Embedded Coder™ Release Notes

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.

Embedded Coder™ Release Notes

Contents

Bug Reports 1

Summary by Version 2

Version 6.0 (R2011a) Embedded Coder Software 4

Compatibility Summary for Embedded Coder Software 35

Bug Reports

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 http://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.

Summary by Version

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

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Latest Version V6.0 (R2011a)	Yes Details	Yes Summary	Bug Reports Includes fixes

Using Release Notes

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

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

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

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

What Is in the Release Notes

New Features and Changes

- New functionality
- Changes to existing functionality

Version Compatibility Considerations

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

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

Fixed Bugs and Known Problems

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

Access Bug Reports using your MathWorks Account.

Documentation on the MathWorks Web Site

Related documentation is available on `mathworks.com` for the latest release and for previous releases:

- Latest product documentation
- Archived documentation

Version 6.0 (R2011a) Embedded Coder Software

This table summarizes what is new in Version 6.0 (R2011a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary	Bug Reports Includes fixes

New features and changes introduced in this version are:

- "Coder Product Restructuring" on page 5
- "Data Management Enhancements and Changes" on page 10
- "AUTOSAR Enhancements" on page 13
- "SIL and PIL Enhancements" on page 15
- "Code Generation Enhancements" on page 16
- "Code Generation Verification (CGV) API Updates" on page 17
- "MISRA-C Code Generation Objective" on page 21
- "New Model Advisor Check for Code Efficiency of Lookup Table Blocks" on page 22
- "Enhanced Code Generation Optimization" on page 22
- "Target Function Library Replacement Based on Computation Method for Reciprocal Sqrt, Sine, and Cosine" on page 23
- "C++ Encapsulation Allowed for Referenced Models in For Each Subsystems" on page 23
- "Improved Code Generation for Portable Word Sizes" on page 24
- "Improved Comments in the Generated Code" on page 24
- "Replacement Data Types and Simulation Mode for Referenced Models" on page 24
- "Changes for Embedded IDEs and Embedded Targets" on page 25

- "Changes to ver Function Product Arguments" on page 33
- "New and Enhanced Demos" on page 34

Coder Product Restructuring

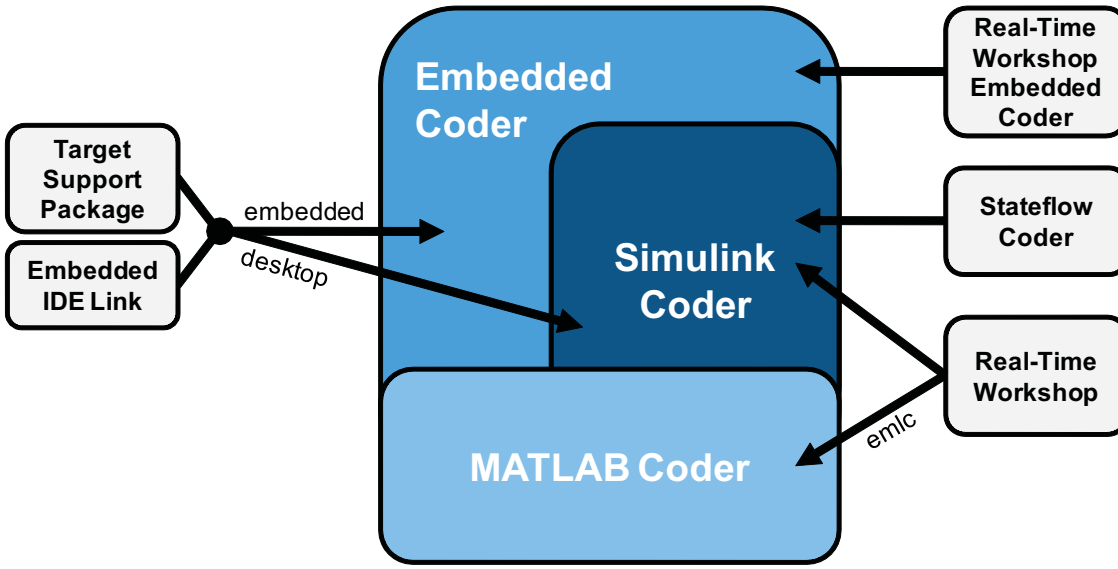
- "Product Restructuring Overview" on page 5
- "Resources for Upgrading from Real-Time Workshop Embedded Coder" on page 6
- "Migration of Embedded MATLAB Coder Features to MATLAB® Coder" on page 7
- "Migration of Embedded IDE Link and Target Support Package Features to Simulink® Coder and Embedded Coder" on page 7
- "Interface Changes Related to Product Restructuring" on page 8
- "Simulink Graphical User Interface Changes" on page 9
- "Compatibility Considerations" on page 9

Product Restructuring Overview

In R2011a, the Embedded Coder™ product replaces the Real-Time Workshop® Embedded Coder product. Additionally,

- The Simulink® Coder™ product combines and replaces the Real-Time Workshop and Stateflow® Coder™ products
- 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 Embedded IDE Link™ and Target Support Package™ products 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.



