP2 tools and potentially manage large parameter sets.

For more information, see Customize an ASAP2 File.

Serial External Mode Communication Using `rtiostream` API

In R2012a, you can create a serial transport layer for Simulink external mode communication using the `rtiostream` API. For more information, see Create a Transport Layer for External Communication.

Improved Data Transfer in External Mode Communication

In Simulink External mode communication, the `rt_OneStep` function runs in the foreground and the `while` loop of the `main` function runs in the background. See Real-Time Single-Tasking Systems. Previously, with code generated for GRT and Embedded Coder bareboard ERT targets, data transfer between host and server was performed by functions within the model step function in `rt_OneStep`. The data transfer between host and server (in the foreground) would slow down model execution, potentially impairing real-time performance.

Now, the function that is responsible for data transfer between host and server (`rtExtModeOneStep`) is inserted in the `while` loop of the `main` function. As the execution of the `while` loop in the `main` function is a background task, real-time performance potentially is enhanced.

Changes for Desktop IDEs and Desktop Targets

Support Added for GCC 4.4 on Host Computers Running Linux with Eclipse IDE

Simulink Coder software now supports GCC 4.4 on host computers running Linux with Eclipse™ IDE. This support is on both 32-bit and 64-bit host Linux platforms.

If you were using an earlier version of GCC on Linux with Eclipse, upgrade to GCC 4.4.

Limitation: Parallel Builds Not Supported for Desktop Targets

The Simulink Coder product provides an API for MATLAB® Distributed Computing Server™ and Parallel Computing Toolbox™ products. The API allows these products to perform parallel builds that reduce build time for referenced models. However, the API does not support parallel builds for models whose **System target file** parameter is set to `idelink_ert.tlc` or `idelink_grt.tlc`. Thus, you cannot perform parallel builds for Desktop Targets.

Code Generation Report Enhancements

Post-build Report Generation

In previous releases, if you did not configure your model to create a code generation report, you had to build your model again to open the code generation report. You can now generate a code generation report after the code generation process completes without building your model again. This option is available on the model diagram **Tools** menu. After building your model, select **Tools > Code Generation > Code Generation Report > Open Model Report**. You can also open a code generation report after building a subsystem. For more information on creating and opening the code generation report, see Generate an HTML Code Generation Report.

Generate Code Generation Report Programmatically

At the MATLAB command line, you can generate, open, and close an HTML Code Generation Report with the following functions:

- `coder.report.generate` generates the code generation report for the specified model.
- `coder.report.open` opens an existing code generation report.
- `coder.report.close` closes the code generation report.

Searching in the Code Generation Report

You can now search within the code generation report using a search box in the navigation section. After entering text in the search box, the current page scrolls to the first match and highlights all of the matches on the page. To access the **Search** text box, press **Ctrl-F**.

New Reserved Keywords for Code Generation

The Simulink Coder software includes the following reserved keywords to the Simulink Coder Code Generation keywords list. For more information, see Reserved Keywords.

ERT	LINK_DATA_STREAM	NUMST	RT_MALLOC
HAVESTDIO	MODEL	PROFILING_ENABLED	TID01EQ
INTEGER_CODE	MT	PROFILING_NUM_SAMPLES	USE_RTMODEL
LINK_DATA_BUFFER_SIZE	NCSTATES	RT	VCAST_FLUSH_DATA

Improved MAT-File Logging

R2012a enhances Simulink Coder MAT-file logging to allow logging of multiple data points per time step, by reallocating buffer memory during target execution. Generated code logging results now match simulation results for blocks executing multiple times per step, such as blocks in an iterator subsystem. Previously, code generation issued a warning that the logged results for blocks executing in an iterator subsystem might not match the results from simulation.

rtwdemo_f14 Being Removed in a Future Release

Compatibility Considerations: Yes

The demo model `rtwdemo_f14` will be removed in a future release of Simulink Coder software.

Compatibility Considerations

In R2012a, you can still open `rtwdemo_f14` by entering `rtwdemo_f14` in the MATLAB Command Window. Going forward, transition to using `f14`, `sldemo_f14`, or a Simulink Coder model in the `rtwdemos` list.

New and Enhanced Demos

The following demos have been enhanced in R2012a:

Demo...	Now...
`rtwdemo_asap2`	• Illustrates ASAP2 file generation for test pointed signals and states. • Shows how to generate a single ASAP2 file from files for top and referenced models. • Generates STD_AXIS and FIX_AXIS descriptions for lookup table breakpoints.
`rtwdemo_configuration_set`	Shows how to use the Code Generation Advisor and the `Simulink.ConfigSet` saveAs method.

Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

Search R2012a Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2012a&keyword=Incorrect+Code+Generation`

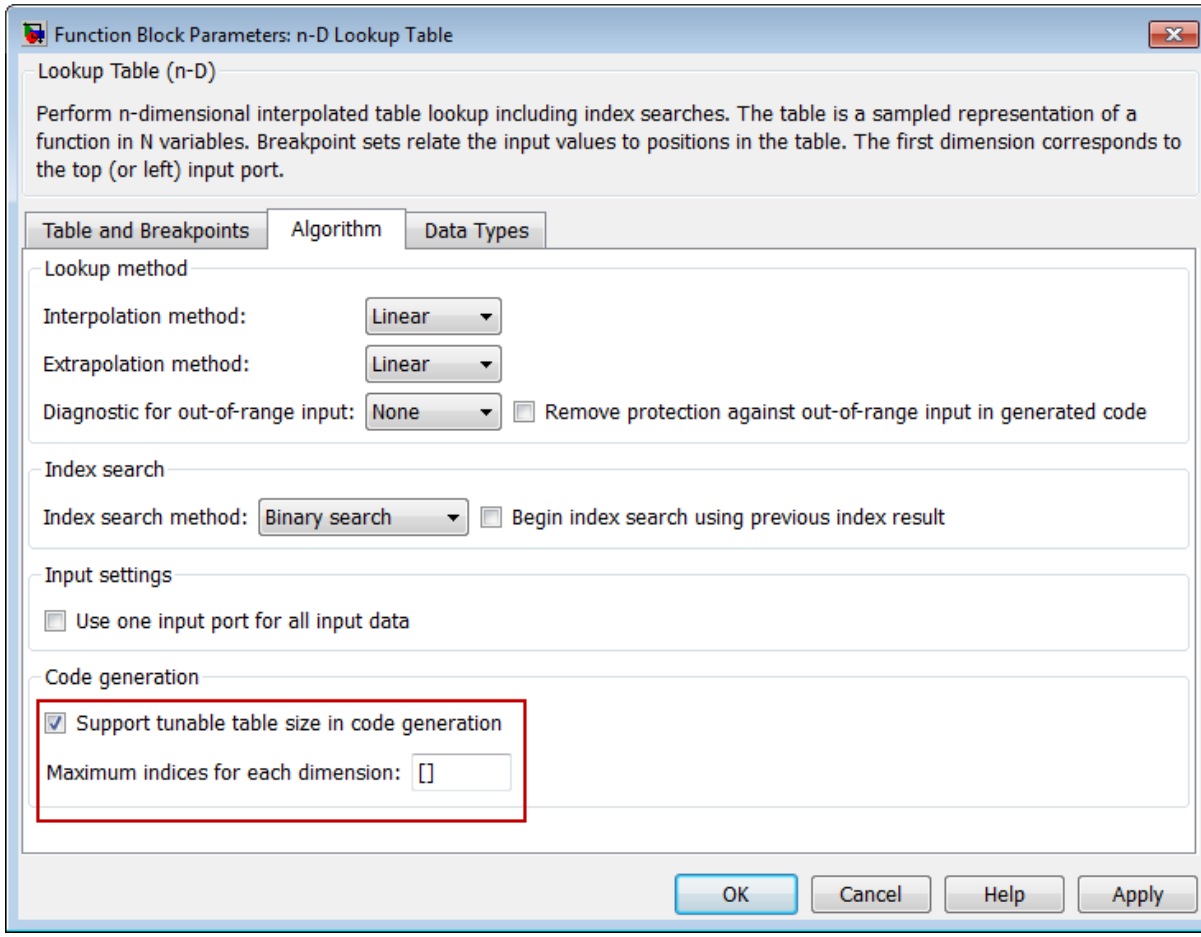
All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2012a&product=RT`

R2011b

Version: 8.1
New Features: Yes
Bug Fixes: Yes

n-D Lookup Table Block Supports Tunable Table Size

The n-D Lookup Table block provides new parameters for specifying a tunable table size in the generated code.



This enhancement enables you to change the size and values of your lookup table and breakpoint data without regenerating or recompiling the code.

Complex Output Support in Generated Code for the Trigonometric Function Block

In previous releases, the imaginary part of a complex output signal was always zero in the generated code for the Trigonometric Function block. In R2011b, this limitation no longer exists. Code that you generate for a function in this block now supports complex outputs.

Code Optimizations for the Combinatorial Logic Block

The Simulink Coder build process uses a new technique to provide more efficient code for the Combinatorial Logic block.

