Simulink® Release Notes

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

	Latest news:	www.mathworks.com
	Sales and services:	www.mathworks.com/sales_and_services
	User community:	www.mathworks.com/matlabcentral
	Technical support:	www.mathworks.com/support/contact_us
	Phone:	508-647-7000

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

Simulink® Release Notes

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

R2016b

R2016a

R2015aSP1

Bug Fixes

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010bSP2

Bug Fixes

R2010bSP1

Bug Fixes

R2010a

R2009bSP1

Bug Fixes

R2009b

R2009a

R2008b

R2008a

R2007b

R2007a+

Bug Fixes

R2006b

R2006a+

No New Features or Changes

R2006a

R14SP2

R2017a

Version: 8.9

New Features

Bug Fixes

Compatibility Considerations

Simulation Analysis and Performance

Parallel Simulations: Directly run multiple parallel simulations from the `parsim` command

Using this feature, you can iteratively change various parameter values in your model, starting from a baseline setting, and perform a series of simulations with these values. You can provide these changes to your model through a `SimulationInput` object and run multiple simulations with them. Some common use cases for this approach include model testing, design of experiments, Monte Carlo runs, and sensitivity and robustness analysis.

In addition, with a Parallel Computing Toolbox™ license, you can use the `parsim` command to run multiple simulations in parallel. With the `parsim` command, the amount of custom code you need to run multiple simulations in parallel is significantly reduced compared to using the `sim` command within a `parfor` loop.

When you use a `SimulationInput` object to run multiple simulations, the simulations use the values in the `SimulationInput` object rather than the values defined in your model. This enables you to run multiple simulations without needing to modify your model between each simulation. The `SimulationInput` object allows you to change the following settings in your model:

- Initial State
- External Inputs
- Model Parameters
- Block Parameters
- Variables

For more information, see "Run Multiple Simulations".

Simulink Cache: Get simulation results faster by using shared model artifacts

In R2017a, performing an update diagram or running a simulation on a new model that builds a model reference SIM target or rapid accelerator target creates a Simulink® cache

file (an `.slxc` file). Use Simulink cache files to share referenced model build artifacts without repeating the cost of a first-time build. Some examples of situations when the cache files can speed up simulation include:

- First-time builds for later use of a referenced model by yourself or others
- Parallel simulations

For details, see "Reuse Simulation Builds for Faster Simulations".

Inport File Streaming: Stream large input signals from MAT-files without loading the data into memory

To log, load, and analyze large amounts of simulation data that cannot be stored in memory, you can use Simulink data streaming and MATLAB® big data features. R2017a includes features that help you to:

- Use big data in Simulink logging and loading.
- Use in-memory `timetable` format data within `Simulink.SimulationData.Dataset` objects, in combination with other kinds of data, such as `timeseries`.
- Use MATLAB big data analysis features.

For details, see "Working with Big Data for Simulations".

Log and Load Big Data

You can use a `matlab.io.datastore.SimulationDatastore` object in a `Simulink.SimulationData.Dataset` object to stream data into a root Inport block. Create a `SimulationDatastore` object for an individual signal that is stored in a in-file `Dataset` object referenced by a `Simulink.SimulationData.DatasetRef` object.

To obtain a `SimulationDatastore` object from a `DatasetRef` object, you can use curly braces indexing syntax with a `DatasetRef` object (for example, `simDatastore = dsr{3}` returns the third `DatasetRef` element as a `SimulationDatastore` object.

To load logged signal data using root Inport blocks, you can use the new `createInputDataset` function to create a `Dataset` object that contains elements that correspond to root-level Inport blocks in the model.

Use timetable Data in Dataset Objects

You can use the new **DatasetSignalFormat** configuration parameter to specify whether you want data logged in `Dataset` format to be saved in `timeseries` (default) or in-memory `timetable` format within the `Dataset` object.

When you read data from a `SimulationDatastore` object, the data is returned as a `timetable` object.

Analyze in MATLAB Big Data Created by Simulink

You can use a `SimulationDatastore` object to output a MATLAB tall `timetable` object to analyze big data from a simulation.

Unified Streaming and Logging: Mark a signal once to stream it to the Simulation Data Inspector and log it to the MATLAB workspace

In R2017a, when you log signals, the data is logged to the workspace and streamed to the Simulation Data Inspector.

1   In the Simulink Editor, select one or more signals.

2
    Click the **Simulation Data Inspector** button arrow ![icon] and click **Log Selected Signals**.

3   Use the **Data Import/Export** > **Signal logging** configuration parameter to control whether signal data is exported to the MATLAB workspace at the end of simulation. If you clear this parameter, signals are streamed to the Simulation Data Inspector but not exported to the workspace.

Note: You can use normal, accelerator, rapid accelerator, SIL, and PIL simulation modes when you log and stream to the Simulation Data Inspector.

You use signal logging with these products:

- Stateflow® — Elements in a chart that can be logged are not streamed, but are logged to the workspace.
- Simulink Real-Time™

Simulation Data Inspector: Run simulation comparisons with a new UI, time tolerance support, and faster performance

The Simulation Data Inspector has a new, streamlined interface. Examples of the changes include:

- Inspection and formatting options are integrated into the **Inspect** and **Compare** views, instead of being on a toolbar.
- You can specify display options for axes lines and plots, such as color.
- You can format, group, and sort the signal data.
- The **Compare** view displays status information how many signals are within the tolerance range.

When you compare simulation runs and signals, you can set now set time tolerances. The plot for each signal displays a band representing the tolerance range.