Resources for Upgrading from Real-Time Workshop Embedded Coder

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

- "Compatibility Summary for Embedded Coder Software" on page 35 (latest release)
- On the MathWorks web site, in the Archived documentation, select R2010b, and view the following tables, which are provided in the release notes for Real-Time Workshop Embedded Coder: *Compatibility Summary for Real-Time Workshop Embedded Coder Software*:

  This table provides 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 and view the corresponding tables for Embedded IDE Link or Target Support Package:

- *Compatibility Summary for Embedded IDE Link* (R2010b)
- *Compatibility Summary for Target Support Package* (R2010b)

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 "Generating C/C++ Code from MATLAB Code" in the MATLAB Coder documentation.

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 following table summarizes the transition of the Embedded IDE Link and Target Support Package supported hardware and software 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

Former Product	Supported Hardware and Software	Simulink Coder	Embedded Coder
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

Interface Changes Related to Product Restructuring

You will see interface changes as part of restructuring the Coder products.

- 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, changes to code generation related elements
- In error messages, tool tips, demos, and product documentation, references to Real-Time Workshop Embedded Coder, Real-Time Workshop, and Stateflow Coder and related terms are 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 Parameter 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 file name (no extension)**	On new **Code Generation** pane and renamed: • **Function packaging** • **Function name options** • **Function name** • **File name options** • **File name (no extension)**

Compatibility Considerations

In the Help browser **Contents** pane, Embedded Coder is now listed with the products for MATLAB, because Embedded Coder now supports both MATLAB Coder and Simulink Coder workflows.

Data Management Enhancements and Changes

- "Memory Section Enhancements" on page 10
- "No Longer Able to Set RTWInfo or CustomAttributes Property of Simulink Data Objects" on page 10
- "Parts of Data Class Infrastructure No Longer Available" on page 11
- "No Longer Generating Pragma for Data Defined with Built-In Storage Class ExportedGlobal, ImportedExtern, or ImportedExternPointer" on page 12
- "Simulink.CustomParameter and Simulink.CustomSignal Data Classes To Be Deprecated in a Future Release" on page 13

Memory Section Enhancements

- `Pragmas` are now added to data and function declarations (prior to R2011a they were added to definitions only); at compile time, this makes the compiler aware of memory locations for functions and data, potentially optimizing generated code
- New function category is available for shared utilities on the **Code Generation > Memory Sections** pane: "Shared utility"
- Referenced models can have a memory section that is different from that of the top model for the `InitTerm` and `Execute` function categories

No Longer Able to Set RTWInfo or CustomAttributes Property of Simulink Data Objects

You can no longer set the `RTWInfo` or `CustomAttributes` property of a Simulink data object from the MATLAB Command Window or a MATLAB script. Attempts to set these properties generate an error.

Although you cannot set `RTWInfo` or `CustomAttributes`, you can still set subproperties of `RTWInfo` and `CustomAttributes`.

Compatibility Considerations. Operations from the MATLAB Command Window or a MATLAB script, which set the data object property `RTWInfo` or `CustomAttributes`, generate an error.

For example, a MATLAB script might set these properties by copying a data object as shown below:

```
a = Simulink.Parameter;
b = Simulink.Parameter;
b.RTWInfo = a.RTWInfo;
b.RTWInfo.CustomAttributes = a.RTWInfo.CustomAttributes;
 .
 .
 .
```

To copy a data object, use the object's `deepCopy` method.

```
a = Simulink.Parameter;
b = a.deepCopy;
.
.
.
```

Parts of Data Class Infrastructure No Longer Available

Simulink has been generating warnings for usage of the following data class infrastructure features for several releases. As of R2011a, the features are no longer supported.

- Custom storage classes not captured in the custom storage class registration file (`csc_registration`) – *warning displayed since R14SP2*
- Built-in custom data class attributes `BitFieldName` and `FileName+IncludeDelimiter` – *warning displayed since R2008b*

Instead of...	Use...
`BitFieldName`	`StructName`
`FileName+IncludeDelimiter`	`HeaderFile`

- Initial value of MPT data objects inside `mpt.CustomRTWInfoSignal` – *warning displayed since R2006a*

Compatibility Considerations.

- When you use a removed feature, Simulink now generates an error.
- When loading a MAT-file that uses an unsupported feature, the load operation suppresses the generated error such that it is not visible. In addition, MATLAB silently deletes data that had been associated with the unsupported feature. To prevent loss of data when loading a MAT-file, load and resave the file with R2010b or earlier.

No Longer Generating Pragma for Data Defined with Built-In Storage Class ExportedGlobal, ImportedExtern, or ImportedExternPointer

The code generator no longer generates a `pragma` around definitions or declarations for data that has the following built-in storage classes:

- `ExportedGlobal`
- `ImportedExtern`
- `ImportedExternPointer`

Prior to R2011a, based on model configuration parameters for specifying memory sections and the built-in storage class defined for data, the code generator would do the following:

For Built-In Storage Class...	Generate `pragma` Around...
`ExportedGlobal`	Data definition and declaration
`ImportedExtern`	Data declaration
`ImportedExternPointer`	Data declaration

The code generator now treats data with these built-in storage classes like custom storage classes with no memory section specified.