Benefits include:

- Reuse of variables
- Dead code elimination
- Constant folding
- Expression folding

For example, in previous releases, temporary buffers were created to carry concatenated signals for this block. In R2011b, the build process eliminates unnecessary temporary buffers and writes the concatenated signal to the downstream global buffer directly. This enhancement reduces the stack size and improves code execution speed.

Code Optimizations for the Product Block

Compatibility Considerations: Yes

The Simulink Coder build process provides more efficient code for matrix inverse and division operations in the Product block. The following summary describes the benefits and when each benefit is available:

Benefit	Small matrices (2-by-2 to 5-by-5)	Medium matrices (6-by-6 to 20-by-20)	Large matrices (larger than 20-by-20)
Faster code execution time	Yes, much faster	No, slightly slower	Yes, faster
Reduced ROM and RAM usage	Yes, for real values	Yes, for real values	Yes, for real values
Reuse of variables	Yes	Yes	Yes
Dead code elimination	Yes	Yes	Yes
Constant folding	Yes	Yes	Yes
Expression folding	Yes	Yes	Yes
Consistency with MATLAB Coder	Yes	Yes	Yes

Compatibility Considerations

In the following cases, the generated code might regress from previous releases:

- The ROM and RAM usage increase for complex input data types.
- For blocks configured with 3 or more inputs of different dimensions, the code might include an extra buffer to store temporary variables for intermediate results.

Enhanced MISRA C Code Generation Support for Stateflow Charts

In previous releases, the code generated to check whether or not a state in a Stateflow chart was active included a line that looked something like this:

```
if (mdl_state_check_er_DWork.is_active_c1_mdl_state_c == 0)
```

In R2011b, that line has been modified to:

```
if (mdl_state_check_er_DWork.is_active_c1_mdl_state_c == 0U)
```

This enhancement supports MISRA C® 2004, rule 10.1.

Change for Constant Sample Time Signals in Generated Code

Compatibility Considerations: Yes

In previous releases, constant sample time signals were initialized even if the **Data Initialization** field of their custom storage class was set to `None`.

In R2011b, constant sample time signals using a custom storage class for which the **Data Initialization** field is set to `None` will not be initialized for non-conditionally executed systems in generated code.

Compatibility Considerations

If you use such constant time signals, you will notice that they are not initialized in the generated code in R2011b. To enable their initialization, change the setting of the **Data Initialization** field of their custom storage class from `None` to another value.

New Code Generation Advisor Objective for GRT Targets

In R2011b, `Execution efficiency` is now available as a Code Generation Advisor objective for models with generic real-time (GRT) targets. You can use this objective to achieve faster code execution times for your models. For more information, see Application Objectives.

Improved Integer and Fixed-Point Saturating Cast

Simulink Coder software now eliminates more dead branches in both integer and fixed-point saturation code.

Generate Multitasking Code for Concurrent Execution on Multicore Processors

The Simulink Coder product extends the concurrent execution modeling capability of the Simulink product. With Simulink Coder, you can generate multitasking code that uses POSIX threads (Pthreads) or Windows threads for concurrent execution on multicore processors running Linux, Mac OS X, or Windows.

See Configuring Models for Targets with Multicore Processors.

Changes for Desktop IDEs and Desktop Targets

New Target Function Library for Intel IPP/SSE (GNU)

This release adds a new Target Function Library (TFL), `Intel IPP/SSE (GNU)`, for the GCC compiler. This library includes the Intel Performance Primitives (IPP) and Streaming SIMD Extensions (SSE) code replacements.

Support Added for Single Instruction Multiple Data (SIMD) with Intel Processors

This release adds support for the SIMD capabilities of the Intel® processors. The use of SIMD instructions increases throughput compared to traditional Single Instruction Single Data (SISD) processing.

The `Intel IPP/SSE (GNU)` TFL (code replacement library) optimizes generated code for SIMD.

The performance of the SIMD-enabled executable depends on several factors, including:

- Processor architecture of the target
- Optimized library support for the target
- The type and number of TFL replacements in the generated algorithmic code

Evaluate the performance of your application before and after using the TFL.

To use the SIMD capabilities with GCC and Intel processors, enable the `Intel IPP/SSE (GNU)` TFL. See Code Replacement Library (CRL).

Reserved Keyword `UNUSED_PARAMETER`

The Simulink Coder software adds the `UNUSED_PARAMETER` macro to the reserved keywords list for code generation. To view the complete list, see Reserved Keywords. In R2011b, code generation now defines `UNUSED_PARAMETER` in `rt_defines.h`. Previously, it was defined in *`model`*`_private.h`.