In R2017a, to save Simulation Data Inspector session data and setup is faster than in previous releases, you can specify the `.mldatx` file extension instead of the `.mat` extension. A status overlay displays information about the save operation. For `.mldatx` files, you can specify preferences to compress the file and to limit the amount of data stored. When you open a `.mldatx` file with MATLAB closed, MATLAB and the Simulation Data Inspector open.

Dashboard Block Connection Indicators: Easily determine which block in your model is associated with a given Dashboard block

Dashboard block connection indicators provide visual indications of the connections between Dashboard blocks and signals and components in your model. You can click on a Dashboard block to highlight the connected block or signal and vice versa.
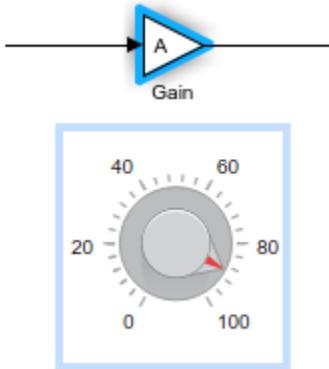
Dashboard block binding indicators also allow you to jump to the connected element through subsystem hierarchies to locate the connection in the model.
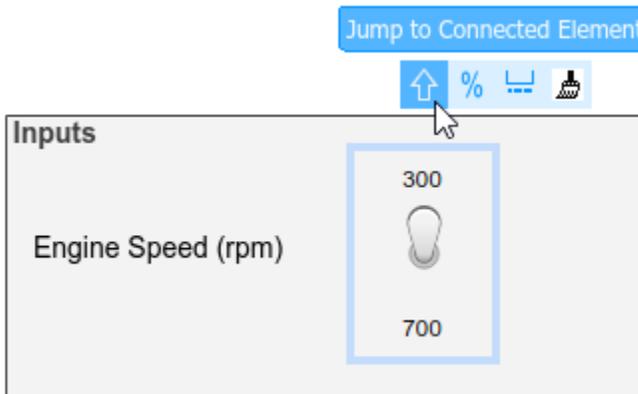
Signal Tracing: Incrementally trace and highlight paths for debugging

Use the **Highlight Signal to Destination** option from the context menu for the signal to highlight the actual destination for a signal. For example, selecting this option for a bus highlights only the signals actually selected from the bus. For more information, see "Display Signal Sources and Destinations".

Root-Level Inport Blocks: Create dataset for root-level Inport blocks

To generate a `Simulink.SimulationData.Dataset` object from the root-level Inport blocks in a model, you can use the `createInputDataset` function. Signals in the generated dataset have the properties of the Inport blocks and the corresponding ground values at model start and stop times. You can create `timeseries` objects for the time and values for signals for which you want to load data for simulation. The other signals use ground values.

Simulation Logging Data and Metadata: Access simulation data and information more directly

You can use these new capabilities to access simulation outputs in `SimulationOutput` objects and data in `Dataset` format more easily.

- Access logged simulation data (for example, `tout`, `xout`, and `yout`) as properties of the `out` element of a Simulink.SimulationOutput object. For example, you can access output data that uses the default variable named `yout`:

  ```
  simOut = sim('vdp','SaveOutput','on')
  myOutput = simOut.simOut.yout
  ```
- Use the Variable Editor to access logged data.

  - Each element in a Simulink.SimulationData.Dataset is a separate row.
  - Display `Dataset` objects to view information about elements, such as the block path and type of data the element contains.
  - Use the `openvar` function to view a logged data variable in the Variable Editor.
  - Use curly braces to streamline indexing syntax to access elements in a `Dataset` object, instead of using `get`, `getElement`, `setElement`, or `addElement` methods. For example, to return the second element of a `Dataset`:

    ```
    xout{2}
    ```

You can use these new capabilities to access simulation logging error messages and metadata on a `SimulationOutput` object.

- View error message and information about the stack and causes for simulation data by using the `SimulationOutput` object `ErrorMessage` property.
- Display the error message in the Variable Editor, by double-clicking the **ErrorMessage** row.

- Access an exception by using:

  `Simulink.SimulationMetadata.ExecutionInfo.ErrorDiagnostics.Diagnostic()`
- View simulation metadata by using the `SimulationMetadata` property of the `SimulationOutput` object.
- Use tab completion to access `SimulationMetadata` object properties such as `ModelInfo` and to access field names.
- Display simulation metadata in the Variable Editor using one of these approaches:
  - Select the **Show Simulation Metadata** check box (which displays the data in a tree structure).
  - Double-click the **SimulationMetadata** row.
  - View the `SimulationOutput` object.

For parallel simulations, for which you specify an array of input objects, if you are logging to file, Simulink

- Creates Simulink.SimulationData.DatasetRef objects to access output data in the MAT-file and includes those objects in the `SimulationOutput` object data
- Enables the `CaptureErrors` argument for simulation.

Rapid Accelerator mode: Rapid Accelerator now supports S-functions without source code

Rapid Accelerator mode can now use C S-functions without storing the S-function source code. You do not need to collocate the S-function source code with the S-function binaries. This option enables easier sharing of models.

Signal Editor: Create and edit input signals that can be organized for multiple simulations

Use the Signal Editor to create and edit input signals that you can organize for multiple simulations. For more information, see "Create and Edit Signal Data".

Improved simulation performance when stepping back is enabled

The performance of stepping back using the Simulation Stepper has been improved. For more information on the Simulation Stepper, see Simulation Stepping.