Compatibility Considerations. To work around this change, select a custom storage class that uses the memory section of interest for the data.

Simulink.CustomParameter and Simulink.CustomSignal Data Classes To Be Deprecated in a Future Release

In a future release, data classes `Simulink.CustomParameter` and `Simulink.CustomSignal` will no longer be supported because they are equivalent to `Simulink.Parameter` and `Simulink.Signal`.

Compatibility Considerations. If you use the data class `Simulink.CustomParameter` or `Simulink.CustomSignal`, Simulink posts a warning that identifies the class and describes one or more techniques for eliminating it. You can ignore these warnings in R2011a, but consider making the described changes now because the classes will be removed in a future release.

AUTOSAR Enhancements

The following enhancements are available in R2011a.

Calibration Parameters

Previously, the software supported only calibration parameters that were defined by a calibration component. These parameters could be accessed by all AUTOSAR Software Components. The AUTOSAR standard also specifies an internal calibration parameter that is defined and accessed by only one AUTOSAR Software Component. The software now supports:

- AUTOSAR internal calibration parameters, including the import and export of initial values of these parameters.
- A bus object data type (AUTOSAR record type) to import and export both kinds of calibration parameters.

For more information, see “Calibration Parameters” and “Configuring Calibration Parameters” in the Embedded Coder documentation.

Multiple Runnables from Virtual Subsystems

Previously, if a wrapper subsystem had virtual subsystems containing function-call subsystems, you could not export the function-call subsystems as AUTOSAR runnables from the wrapper subsystem level. Now, within a wrapper subsystem, you can group function-call subsystems into virtual subsystems and generate runnables for these function-call subsystems. See

"Configuring Multiple Runnables" and "Exporting AUTOSAR Software Component" in the Embedded Coder documentation.

Support for Code Descriptor Elements

The AUTOSAR standard specifies that the XML description of an AUTOSAR Software Component implementation must contain code descriptor elements to describe generated source files and include header files. This feature allows AUTOSAR authoring tools that import software components to automate the building process for source code.

Previously, the software did not generate the software component implementation file (*`modelname`*`_implementation.arxml`) with these code descriptor elements. Now, when you build a Simulink model for an AUTOSAR target, the software generates a `CODE-DESCRIPTORS` element within the `SWC_IMPLEMENTATION` element. The `CODE-DESCRIPTORS` element contains `XFILE` elements that provide descriptions of the generated code.

For example, if you build the model `rtwdemo_autosar_counter`, the generated file `rtwdemo_autosar_counter_implementation.arxml` has the following `SWC_IMPLEMENTATION` element:

```
....
<SWC-IMPLEMENTATION>
  <SHORT-NAME>rtwdemo_autosar_counter</SHORT-NAME>
  <CODE-DESCRIPTORS>
    <CODE>
      <SHORT-NAME>Code</SHORT-NAME>
      <TYPE>SRC</TYPE>
      <XFILES>
        <XFILE>
          <SHORT-NAME>rtwdemo_autosar_counter_c</SHORT-NAME>
          <CATEGORY>GeneratedFile</CATEGORY>
          <URL>rtwdemo_autosar_counter_autosar_rtw\rtwdemo_autosar_counter.c</URL>
          <TOOL>Embedded Coder</TOOL>
          <TOOL-VERSION>5.6</TOOL-VERSION>
        </XFILE>
        <XFILE>
          <SHORT-NAME>rtwdemo_autosar_counter_h</SHORT-NAME>
          <CATEGORY>GeneratedFile</CATEGORY>
```

```
          <URL>rtwdemo_autosar_counter_autosar_rtw\rtwdemo_autosar_counter.h</URL>
          <TOOL>Embedded Coder</TOOL>
          <TOOL-VERSION>5.6</TOOL-VERSION>
        </XFILE>
          ...
      </XFILES>
    </CODE>
  </CODE-DESCRIPTORS>
  <CODE-GENERATOR>Embedded Coder 5.6 (R2011a) 26-Aug-2010</CODE-GENERATOR>
  <PROGRAMMING-LANGUAGE>C</PROGRAMMING-LANGUAGE>
</SWC-IMPLEMENTATION>
....
```

SIL and PIL Enhancements

Code Execution Profiling

You can collect execution time measurements in a specified base workspace variable during a software-in-the-loop (SIL) or processor-in-the-loop (PIL) simulation. At the end of the simulation, you can view or analyze the measurements within the MATLAB environment. This feature allows you to collect an execution time profile for each task within your generated code.

The software supports code execution profiling for all types of SIL or PIL simulations except the SIL block.

For more information, see “Code Execution Profiling” in the Embedded Coder documentation.

PIL Block Parameter Tuning

R2011a supports parameter tuning for the PIL block, which allows you to change tunable workspace parameters between or during simulations without regenerating code. This feature also includes support for tunable structure parameters. For more information, see “I/O Support” and “Tunable Parameters and SIL/PIL”.