Target API for Verifying MATLAB® Distributed Computing Server™ (MDCS) Worker Configuration for Parallel Builds

R2010b added the ability to use remote workers in MDCS configurations for parallel builds of model reference hierarchies. This introduced the possibility that parallel workers might have different configurations, some of which might not be compatible with a specific Simulink Coder target build. For example, the required compiler might not be installed on a worker system.

R2011b provides a programming interface that target authors can use to automatically check the configuration of parallel workers and, if the parallel workers are not set up as required, take action, such as throwing an error or reverting to sequential builds. For more information, see Support Model Referencing in the Simulink Coder documentation.

For more information about parallel builds, see Reduce Build Time for Referenced Models in the Simulink Coder documentation.

License Names Not Yet Updated for Coder Product Restructuring

The Simulink Coder and Embedded Coder license name strings stored in `license.dat` and returned by the `license ('inuse')` function have not yet been updated for the R2011a coder product restructuring. Specifically, the `license ('inuse')` function continues to return `'real-time_workshop'` for Simulink Coder and `'rtw_embedded_coder'` for Embedded Coder, as shown below:

```
>> license('inuse')
matlab
matlab_coder
real-time_workshop
rtw_embedded_coder
simulink
>>
```

The license name strings intentionally were not changed, in order to avoid license management complications in situations where Release 2011a or higher is used alongside a preR2011a release in a common operating environment. MathWorks plans to address this issue in a future release.

For more information about using the function, see the `license` documentation.

New and Enhanced Demos

The following demos have been enhanced in R2011b:

Demo...	Now...
`rtwdemo_pmsmfoc_script`	Shows how you can perform system-level simulation and algorithmic code generation using Field-Oriented Control for a Permanent Magnet Synchronous Machine

Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

Search R2011b Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2011b&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2011b&product=RT`

R2011a

Version: 8.0
New Features: Yes
Bug Fixes: Yes

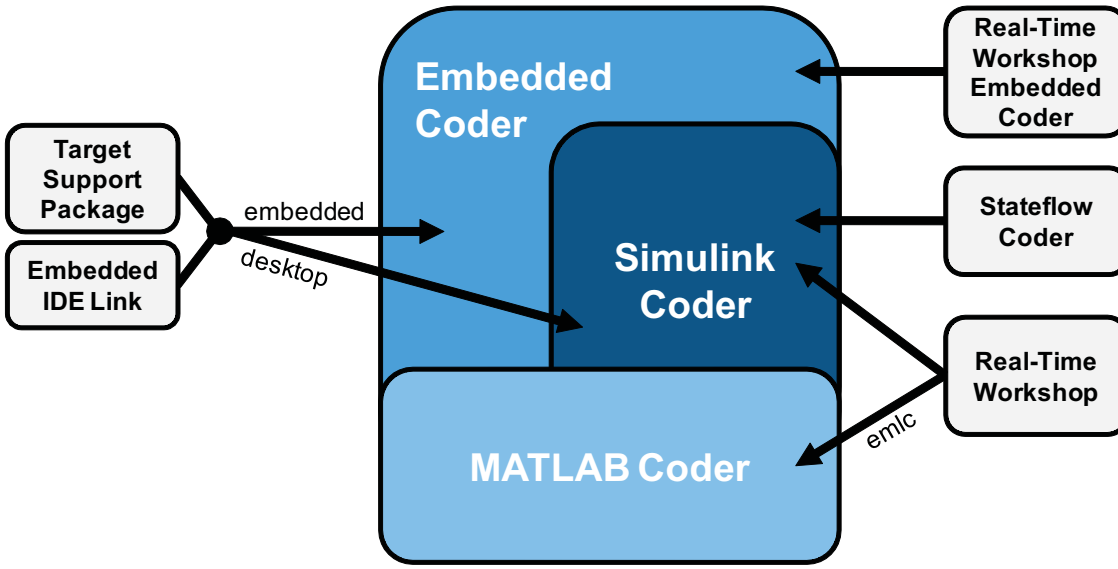
Coder Product Restructuring

Product Restructuring Overview

In R2011a, the Simulink Coder product combines and replaces the Real-Time Workshop® and Stateflow® Coder™ products. Additionally,

- The Real-Time Workshop facility for converting MATLAB code to C/C++ code, formerly referred to as Embedded MATLAB® Coder, has migrated to the new MATLAB Coder product.
- The previously existing products Embedded IDE Link™ and Target Support Package™ have been integrated into the new Simulink Coder and Embedded Coder products.