Simulink Diagnostic Management: Control which simulation and fixed-point diagnostic warnings you receive from specific blocks, including model reference

Select blocks with certain diagnostic suppressions by default

Beginning in R2017a, the Counter Free-Running, HDL Counter, Counter Limited, and Extract Bits blocks no longer report wrap on overflow warnings. The blocks continue to report errors due to wrap on overflows. You can restore the warning diagnostic by breaking the library link and using the `Simulink.restoreDiagnostic` function.

Diagnostic suppressor functions support `MSLDiagnostic` as input argument

You can now suppress and restore certain diagnostic warnings thrown by your model using a `Simulink.MSLDiagnostic` object as an input to the `Simulink.suppressDiagnostic` and `Simulink.restoreDiagnostic` functions.

To use simulation metadata and `MSLDiagnostic` objects, use `set_param` to set `ReturnWorkspaceOutputs` to `on`. Store the simulation output in a variable.

```
set_param(model_name,'ReturnWorkspaceOutputs','on');
out = sim(model_name);
```

Access the `MSLDiagnostic` object through the simulation output.

```
diag = out.getSimulationMetadata.ExecutionInfo.WarningDiagnostics(1).Diagnostic

diag =

  MSLDiagnostic with properties:

    identifier: 'SimulinkFixedPoint:util:fxpParameterPrecisionLoss'
       message: 'Parameter precision loss occurred for 'Value' of
       'Suppressor_CLI_Demo/one'. The parameter's value cannot be
       represented exactly using the run-time data type. A small
       quantization error has occurred. To disable this warning or error,
       in the Configuration Parameters > Diagnostics > Data Validity pane,
       set the 'Detect precision loss' option in the Parameters group to 'none'.'
         paths: {'Suppressor_CLI_Demo/one'}
         cause: {}
         stack: [0×1 struct]
```

Use the `Simulink.suppressDiagnostic` function to suppress the diagnostic warning specified by the `MSLDiagnostic` object, `diag`.

```
Simulink.suppressDiagnostic(diag)
```

You can restore the diagnostic using the `Simulink.restoreDiagnostic` function

```
Simulink.restoreDiagnostic(diag)
```

Improved workflow for suppressing diagnostics from referenced models

You can now suppress certain diagnostic warnings for specified instances of warnings in a referenced model. By accessing the `MSLDiagnostic` object of the specific instance of the warning, you can suppress the warning only when the block inside the referenced model is simulated from the specified top model.

Absolute tolerance for continuous variable step solver tied to the relative tolerance

The behavior of the auto absolute tolerance for continuous variable step solvers has changed. Previously, setting the absolute tolerance to `auto` specified the initial value as 1e-6, which was adjusted by the solver during simulation. Now, the absolute tolerance is tied to the relative tolerance. The value is initially set to the minimum of 1e-6 and 1e-3 times the relative tolerance and adjusted by the solver during simulation. This change enhances the overall solver robustness and improves simulation accuracy. For models with auto absolute tolerance developed before R2017a, simulation performance may be improved by increasing maximum step size and loosening relative tolerance.

Simulink Editor

Automatic Port Creation: Add inports and outports to blocks when routing signals

For some blocks in Simulink models, dragging a line to connect another block to it adds a port. For example, dragging a line from a block to a subsystem adds a port to the subsystem and the inport or output block inside the subsystem. For an example, see "Build and Edit a Model in the Simulink Editor".

These blocks add ports when you connect a signal to them:

- Subsystem blocks except masked blocks and the Configurable Subsystem block
- Stateflow charts, truth tables, and state transition tables
- Bus Creator, Bus Selector, Mux, Demux, and Merge blocks
- Vector Concatenate and Matrix Concatenate blocks
- Add, Sum, Subtract, Sum of Elements, Product, and Product of Elements blocks
- Scope blocks
- Logical Operator blocks
- MATLAB Function block

Model Block Masking: Customize the parameter dialog boxes for referenced models

You can now apply a mask on a Simulink model and create a customized parameter dialog box for the referenced model. For more information, see "Create and Reference a Masked Model".

Quick Find: Use a modifier to search for model properties in search box

In R2017a, you can search for property values in a model using a modifier in the search box. Enter the property and value you want to search for, in the form `Property:Value`. For more information on using modifiers in the **Find** box, see "Search for Model Elements Using Find".

Format Painter: Copy formatting between model elements

Using the ellipsis menu on a block, area, or signal line in a model, you can copy the formatting from one model element to another. The **Copy Formatting** brush appears on the menu when the model element you select has formatting applied. Examples of formatting include a font change or foreground or background color. For more information, see "Adjust Visual Presentation to Improve Model Readability".

Refresh Library Browser: Update quick insert list with custom libraries using menu command

In R2017a, you can update the contents of the quick insert list to use your current configuration of custom libraries by using the **Refresh Library Browser** command. Use the quick insert interface to add blocks to your model without leaving the canvas. To use quick insert, click where you want to add a block and start typing the name of the block.

To ensure that the quick insert interface includes the blocks from your custom libraries currently in effect, use the Library Browser **Refresh Library Browser** command. In the Library Browser, right-click anywhere in the library list and select **Refresh Library Browser**.

Functionality Being Removed or Changed

Two functions in the `Simulink.BlockDiagram` class have been renamed.

Functionality

These functions were renamed:

- `Simulink.BlockDiagram.copyContentsToSubSystem`
- `Simulink.BlockDiagram.createSubSystem`

Result

The renamed functions still run.