Top-Model SIL/PIL and PIL Block Parameter Initialization

R2011a supports automatic definition and initialization of parameters with imported storage classes. For more information, see “I/O Support” and “Imported Data Definitions”.

Model Block Parameter Tuning and Model Initialization

Previously, the software did not support the following features for Model block SIL/PIL:

- Simplified initialization mode
- Tunable structure parameters

R2011a now supports these features. For more information, see “Configuration Parameters Support”, “I/O Support”, and “Tunable Parameters and SIL/PIL”.

Code Generation Enhancements

Improved Code for Data Store Memory In-place Assignment

Previously, the generated code for a Data Store Memory block used data copies to perform data store assignments. The generated code now eliminates the data copies and performs an in-place assignment. This improvement generates less code, uses less memory, and provides faster execution.

Improvements to Target Function Library Replacements

Enhancements to Target Function Library Replacements (TFL) include:

- If multiple TFL replacements occur within a function, temporary variables are now reused instead of creating extra temporary variables. This enhancement reduces the stack size during TFL replacement.
- During TFL replacement, if unnecessary temporary variables are introduced when block output is not the returned value of the function but one of the input arguments, code generation now removes the temporary variable. This enhancement improves execution speed and requires less memory.

For more information, see “Introduction to Target Function Libraries”.

Improved Loop Fusion

Code generation now includes the following:

- An improved loop fusion algorithm that reduces data copies. This enhancement decreases stack size, ROM consumption, and code generation time.
- Selectively fuses loops when the loop count is larger than the “Loop unrolling threshold”. In these cases, loop unrolling allows the code generator to perform more optimizations. In addition, the code generator groups the statements together to assign values to the elements of a signal or parameter array, which improves data access and code readability.

Improved Array Indexing

The generated code is optimized for more efficient array indexing. When a complex instruction is used repeatedly in an array index, the instruction is replaced with a temporary variable to perform the calculation more efficiently. This enhancement improves execution speed and reduces code size.

Improvement on Matrix Parameter Pooling

For matrix parameters with the same flattened value, the generated code now pools the matrix parameters even when they have different shapes. This enhancement reduces ROM consumption.

Readability Improvements Involving Data References

For references to the root inport and outport, as well as DWork, unnecessary parentheses are removed from the generated code. This enhancement produces more readable code.

Code Generation Verification (CGV) API Updates

Support for Adding Multiple Callback Functions

In R2011a, the cgv.CGV class includes new methods to add callback functions. These methods replace the `cgv.CGV.addCallback` method which added only a

pre-execution callback function. Now, the new methods allow CGV to invoke callback functions at several stages of the cgv.CGV.run execution. The new methods are:

- cgv.CGV.addHeaderReportFcn adds a callback function invoked before executing any input data in the `cgv.CGV` object.
- cgv.CGV.addPreExecReportFcn adds a callback function invoked before executing each input data file in the `cgv.CGV` object.
- cgv.CGV.addPreExecFcn adds a callback function invoked before executing each input data file in the `cgv.CGV` object.
- cgv.CGV.addPostExecReportFcn adds a callback function invoked after executing each input data file in the `cgv.CGV` object.
- cgv.CGV.addPostExecFcn adds a callback function invoked after executing each input data file in the `cgv.CGV` object.
- cgv.CGV.addTrailerReportFcn adds a callback function invoked after executing all input data in the `cgv.CGV` object.

New Functionality Added to the cgv.CGV Class

The `cgv.CGV` class now includes the following methods:

- cgv.CGV.activateConfigSet activates the configuration set of a model.
- cgv.CGV.addBaseline adds a file of baseline data for comparison.
- cgv.CGV.copySetup creates a copy of a `cgv.CGV` object.
- cgv.CGV.setMode specifies the mode of execution (`sim`, `sil`, or `pil`).
- cgv.CGV.copySetup returns the status of the execution of the `cgv.CGV` object.

The `cgv.CGV` class now includes the following properties:

- `Name`
- `Description`

Compatibility Considerations

Previously, the `cgv.CGV` class included parameters that you set to perform automatic configuration checks of your model. In R2011a, `cgv.CGV` class no longer performs automatic configuration checks. Instead, you can use the cgv.Config class to perform a manual configuration check of your model. Before calling `cgv.CGV.run`, MathWorks recommends that you perform a manual configuration check of your model. Otherwise, an error might occur later in the process. For more information, see "Verifying Numerical Equivalence with Code Generation Verification".

Changes to the `cgv.CGV` class parameters are listed in the following table.

Parameter	What Happens When You Use Parameter?	Use This Parameter Instead	Compatibility Considerations
`LogMode` removed from `cgv.CGV`	Errors	`LogMode` parameter in `cgv.Config`	To check your model before running CGV, pass the `LogMode` parameter to the constructor for `cgv.Config`. Then call the cgv.Config.configModel method to adjust the model configuration.
`Processor` removed from `cgv.CGV`	Errors	`Processor` parameter in `cgv.Config`	To check your model before running CGV, pass the `Processor` parameter to the constructor for `cgv.Config`. Then call the cgv.Config.configModel method to adjust the model configuration.