The following figure shows the R2011a transitions for C/C++ code generation related products, from the R2010b products to the new MATLAB Coder, Simulink Coder, and Embedded Coder products.



The following sections address topics related to the product restructuring.

Resources for Upgrading from Real-Time Workshop or Stateflow Coder

If you are upgrading to Simulink Coder from Real-Time Workshop or Stateflow Coder, review information about compatibility and upgrade issues at the following locations:

- *Release Notes for Simulink Coder* (latest release), "Compatibility Summary" section
- In the Archived documentation on the MathWorks web site, select R2010b, and view the following tables, which are provided in the release notes for Real-Time Workshop and Stateflow Coder:
  - *Compatibility Summary for Real-Time Workshop Software*
  - *Compatibility Summary for Stateflow and Stateflow Coder Software*

  These tables provide compatibility information for releases up through R2010b.

- If you use the Embedded IDE Link or Target Support Package capabilities that now are integrated into Simulink Coder and Embedded Coder, go to the Archived documentation, select R2010b, and view the corresponding tables for each product:
  - *Compatibility Summary for Embedded IDE Link*
  - *Compatibility Summary for Target Support Package*

You can also refer to the rest of the archived documentation, including release notes, for the Real-Time Workshop, Stateflow Coder, Embedded IDE Link, and Target Support Package products.

Migration of Embedded MATLAB Coder Features to MATLAB Coder

In R2011a, the MATLAB Coder function `codegen` replaces the Real-Time Workshop function `emlc`. The `emlc` function still works in R2011a but generates a warning, and will be removed in a future release. For more information, see Migrating from Real-Time Workshop emlc Function in the MATLAB Coder release notes.

Migration of Embedded IDE Link and Target Support Package Features to Simulink Coder and Embedded Coder

In R2011a, the capabilities formerly provided by the Embedded IDE Link and Target Support Package products have been integrated into Simulink Coder and Embedded Coder. The follow table summarizes the transition of the Embedded IDE Link and Target Support Package hardware and software support into coder products.

Former Product	Supported Hardware and Software	Simulink Coder	Embedded Coder
Embedded IDE Link	Altium® TASKING		x
	Analog Devices™ VisualDSP++®		x
	Eclipse IDE	x	x
	Green Hills® MULTI®		x
	Texas Instruments™ Code Composer Studio™		x
Target Support Package	Analog Devices Blackfin®		x
	ARM®		x
	Freescale™ MPC5xx		x
	Infineon® C166®		x
	Texas Instruments C2000™		x
	Texas Instruments C5000™		x
	Texas Instruments C6000™		x
	Linux OS	x	x
	Windows OS	x	
	VxWorks® RTOS		x

User Interface Changes Related to Product Restructuring

Some user interface changes were made as part of merging the Real-Time Workshop and Stateflow Coder products into Simulink Coder. They include:

- Changes to code generation related elements in the Simulink Configuration Parameters dialog box

- Changes to code generation related elements in Simulink menus
- Changes to code generation related elements in Simulink blocks, including block parameters and dialog boxes, and block libraries
- References to Real-Time Workshop and Stateflow Coder and related terms in error messages, tool tips, demos, and product documentation replaced with references to the latest software

Simulink Graphical User Interface Changes

Where...	Previously...	Now...
Configuration Parameters dialog box	**Real-Time Workshop** pane	**Code Generation** pane
Model diagram window	**Tools > Real-Time Workshop**	**Tools > Code Generation**
Subsystem context menu	**Real-Time Workshop**	**Code Generation**
Subsystem Parameters dialog box	Following parameters on main pane: • **Real-Time Workshop system code** • **Real-Time Workshop function name options** • **Real-Time Workshop function name** • **Real-Time Workshop file name options** • **Real-Time Workshop**	On new **Code Generation** pane and renamed: • **Function packaging** • **Function name options** • **Function name** • **File name options** • **File name (no extension)**

Where...	Previously...	Now...
	file name (no extension)	

Changes for Desktop IDEs and Desktop Targets

Feature Support for Desktop IDEs and Desktop Targets

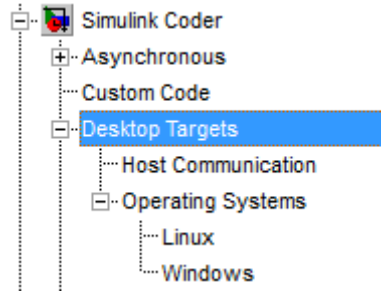
The Simulink Coder software provides the following features as implemented in the former Target Support Package and Embedded IDE Link products:

- Automation Interface
- External Mode
- Multicore Deployment with Rate Based Multithreading
- Makefile Generation (XMakefile)

Note You can only use these features in the 32-bit version of your MathWorks products. To use these features on 64-bit hardware, install and run the 32-bit versions of your MathWorks products.