Use Instead

The functions were renamed to:

- `Simulink.BlockDiagram.copyContentsToSubsystem`

- `Simulink.BlockDiagram.createSubsystem`

Compatibility Consideration

For each function, you can use the earlier or newer syntax.

Compatibility Considerations

Scripts that use the earlier syntax will continue to work. Use the new syntax going forward to match the spelling of other functions in this class.

Optimize rendering during mask icon drawing

When the mask drawing commands in the Mask Editor do not have dependency on the mask workspace, you can specify the value of the **Run initialization** option as **Off**.

Setting the value to **Off** helps to optimize Simulink performance during mask icon rendering because the mask initialization commands are not executed. For more information, see "Rules for Initialization commands".

Component-Based Modeling

Reduced Bus Wiring: Quickly group signals as buses and automatically create bus element ports for fewer signal lines between and within subsystems

The Ports & Subsystems library contains new In Bus Element and Out Bus Element blocks. These bus element port blocks provide a simplified and flexible way to use bus signals as inputs and outputs to subsystems.

The In Bus Element block is equivalent to an Inport block combined with a Bus Selector block. The Out Bus Element block is equivalent to an Outport block combined with a Bus Creator block. These new blocks are of block type Inport and Outport, respectively. There are no specifications allowed on bus element port blocks, which support inherited workflows. You cannot use the Block Parameters dialog box of a bus element port block to specify bus element attributes, such as data type or dimensions.

To work with buses at subsystem interfaces, consider using In Bus Element and OutBusElement blocks. This bus element port block combination:

- Reduces signal line complexity and clutter in a block diagram.
- Makes it easy to change the interface incrementally.
- Allows access to a bus element closer to the point of usage.

  - For input, avoid a duplicate Inport blocks and a Bus Selector, Goto, and From block configuration.
  - For output, avoid a Goto, From, and Bus Creator block configuration.

This model uses Inport, Bus Selector, Bus Creator, and Outport blocks.
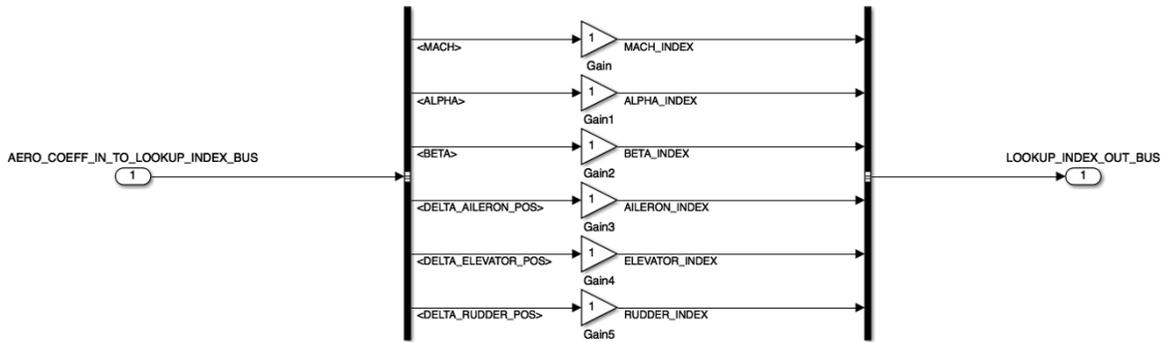
Here is an equivalent model using bus element port blocks.
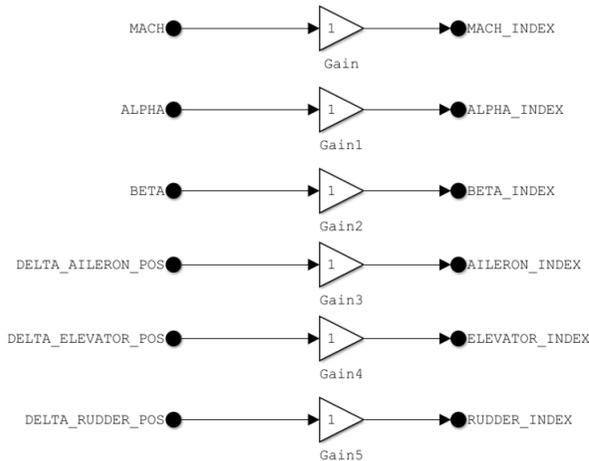
You can refactor a subsystem interface that has Inport, Bus Selector, Bus Creator, and Outport blocks to use In Bus Element and Out Bus Element blocks. Conversions are supported only when the signal lines or blocks do not have any extra specifications. You can use single-click operations to convert:

- Inport and Bus Selector blocks in a subsystem to In Bus Element blocks.
- Outport and Bus Creator blocks in a subsystem to Out Bus Element blocks.

To transform input or output interfaces of subsystems to use bus element port block, you can use marquee selection options.

For more information, see "Simplify Subsystem Bus Interfaces".

Bus and Vector Mixtures Not Supported

A mixtures of bus and vector (mux) signals occurs when some blocks treat a signal as a vector, while other blocks treat that same signal as a bus. Mixing bus and vector signals in a model causes your model to be less robust. Configuring your model to prevent bus and vector mixtures:

- Improves loop handling
- Produces clear error messages
- Contributes to consistent edit and compile-time behavior

To check a model for bus signals that are used as vectors, use the new Model Advisor **"Check bus signals treated as vectors"** check.

Compatibility Considerations

In R2017a, the Demux block parameter **Bus selection mode** (`BusSelectionMode`) is no longer supported.