Parameter	What Happens When You Use Parameter?	Use This Parameter Instead	Compatibility Considerations
`SaveModel` removed from `cgv.CGV`	Errors	`SaveModel` parameter in `cgv.Config`	To check your model before running CGV, pass the `SaveModel` parameter to the constructor for `cgv.Config`. Then call the cgv.Config.configModel method to adjust the model configuration.
`ConfigModel` removed from `cgv.CGV`	Warns if set to `off` Errors if set to `on`	cgv.Config.configModel method	To check your model before running CGV, replace the `cgv.CGVConfigModel` parameter with a call to the cgv.Config.configModel method
`CheckInterface` parameter from `cgv.CGV`	Warns if set to `off` Errors if set to `on`	`CheckOutports` parameter in `cgv.Config`	To check your model before running CGV, pass the `CheckOutports` parameter to the constructor for `cgv.Config`. Then call the cgv.Config.configModel method to adjust the model configuration.
`tasking` and `custom` values removed from the `Connectivity` parameter of `cgv.CGV`	Errors	`pil`, a new value for the `cgv.CGV` `Connectivity` parameter	Replace calls to the `cgv.CGV` constructor using the parameter-value arguments, `('Connectivity',`

Parameter	What Happens When You Use Parameter?	Use This Parameter Instead	Compatibility Considerations
			'tasking') or ('Connectivity', 'custom'), with ('Connectivity, 'pil').

Changes to the cgv.Config class parameters are listed in the following table:

Parameter	What Happens When You Use Parameter?	Use This Parameter Instead	Compatibility Considerations
CheckOutports parameter added to cgv.Config	Defaults to on. Compiles the model. Then checks that the model outport configuration is compatible with the cgv.CGV object.		If your script fixes errors reported by cgv.Config, you can set CheckOutports to off.
LogMode parameter from cgv.Config	Change in behavior		If you do not give a value for LogMode, no changes are made to the configuration parameters for logging.

MISRA-C Code Generation Objective

The Code Generation Advisor now includes a new objective for MISRA-C:2004 guidelines. To set the new objective, open the Configuration Parameters dialog box and select the **Code Generation** pane. In the Code Generation Advisor section, click the **Set objectives** button to open the Code Generation Advisor dialog box. In the **Available objectives** list, select MISRA-C:2004 guidelines and click the select button (arrow pointing right) to move the

objective to the **Selected objectives** list. For more information on setting objectives, see "Configuring Code Generation Objectives".

New Model Advisor Check for Code Efficiency of Lookup Table Blocks

The Simulink Model Advisor includes the following new check for code efficiency of lookup table blocks: "Identify lookup table blocks that generate expensive out-of-range checking code". By default, the following blocks generate code that checks for out-of-range breakpoint inputs:

- 1-D Lookup Table
- 2-D Lookup Table
- n-D Lookup Table
- Prelookup

Similarly, the Interpolation Using Prelookup block generates code that checks for out-of-range index inputs. Running this Model Advisor check helps you identify lookup table blocks that generate out-of-range checking code for breakpoint or index inputs.

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

Enhanced Code Generation Optimization

The **Optimize using specified minimum and maximum values** code generation option now takes into account the minimum and maximum values specified for:

- A `Simulink.Parameter` object provided that it is used on its own. It does not use these minimum and maximum values if the object is part of an expression. For example, if a Gain block has a gain parameter specified as `K1`, where `K1` is defined as a `Simulink.Parameter` object in the base workspace, the optimization takes the minimum and maximum values of `K1` into account. However, if the Gain block has a gain parameter of `K1+5` or `K1+K2+K3`, where `K2` and `K3` are also `Simulink.Parameter` objects,

the optimization does not use the minimum and maximum values of `K1`, `K2` or `K3`.

- All design ranges specified on block outputs in a conditionally-executed subsystem, except for the block outputs that are directly connected to an Outport block.

For more information, see "Optimizing Generated Code Using Specified Minimum and Maximum Values".

Target Function Library Replacement Based on Computation Method for Reciprocal Sqrt, Sine, and Cosine

Target function libraries (TFLs) now support the ability to control replacement of certain math functions using their computation method as a distinguishing attribute. For example,

- The rSqrt block can be configured to use either of two computation methods, `Newton-Raphson` or `Exact`.
- The Trigonometric Function block, with **Function** set to `sin` or `cos`, can be configured to use either of two approximation methods, `CORDIC` or `None`.

You can configure TFL table entries to replace these functions for one or all of the available computation methods. For example, you could replace only `Newton-Raphson` instances of the `rSqrt` function.

For more information, see "Replacing Math Functions Based on Computation Method" in the Embedded Coder documentation.

C++ Encapsulation Allowed for Referenced Models in For Each Subsystems

In previous releases, due to a code generation limitation, code could not be generated for a For Each Subsystem block under the following conditions:

- The For Each Subsystem block directly or indirectly contains a Model block.
- The Model block references a model for which C++ encapsulation is selected.