Location of Blocks for Desktop Targets

Blocks from the former Target Support Package product and Embedded IDE Link product are now located in Simulink Coder under Desktop Targets.

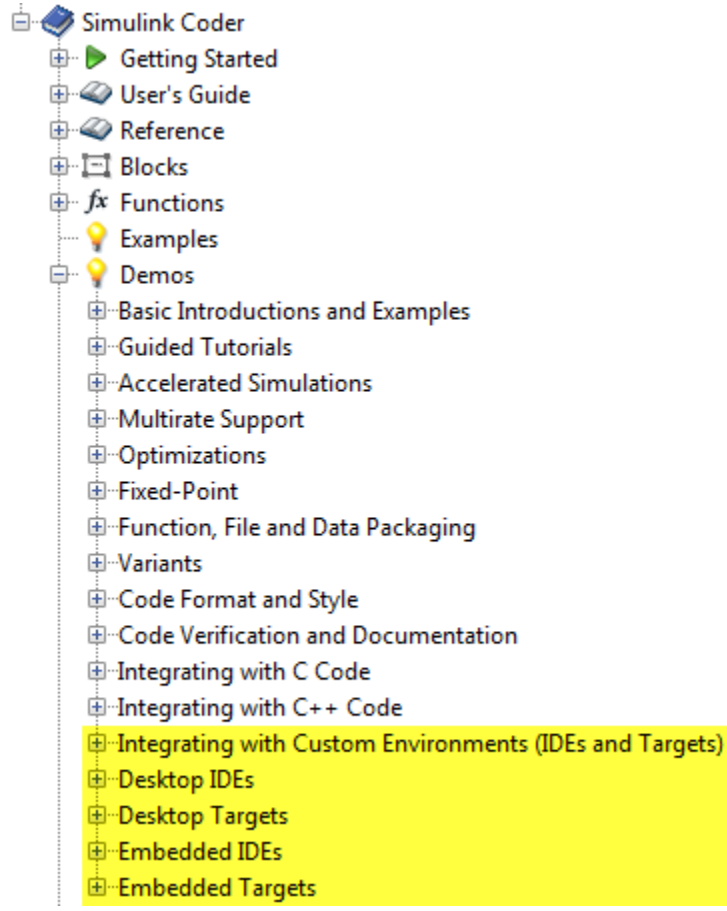


Desktop Targets includes the following types of blocks:

- Host Communication
- Operating Systems
  - Linux
  - Windows

Location of Demos for Desktop IDEs and Desktop Targets

Demos from the former Target Support Package product and Embedded IDE Link product now reside under Simulink Coder product help. Click the expandable links, as shown.



Multicore Deployment with Rate Based Multithreading

You can deploy rate-based multithreading applications to multicore processors running Windows and Linux. This feature potentially improves performance by taking advantage of multicore hardware resources.

Also see the Running Target Applications on Multicore Processors user's guide topic.

Code Optimizations for Discrete State-Space Block, Product Block, and MinMax Block

The Simulink Coder build process uses a new technique to provide more efficient code for the following blocks:

- Discrete State-Space
- Product (element-wise matrix operations)

Benefits include:

- Reuse of variables
- Dead code elimination
- Constant folding
- Expression folding

For example, in previous releases, temporary buffers were created to carry concatenated signals for these blocks. In R2011a, the build process eliminates unnecessary temporary buffers and writes the concatenated signal to the downstream global buffer directly. This enhancement reduces the stack size and improves code execution speed.

The build process also provides more efficient code for the MinMax block. In R2011a, expression folding is enhanced with several local optimizations that enable more aggressive folding. This enhancement improves code efficiency for foldable matrix operations.

Ability to Share User-Defined Data Types Across Models

In previous releases, user-defined data types that were shared among multiple models generated duplicate type definitions in the *model*_types.h file for each model. R2011a provides the ability to generate user-defined data type definitions into a header file that can be shared across multiple models, removing the need for duplicate copies of the data type definitions. User-defined data types that you can set up in a shared header file include:

- Simulink data type objects that you instantiate from the classes `Simulink.AliasType`, `Simulink.Bus`, `Simulink.NumericType`, and `Simulink.StructType`
- Enumeration types that you define in MATLAB code

For more information, see Share User-Defined Data Types Across Models in the Simulink Coder documentation.