In R2017a, these checks and diagnostics are no longer available:

- In the Model Advisor, the **Check bus usage** check
- In the Upgrade Advisor, the **Check Mux blocks that create bus signals** check
- The **Configuration Parameters** > **Diagnostics** > **Connectivity** > **Mux blocks used to create bus signals** parameter

For R2017a, to handle a legacy model that contains vector and bus mixtures:

1 In R2016b, run the Upgrade Advisor with the **Check Mux blocks that create bus signals** check enabled.
2 In R2017a, open the upgraded model.

You can continue to use existing `slreplace_mux` function to replace with Bus Creator blocks any Mux blocks are used to create buses. However, in a future release, the `slreplace_mux` function could be removed.

Inline Variants: Single-Input/Single-Output Inline Variant blocks support zero active variant control

A variant model with Single-Input/Single-Output Inline Variant block can now be simulated without an active variant choice. You can use the **Allow zero active variant controls** option to specify the choice.

For more information on Inline Variants, see "Define, Configure, and Activate Variants".

Searchable, sortable tables for parameterizing reusable models with model arguments

Before R2017a, you used comma-separated lists to identify model arguments and specify argument values for reusable referenced models.

In R2017a:

- To identify variables in a model workspace as model arguments, in the Model Explorer **Contents** pane, use the **Argument** check box instead of a comma-separated list.
- To specify argument values in a Model block, use a searchable, sortable table instead of a comma-separated list.

Now, when you set the programmatic parameter of the Model block, `ParameterArgumentValues`, you can use a structure instead of a comma-separated list. See "Parameterize Instances of a Reusable Referenced Model".

Compatibility Considerations

If you use the **Argument** check box in the Model Explorer, you must modify scripts that manipulate the `ParameterArgumentValues` parameter of referencing Model blocks. Make the scripts set `ParameterArgumentValues` by using structures instead of comma-separated lists. If you do not modify the scripts, they generate errors while trying to use comma-separated lists.

Project and File Management

Simulink Project Upgrade: Easily update all the models in your Simulink Project to the latest release

Easily upgrade all the models in your project using the Upgrade Project tool in Simulink Project, by selecting **Run Checks** > **Upgrade**. You can upgrade all models in your project to the latest release using a simple workflow. The tool can apply all fixes automatically when possible, upgrade all model hierarchies in the project at once, and produce a report. You do not need to open the Upgrade Advisor.

For details, see "Upgrade All Project Models".

Missing Product Identification: Fix models with unresolved library links and unknown block types by finding and installing missing products

Simulink now tries to help you find and install missing products that a model needs to run. If you open a model that contains built-in blocks or library links from missing products, you see labels and links to help you fix the problem.

- Blocks are labeled with missing products (for example, **SimEvents not installed**)
- Tooltips include the name of the missing product
- Messages provide links to open Add-On Explorer and install the missing products
- Simulink Project dependency analysis reports missing products required by a project.

To find a link to open Add-On Explorer and install the product:

- For built-in blocks, open the Diagnostic Viewer, and click the link in the warning.
- For unresolved library links, double-click the block to view details and click the link.
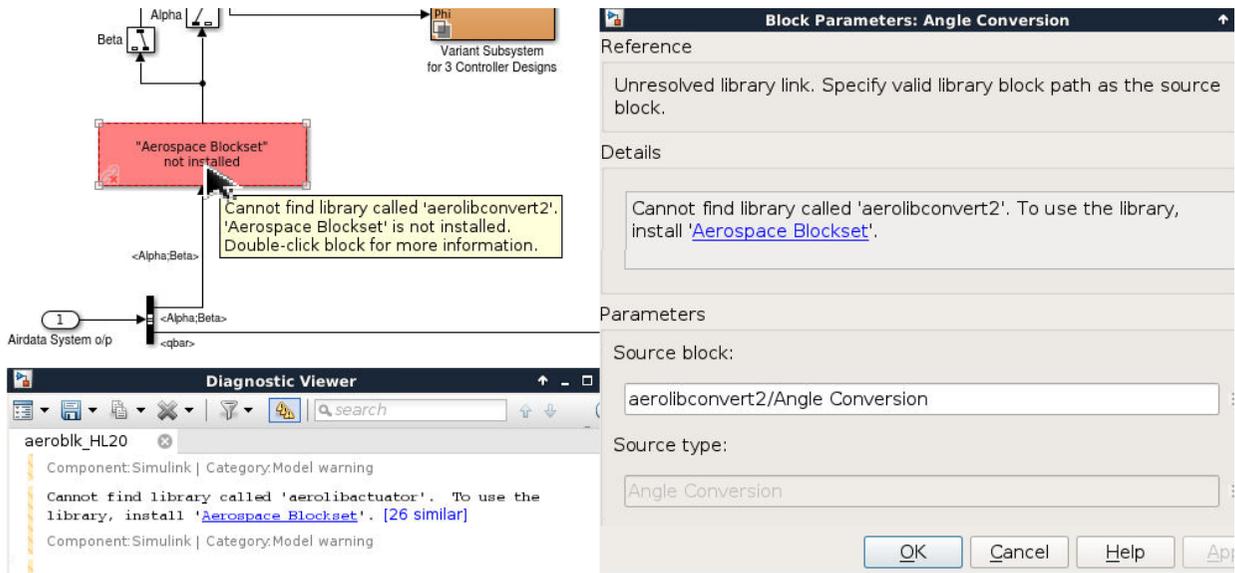
For details, see the Unresolved Link block reference page, and "Check Dependency Results and Resolve Problems".