R2011a removes this limitation. You can now generate code for a For Each Subsystem in which a referenced model uses C++ encapsulation.

Improved Code Generation for Portable Word Sizes

In the software-in-the-loop (SIL) simulation work flow, the model option **Enable portable word sizes** allows you to take code intended for a specific target platform and compile and run the same code on a MATLAB host platform that uses different processor word sizes. R2011a enhances the code generated for portable word sizes by inserting explicit casts to help protect against integral promotion differences and other behavior differences between host and target. This potentially can reduce the incidence of numerical differences due to host/target behavior differences. For more information, see "Configuring Hardware Implementation Settings for SIL" and "Portable Word Sizes Limitations" in the Embedded Coder documentation.

Improved Comments in the Generated Code

R2011a provides improvements to comment generation for better readability and understanding of the generated code. Specifically, comments are located closer to the referring code and more accurately reflect the intent of the code. An end comment is now included at the end of a control flow block of code. For information on customizing comments in the generated code, see "Customizing Comments in Generated Code".

Replacement Data Types and Simulation Mode for Referenced Models

To replace built-in data type names with user-defined data type names in the generated code for a referenced model, you must set the **Simulation mode** parameter for the Model block to one of the following:

- `Normal`
- `Software-in-the-loop (SIL)`
- `Processor-in-the-loop (PIL)`

For more information, see "Renaming and Replacing Data Types" and "Referenced Model Simulation Modes" in the Simulink documentation.

Changes for Embedded IDEs and Embedded Targets

- "Feature Support for Embedded IDEs and Embedded Targets" on page 25
- "Execution Profiling during PIL Simulation" on page 26
- "Location of Blocks for Embedded Targets" on page 26
- "Location of Demos for Embedded IDEs and Embedded Targets" on page 28
- "Multicore Deployment with Rate-Based Multithreading" on page 29
- "Windows-Based Code Generation and Remote Build On Linux Target (BeagleBoard)" on page 29
- "Capture Video Input from USB Cameras on Linux and Embedded Linux" on page 29
- "Changes to Frame-Based Processing" on page 29
- "New Support for Analog Devices Blackfin BF50x and BF51x Processors" on page 31
- "Generate Optimized Fixed-Point Code for ARM Cortex-M3, Cortex-A8, and Cortex-A9 Processors" on page 32
- "Support for Versions 5.0.6 and 5.1.6 of Green Hills® MULTI" on page 32
- "Support for Texas Instruments Delfino C2834x Processors" on page 32
- "Ending Support for Altium TASKING in a Future Release" on page 33
- "Ending Support for Freescale MPC5xx in a Future Release" on page 33
- "Ending Support for Infineon® C166 in a Future Release" on page 33
- "Removed Methods and Arguments" on page 33

Feature Support for Embedded IDEs and Embedded Targets

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

- Automation Interface
- Processor-in-the-Loop (PIL) Simulation
- Execution Profiling

- Execution Profiling during PIL Simulation
- Stack Profiler
- External Mode
- Schedulers and Timing
- Makefile Generation (XMakefile)
- Target Function Library (TFL) Optimization
- Multicore Deployment for Rate Based Multithreading

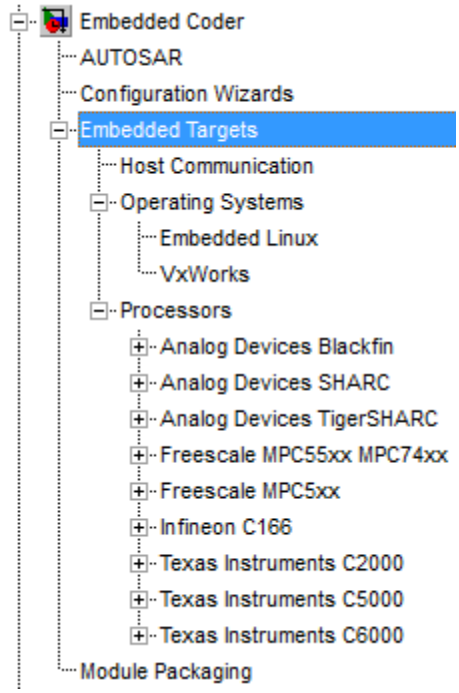
Note You can only use these features in the 32-bit version of your MathWorks products. To use these features on 64-bit hardware, install and run the 32-bit versions of your MathWorks products.

Execution Profiling during PIL Simulation

During Processor-in-the-loop (PIL) simulation, you can profile synchronous tasks in code running on the target. For more information, see Execution Profiling during PIL Simulation

Location of Blocks for Embedded Targets

Blocks from the former Target Support Package product and Embedded IDE Link product now reside under Embedded Coder in the Embedded Targets block library, as shown.