C API Provides Access to Root-Level Inputs and Outputs

The C API now provides programmatic access to root-level inputs and outputs. This allows you to log and monitor the root-level inputs and outputs of a model while you run the code generated for the model. To generate C API code for accessing root-level inputs and outputs at run time, select the model option **Generate C API for: root-level I/O**.

Macros for accessing C API generated structures are located in *matlabroot*`/rtw/c/src/rtw_capi.h` and *matlabroot*`/rtw/c/src/rtw_modelmap.h`, where *matlabroot* represents your MATLAB installation root.

For more information, see Generate C API for: root-level I/O and Data Interchange Using the C API in the Simulink Coder documentation.

ASAP2 File Generation Supports Standard Axis Format for Lookup Tables

In previous releases, ASAP2 file generation for lookup table blocks supported the Fix Axis and Common Axis formats, but not the Standard Axis format, a format in which axis points are global in code but not shared among tables. R2011a adds support for Standard Axis format.

For more information, see Define ASAP2 Information for Lookup Tables in the Simulink Coder documentation.

ASAP2 File Generation Enhancements for Computation Methods

Custom Names for Computation Methods

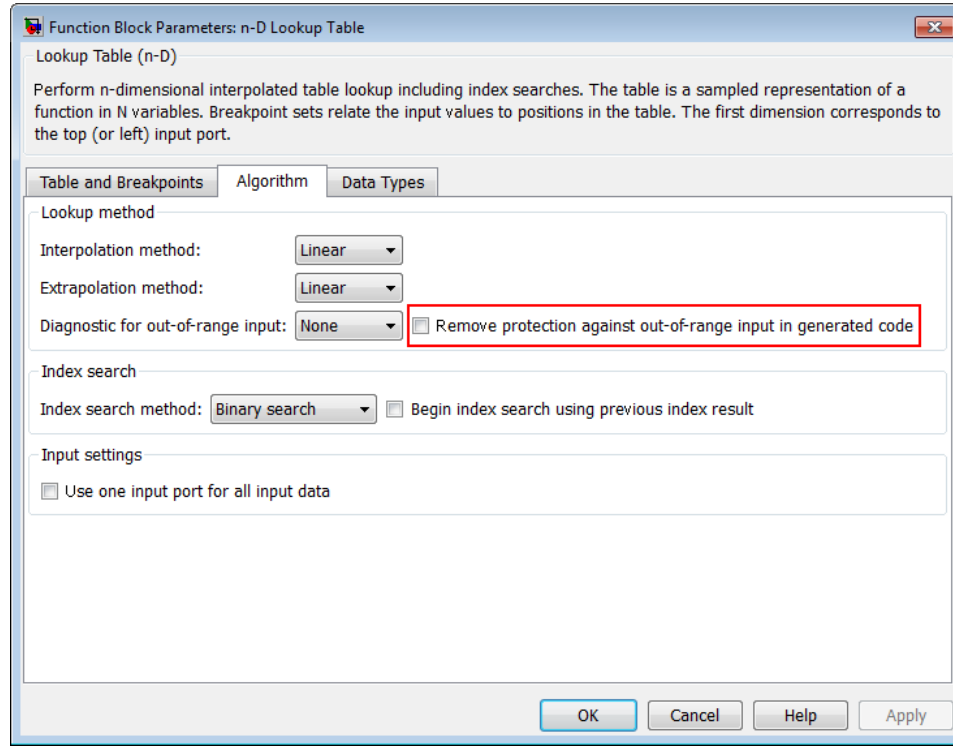
In generated ASAP2 files, computation methods translate the electronic control unit (ECU) internal representation of measurement and calibration quantities into a physical model oriented representation. R2011a adds the MATLAB function `getCompuMethodName`, which you can use to customize the names of computation methods. You can provide names that are more intuitive, enhancing ASAP2 file readability, or names that meet organizational requirements. For more information, see Customize Computation Method Names in the Simulink Coder documentation.

Ability to Suppress Computation Methods for FIX_AXIS When Not Required

Versions 1.51 and later of the ASAP2 specification state that for certain cases of lookup table axis descriptions (integer data type and no doc units), a computation method is not required and the Conversion Method parameter must be set to the value `NO_COMPU_METHOD`. Beginning in R2011a, you can control whether or not computation methods are suppressed when not required, using the Target Language Compiler (TLC) option `ASAP2GenNoCompuMethod`. For more information, see Suppress Computation Methods for FIX_AXIS in the Simulink Coder documentation.