Git Pull: Fetch and merge in one step

You can now use Pull for Git™ from the Simulink Project toolstrip tab. Pull fetches the latest changes and merges them into your current branch. Previously, you had to fetch and merge separately before you could see changes.

For details, see "Pull, Push, and Fetch Files with Git".

Project Creation API: Set up projects programmatically, including shortcuts and referenced projects

New functions enable you to create Simulink projects programmatically. You can set up referenced projects, shortcuts, and the path. Previously you had to use Simulink Project to set up projects interactively. New functions:

- `slproject.create`
- `addPath`
- `removePath`

- `addReference`
- `removeReference`
- `addShortcut`
- `removeShortcut`

For an example, see "Creating Simulink Projects Programmatically".

Referenced Project Change Management: Compare components with checkpoints

You can create a checkpoint for a referenced project. You can then compare the referenced project against the checkpoint to detect any changes.

For details, see "Manage Referenced Project Changes Using Checkpoints".

Source Control Toolstrip: Simplified workflow for working with source control

You can now use the Simulink Project toolstrip tab to perform source control action. Built-in SVN or Git and other source control integrations now have toolstrip buttons for operations such as Commit, Update, Push, Pull, and other actions.

For details, see "Source Control in Simulink Project".

Custom Task Tool: Improved interface for managing custom tasks and creating reports from results

You can run custom tasks on files in Simulink Project with a simplified workflow. The new custom task tool makes it easier to select files, run tasks, and create and save reports. Previously you had to select the Batch Job node in the project tree to run custom tasks.

For details, see "Create a Custom Task Function".

Git Remote Repositories: Connect existing project to a remote repository

In an existing Simulink project under local Git source control, you can now specify a remote repository for the project by clicking Remote on the Simulink Project tab. Previously you could only specify a remote Git repository when you created the project.

For details, see "Add a Project to Git Source Control".

Start Page Example Search: Find featured examples

You can now search for examples in the Simulink start page examples tab. Enter search terms to find examples titles and descriptions of interest, or open further examples on the web.

For details, see "Create a Model".

Model Templates: Simplified workflow for exporting models to templates

Exporting models to templates is simplified and you can add thumbnail images for your templates. The dialog box now prepopulates fields with model information that you can edit if needed.

For details, see "Create a Template from a Model".

Start Page Favorites: Easily get back to your favorite models and projects

In the Simulink start page recent files list, you can add files to favorites. The Favorites list then appears above recent files in the start page, so that you can easily reopen your favorite models and projects.
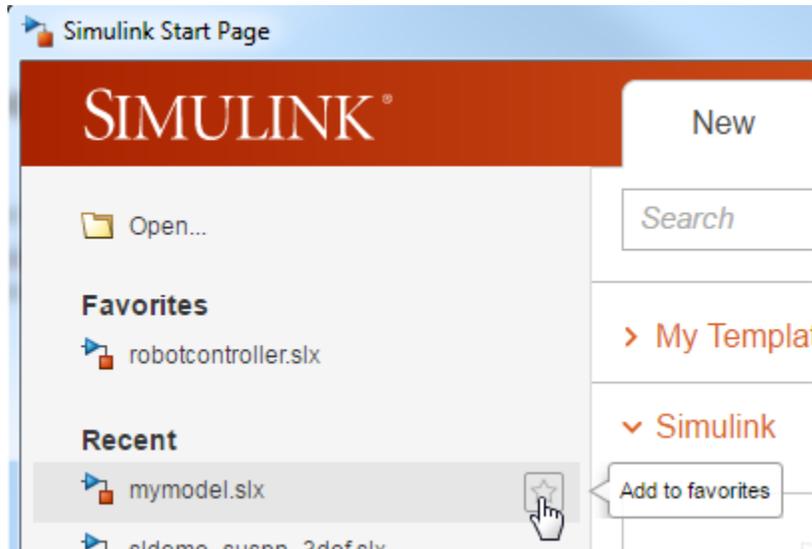
For details, see "Open a Model".

Project Componentization: Include referenced projects in templates for sharing components

When you make a template from a project, you can now include referenced projects so that template users can get all the components. You can choose whether to share references together with main project.

For details, see "Create a Template from the Current Project".

bdIsDirty Function: Programmatically check whether models contain unsaved changes

You can use the new bdIsDirty function to check whether or not loaded block diagrams contain unsaved changes.

For details, see bdIsDirty.

`listRequiredFiles` Function: Get project file dependencies programmatically

You can use the new `listRequiredFiles` function to get the downstream dependencies of a file in a Simulink Project. The function returns the files that the specified file requires to run.

For details, see `listRequiredFiles`.

Data Management

Simulation Data: Easily access simulation output data in the MATLAB Variable Editor and MATLAB Command Window

You can use the Variable Editor to inspect simulation data that is stored as `Simulink.SimulationData.Dataset` or `Simulink.SimulationOutput` objects. Also, for `SimulationOutput` objects, you can view simulation metadata that provides context about the simulation, such as information about the model. For details about the Variable Editor, see "Editing Workspace Variables".

In the MATLAB Command Window

, you can use dot notation to access an element of a `SimulationOutput` object. For example, to return the `xout` element of a `SimulationOutput` object called `simout`, you can enter:

```
xout = simout.yout;
```

Before R2017a, you had to use a `get` function (which continues to work in R2017a):

```
xout = simout.get('yout');
```

Management of workspace variables and mask parameters from block parameters

Before R2017a, you could create or navigate to a workspace variable by right-clicking the value of a block parameter, for example, the **Gain** parameter of a Gain block.