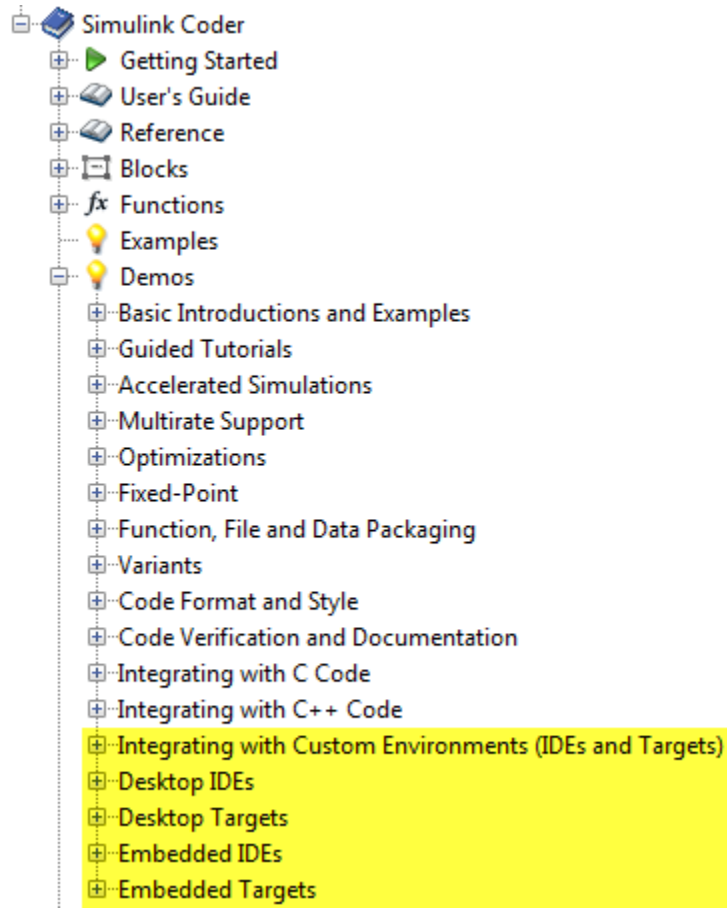
Embedded Targets includes the following types of blocks:

- Host Communication
- Operating Systems
  - Embedded Linux
  - VxWorks
- Processors
  - Analog Devices Blackfin
  - Analog Devices™ SHARC®
  - Analog Devices™ TigerSHARC®
  - Freescale MPC55xx MPC74xx
  - Freescale MPC5xx
  - Infineon C166

- Texas Instruments C2000
- Texas Instruments C5000
- Texas Instruments C6000

Location of Demos for Embedded IDEs and Embedded 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 Embedded Linux and

VxWorks. This feature improves performance by taking advantage of multicore hardware resources.

Also see the “Running Target Applications on Multicore Processors” user’s guide topic.

Windows-Based Code Generation and Remote Build On Linux Target (BeagleBoard)

You can generate a makefile project on a Windows host machine, transfer the makefile project to an remote target running Linux, such as a BeagleBoard, and then build the executable on the remote target.

Capture Video Input from USB Cameras on Linux and Embedded Linux

You can use the new Video Capture block to develop and prototype video applications for Embedded Linux running on platforms such as the BeagleBoard. This block uses the Linux V4L2 API device driver framework, which supports most USB cameras.

Video Capture block can synthesize several types of video outputs. On Linux host computers, the block can also generate a live video output during model simulations. For more information, see the Linux Video Capture block reference topic. See the Video Stabilization demo in the product help for Simulink Coder under “Demos”.

Changes to Frame-Based Processing

Signal processing applications often process sequential samples of data at once as a group, rather than one sample at a time. MathWorks documentation refers to the former as *frame-based processing* and the latter as *sample-based processing*. A *frame* is a collection of samples of data, sequential in time. To perform frame-based processing in MathWorks products, you must have a DSP System Toolbox™ license.

Historically, Simulink-family products that can perform frame-based processing propagate frame-based signals throughout a model. The frame status is an attribute of the signals in a model, just as data type and dimensions are attributes of a signal. The Simulink engine propagates the frame attribute of a signal with a frame bit, which can either be on or off. When the frame bit is on, Simulink interprets the signal as frame-based, and displays it as a double line, rather than as a single line.

Beginning in R2010b, MathWorks started to change the handling of frame-based processing significantly. In the future, signal attributes will not include frame status. Instead, individual blocks will control whether they treat data inputs as frames or as samples.

To transition to this new paradigm, blocks that can perform sample- and frame-based processing contain a new **Input processing** parameter that specifies the appropriate processing behavior. You can set **Input processing** to `Columns as channels (frame based)` or `Elements as channels (sample based)`. The third option, `Inherited (this choice will be removed - see release notes)`, is a temporary selection. This third option helps you migrate your existing models from the old paradigm to the new paradigm.

In R2011a, the following Embedded Coder blocks received a new **Input processing** parameter:

- C62X Real Forward Lattice All-Pole IIR
- C62X Complex FIR
- C62X General Real FIR
- C62X Real IIR
- C64X Real Forward Lattice All-Pole IIR

Compatibility Considerations. When you load an existing model in R2011a, blocks with the new **Input processing** parameter shows a setting of `Inherited (this choice will be removed - see release notes)`. This setting enables your existing models to work as expected until you upgrade them. Upgrade your models as soon as possible.