Lookup Table Block Option to Remove Input Range Checks in Generated Code

When the breakpoint input to a Prelookup, 1-D Lookup Table, 2-D Lookup Table, or n-D Lookup Table block always falls within the range of valid breakpoint values, you can disable range checking in the generated code. By selecting **Remove protection against out-of-range input in generated code** on the block dialog box, your code can be more efficient.



Reentrant Code Generation for Stateflow Charts That Use Events

When you generate code for Stateflow charts that use events, the code does not use a global variable to keep track of the currently active event. Elimination of this global variable enables the code to be reentrant, which allows you to:

- Deploy your code in multithreading environments
- Share the same algorithm with different persistent data
- Compile code that uses function variables that are too large to fit on the stack

In previous releases, reentrant code generation was not possible for charts that used events.

Redundant Check Code Removed for Stateflow Charts That Use Temporal Operators

When you generate code for Stateflow charts that use temporal operators, the code excludes redundant checks for `tick` events and input events that are always true. This enhancement enables the code to be more efficient and applies to temporal operators `after`, `before`, `at`, `every`, and `temporalCount`.

In previous releases, the code generated for a temporal logic expression such as `after(x,tick)` would check for two conditions:

```
(event == tick) && (counter > x)
```

In R2011a, the code generated for `after(x,tick)` checks only for when the temporal counter exceeds x:

```
(counter > x)
```

This enhancement does not apply when a chart with multiple input events has super-step semantics enabled.

Support for Multiple Asynchronous Function Calls Into a Model Block

Simulink and Simulink Coder software now support multiple asynchronous function calls into a Model block. This capability relies in part on the new Asynchronous Task Specification block.

The Asynchronous Task Specification block, in combination with a root-level Inport block, allows you to specify an asynchronous function-call input to a Model block. After placing this block at the output port of each root-level Inport block that outputs a function-call signal, select **Output function call** on the **Signal Attributes** pane of the Inport block. The Inport block then accepts function-call signals. You can use Asynchronous Task Specification blocks to specify parameters for the asynchronous task associated with the respective Inport blocks.

Note Use the new function call API, `LibBlockExecuteFcnCall`, to make function calls from an asynchronous source block to reference model destination blocks.

Note The demo model `rtwdemo_async_mdlreftop` shows how you can simulate and generate code for asynchronous events on a real-time multitasking system, using asynchronous function calls as Model block inputs.

Changes to ver Function Product Arguments

Compatibility Considerations: Yes

The following changes have been made to `ver` function arguments related to code generation products:

- The new argument `'simulinkcoder'` returns information about the installed version of the Simulink Coder product.
- The argument `'rtw'` works but now returns information about Simulink Coder instead of Real-Time Workshop. The software also displays the following message:

  ```
  Warning: Support for ver('rtw') will be removed in a future release.
  Use ver('simulinkcoder') instead.
  ```

- The argument `'coder'`, which previously returned information about the installed version of the Stateflow Coder product, no longer works. The software displays a "not found" warning.

For more information about using the function, see the `ver` documentation.

Compatibility Considerations

If a script calls the `ver` function with the `'rtw'` argument or the `'coder'` argument, update the script appropriately. For example, you can update the `ver` call to use the `'simulinkcoder'` argument, or remove the `ver` call.

Updates to Target Language Compiler (TLC) Semantics and Diagnostic Information

Updates to TLC simplifies semantics and produces diagnostic information when using the scope resolution operator (`::`) and built-in function `EXISTS(::)`.

- If *`var`* can not be resolved in global scope, `::`*`var`* errors out
- If *`var`* can only be resolved in local scope, `EXISTS(::`*`var`*`)` returns false
- Diagnostic information highlights problematic TLC coding

For more information, see Introducing the Target Language Compiler.

Change to Terminate Function for a Target Language Compiler (TLC) Block Implementation

Previously, the code generator attempted to execute the `Terminate` function from the TLC implementation of a block, even if the function did not exist. Now, the code generator only attempts to execute a `Terminate` function if it is defined in the TLC implementation of a block. In the case where the TLC implementation of a block includes a secondary TLC file, which includes a `Terminate` function, that `Terminate` function no longer executes.

New and Enhanced Demos

The following demos have been added in R2011a:

Demo...	Shows How You Can...
`rtwdemo_async_mdlreftop`	Simulate and generate code for asynchronous events on a real-time multitasking system, using asynchronous function calls as Model block inputs.

The following demos have been enhanced in R2011a:

Demo...	Now...
`vipstabilize_fixpt_beagleboard` `videostabilization_host_templ`	Use the new Video Capture block to simulate or capture a video input signal in the Video Stabilization demo.

Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

Search R2011a Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2011a&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2011a&product=RT`