In R2017a, right-clicking a parameter value does not initiate creation or navigation.

Instead, you left-click the new button ⋮ next to the value of the block parameter. The button appears in block dialog boxes, the Property Inspector, the Model Data Editor, and the Model Explorer. For more information, see "Manage Variables from Block Parameters".

- In the Model Data Editor, on the **Parameters** tab, the **Defined In** column no longer appears. To navigate to a workspace variable from a block parameter that appears on the **Parameters** tab, use the new button.

- In the Model Explorer, when you edit the value of a workspace variable or parameter object, the **Open Variable Editor** button no longer appears. To open the Variable Editor, use the new button.

Association of root-level Outport block with `Simulink.Signal` object

Before R2017a, you could not associate a root-level Outport block with a `Simulink.Signal` object.

In R2017a, you can use the Model Data Editor (see "Configure Data Properties by Using a Table") to make this association.

Initial State: Log and load initial states using Dataset format

If you set the `SaveFormat` model parameter to `'Dataset'`, the `Simulink.BlockDiagram.getInitialState` function returns the initial state as a `Simulink.SimulationData.Dataset` object.

To load an initial state, you can specify a `Dataset` object for the **Data Import/Export** > **Initial state** configuration parameter.

Root Inport Mapper Tool Updates

The tool has the following updates:

- Root Inport Mapping has been renamed to Root Inport Mapper.
- To use strong data typing when mapping data from spreadsheets, select **Options** > **Use Strong Data Typing with Spreadsheets**. For more information, see "Set Options for Mapping".

Legacy Code Tool `StartFcnSpec` and `InitializeConditionsFcnSpec` accept outputs as arguments

Previously, when you specified the `StartFcnSpec` and `InitializeConditionsFcnSpec` functions of the Legacy Code Tool data structure, the functions could not accept outputs as arguments. In R2017a, these functions can access output ports if the S-Function option `outputsConditionallyWritten` is set to `true`. With this option setting, the generated S-Function specifies that

1-25

the memory associated with each output port cannot be overwritten and is global (SS_NOT_REUSABLE_AND_GLOBAL).

Utility to generate Simulink representations of custom data types defined by external C code

Before R2017a, to integrate your external C code into Simulink by using an S-function, you needed to manually create:

- Simulink.Bus objects to represent the structure types (struct) that your code defined. For example, you could use the Bus Editor.
- Enumeration definitions to represent the enumerated types (enum) that your code defined. For example, you could use the function Simulink.defineIntEnumType.
- Simulink.AliasType objects to represent primitive typedef statements in your code.

These manual techniques led to data entry errors. Also, maintaining the objects and definitions in Simulink was difficult.

In R2017a, you can use a function, Simulink.importExternalCTypes, that parses your external header files for struct, enum, and typedef type definitions and generates corresponding objects and enumeration classes.

Direct representation of fixed-point data types by **Simulink.AliasType**

A fixed-point data type consists of a word length and a scaling (for example, a 16-bit word length with a 14-bit binary fraction length).

Before R2017a, to specify a custom name for a fixed-point data type, you used one of these techniques that involve Simulink.NumericType objects:

- Use both Simulink.NumericType and Simulink.AliasType objects.

  1 Create a Simulink.NumericType object in the base workspace or a data dictionary.

  2 Configure the Simulink.NumericType object to represent the fixed-point data type.

  3 Create a Simulink.AliasType object.

    4   Configure the `Simulink.AliasType` object to use the `Simulink.NumericType` object as a base data type.

- Use a standalone `Simulink.NumericType` object.

    1   Create a `Simulink.NumericType` object in the base workspace or a data dictionary.

    2   Configure the `Simulink.NumericType` object to represent the fixed-point data type.

    3   Set the `IsAlias` property of the object to `true`.

In R2017a, a `Simulink.AliasType` object can directly represent a fixed-point data type. You can use `Simulink.AliasType` to rename numeric data types including integer, floating-point, and fixed-point types. You do not need to create a `Simulink.NumericType` object.

Display of alias, base, or both data types in a model

Before R2017a, when you displayed signal data types on a block diagram by selecting **Display** > **Signals and Ports** > **Port Data Types**, you were not able to control the display of data type aliases. For example, if you used a `Simulink.AliasType` object to set the data type of a signal, the diagram displayed only the alias. The diagram did not display the underlying base data type (such as `int16`).

In R2017a, you can choose whether to display data type aliases, base data types, or both for all of the signals in a model. In the model, choose an option for **Display** > **Signals and Ports** > **Port Data Type Display Format**. You can temporarily or permanently adjust this setting, which is saved with the model file, to:

- Control the appearance of the model to users who do not need to know the underlying numeric data types.
- Inspect the numeric data types that signals use on a high level, especially while designing and debugging fixed-point models.

More accurate comparison of nondouble data to specified minimum and maximum values

You can specify minimum and maximum values for signals and block parameters in a model. For example, in a Constant block, you can set **Output minimum** and **Output**

maximum to specify design limits for the block output signal. You must set such a design limit by using a literal number (with implicit data type `double`) or an expression that yields a `double` number.

Before R2017a, if the data item (signal or block parameter) used a data type other than `double`, Simulink:

1   Cast the nondouble value of the data item to `double`.

2   Compared the `double` value of the data item to the `double` value of the design limit.

If the storage and typecasts of the design limit or the data item incurred losses of accuracy (quantization) or information (overflow), the comparison could unexpectedly report a violation of the design limit.