To upgrade your existing models, use the `slupdate` function. This function detects all blocks that have **Input processing** set to `Inherited (this choice will be remove - see release notes)`. The function asks you whether to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is `1` (frames), the function sets the **Input processing** parameter to `Columns as channels (frame based)`. If the bit is `0` (samples), the function sets the parameter to `Elements as channels (sample based)`.

A future release will remove the frame bit and the `Inherited (this choice will be removed - see release notes)` option. At that time, if you have not updated the model, the software automatically sets the **Input processing** parameter. The software uses the library default setting of the block to select either `Columns as channels (frame based)` or `Elements as channels (sample based)`. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the removal of the frame bit, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, upgrade your existing modes using `slupdate` as soon as possible.

New Support for Analog Devices Blackfin BF50x and BF51x Processors

You can now generate code for the following embedded processors when you use Embedded Coder software:

- BF504
- BF504F
- BF506F
- BF512
- BF514
- BF516
- BF518

Generate Optimized Fixed-Point Code for ARM Cortex-M3, Cortex-A8, and Cortex-A9 Processors

You can use new Target Function Libraries (TFLs) to generate efficient fixed-point code for the ARM Cortex-M3, Cortex-A8, and Cortex-A9 processors. These TFLs include GCC compiler extensions and intrinsic functions that optimize the code Embedded Coder generates for these processors.

Support for Versions 5.0.6 and 5.1.6 of Green Hills MULTI

Support for Green Hills MULTI software now includes versions 5.0.6 and 5.1.6. For additional information about supported versions, see the Support for Green Hills MULTI topic online.

Support for Texas Instruments Delfino C2834x Processors

You can now generate code for the following embedded processors when you use Embedded Coder software with Texas Instruments Code Composer Studio™ software:

- C28341
- C28342
- C28343
- C28344
- C28345
- C28346

The new "C2834x (c2834xlib)" block library contains the following blocks:

- C2000 CAN Calibration Protocol
- C280x/C2802x/C2803x/C28x3x/c2834x GPIO Digital Input
- C280x/C2802x/C2803x/C28x3x/c2834x GPIO Digital Output
- C280x/C2802x/C2803x/C28x3x/C2834x I2C Receive
- C280x/C2802x/C2803x/C28x3x/C2834x I2C Transmit
- C280x/C2802x/C2803x/C28x3x/C2843x SCI Receive
- C280x/C2802x/C2803x/C28x3x/C2843x SCI Transmit

- C280x/C2802x/C2803x/C28x3x/C2843x SPI Receive
- C280x/C2802x/C2803x/C28x3x/C2843x SPI Transmit
- C280x/C2802x/C2803x/C28x3x/C2843x Software Interrupt Trigger
- C28x Watchdog
- C280x/C2803x/C28x3x/c2834x eCAN Receive
- C280x/C2803x/C28x3x/c2834x eCAN Transmit
- C280x/C2802x/C2803x/C28x3x/c2834x eCAP
- C280x/C2802x/C2803x/C28x3x/c2834x ePWM
- C280x/C2803x/C28x3x/c2834x eQEP

Ending Support for Altium TASKING in a Future Release

Support for the Altium TASKING IDE will end in a future release of the Embedded Coder product.

Ending Support for Freescale MPC5xx in a Future Release

Support for the Freescale MPC5xx processor family will end in a future release of the Embedded Coder product.

Ending Support for Infineon C166 in a Future Release

Support for the Infineon C166 processor family will end in a future release of the Embedded Coder product.

Removed Methods and Arguments

Deprecated the `type` property for the Code Composer Studio IDE object. For example, entering the following text generates an error message:

```
infolist = IDE_Obj.list(type)
```

Changes to ver Function Product Arguments

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

- The new argument `'embeddedcoder'` returns information about the installed version of the Embedded Coder product.
- The argument `'ecoder'`, which previously returned information about the installed version of the Real-Time Workshop® Embedded 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 `'ecoder'` argument, update the script appropriately. For example, you can update the `ver` call to use the `'embeddedcoder'` argument.

New and Enhanced Demos

The following demos have been added in R2011a:

Demo...	Shows How You Can...
`coderdemo_tfl`	Use target function libraries (TFLs) to replace operators and functions in code generated by MATLAB Coder.
`rtwdemo_code_coverage_script`	Generate model coverage and code coverage reports, and use these reports to compare model coverage and code coverage results for any part of a model.
`rtwdemo_pmsmfoc_script`	Perform system-level simulation and algorithmic code generation using Field-Oriented Control for a Permanent Magnet Synchronous Machine.

The following demos have been enhanced in R2011a:

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

Compatibility Summary for Embedded Coder Software

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

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version **V6.0 (R2011a)**	See the **Compatibility Considerations** subheading for each of these new features or changes: • Code Generation Verification classes, `cgv.CGV` and `cgv.Config` • “Changes to ver Function Product Arguments” on page 33 • “Changes to Frame-Based Processing” on page 29 • “Coder Product Restructuring” on page 5