In R2017a, before comparison, Simulink casts the data item and the design limit to the same data type (the data type of the data item). Simulink does not cast the data item to `double` unless you specify it. This more accurate technique can prevent the generation of unnecessary, misleading errors and warnings. However, Simulink still stores the design limit as `double` before comparison.

Compatibility Considerations

The more accurate comparison technique can cause new warnings or errors in R2017a when you:

- Simulate an existing model.
- Run an existing script that sets the value, minimum, or maximum of a block parameter in a model.
- Run an existing script that sets the `Value`, `Min`, or `Max` properties of a parameter object (such as `Simulink.Parameter`).

For example, the R2016b comparison technique could have caused Simulink to overlook limit violations when you used data types that have greater precision or range than `double`. In R2017a, the new comparison technique can cause these limit violations to generate warnings or errors.

- Use the Fixed-Point Tool or the Data Type Assistant to autoscale data items in a fixed-point model. For each data item, these tools can propose a scaling that enables the data item to represent the real-world minimum and maximum values that you

specify. See "Fixed-Point Tool" (Fixed-Point Designer) and "Specify Fixed-Point Data Types".

- Reduce the number of errors that you encounter while interacting with the model by:

  - Using the Fixed-Point Tool to temporarily override data types to `double`. See "Control Data Type Override".

  - Adjusting the setting for the configuration parameter **Simulation range checking** from `error` to `warning` or `none`.

- Round a design limit to the next number furthest from zero that `double` can represent. This technique can resolve a limit violation when the data type of the data item has higher precision than `double` (for example, a fixed-point data type with a 128-bit word length and a 126-bit fraction length) and `double` cannot exactly represent the value of the design limit.

  For example, if an existing model generates a new error in R2017a with the maximum value of a signal set to `98.8847692348509014`, at the command prompt, calculate the next number furthest from zero that `double` can represent.

  ```
  format long
  98.8847692348509014 + eps(98.8847692348509014)

  ans =

    98.884769234850921
  ```

  Use the resulting number, `98.884769234850921`, to replace the maximum value.

Deep copy of handle objects by `Simulink.ModelWorkspace.assignin`

Before R2017a, the `assignin` method of a `Simulink.ModelWorkspace` object did not deeply copy handle objects. As a result, modifications that you made to a handle object in the source workspace (for example, the base workspace or a function workspace) also affected the model workspace.

Suppose you used this code to create a `Simulink.Parameter` object named `myVar` and assign `myVar` into the model workspace of a model named `myModel`:

```
myVar = Simulink.Parameter(5.2);
mdlwks = get_param('myModel','ModelWorkspace');
```

```
assignin(mdlwks,'myVar',myVar);
```

Then, if you modified `myVar` in either the base workspace or the model workspace, the modification affected `myVar` in both workspaces. Each instance of `myVar` stored a handle to the same `Simulink.Parameter` object.

In R2017a, the `assignin` method performs a deep copy so modifications that you make in the source workspace do not affect the model workspace. With this deep copy, in the example, `myVar` in the model workspace is independent of `myVar` in the base workspace.

Use of From Workspace block in a model that uses a data dictionary

Prior to R2017a, if you linked a model to a data dictionary, the model could not contain any From Workspace blocks.

In R2017a, when a model is linked to a data dictionary, From Workspace blocks can acquire input data from variables that you store in the Design Data section of the dictionary or in the other workspaces that From Workspace already supports. To use variables that you store in the other workspaces, set the value of the **Data** parameter by using a call to the `evalin` function.

Because you cannot store a `timeseries` object in the dictionary, to drive the From Workspace block by using a `timeseries` object, you must:

1   Place the object in the base workspace.
2   In the From Workspace block, set the value of the **Data** parameter to, for example, `evalin('base','myTimeseriesObject')`. The argument `'base'` indicates that the object is in the base workspace.

For more information, see From Workspace

Specify 64-bit integer data types without a Fixed-Point Designer license

Beginning in R2017a, specifying a 64-bit integer data type no longer requires a Fixed-Point Designer™ license. To specify a 64-bit integer type, use the `fixdt` function. For example, to specify an `int64` or `uint64` data type, set the output data type of a block to `fixdt(1,64,0)`, or to `fixdt(0,64,0)`, respectively.

Block Enhancements

Support for Scopes in For Each Subsystems

You can place a Scope block within a For Each Subsystem block.

If your model has a Scope block attached to the output port of a For Each Subsystem block, you can move the Scope block into the subsystem block, attach signal lines, and then delete the Outport block.

Scope Blocks: Support for nonvirtual bus and array of buses signals

You can connect nonvirtual bus signals and array of buses signals to a Scope block (and to a Time Scope block if you have a DSP System Toolbox™ license). To display the signals in the Scope block, use normal or accelerator simulation mode. For details, see "Nonvirtual Bus and Array of Buses Signals" and "Save Simulation Data Using a Scope Block".

Specify image file icons for MATLAB System block

You can specify a MATLAB System block icon as an image file using a new option in the MATLAB Editor. While editing your System object™, specify the image file by selecting **System Block** > **Add Image Icon**. After specifying the image file, this code is added to the System object class:

```
function icon = getIconImpl(~)
    % Define icon for System block
    icon = matlab.system.display.Icon('image.png');
end
```

For more information, see "Customize System Block Appearance".

Copy scope to clipboard

To share the output of a signal simulation, copy the scope graphic to your clipboard by selecting **File** > **Copy to Clipboard**. The scope colors are converted to a print-friendly coloring. See "Share Scope Image".

Interactive legend for scopes

If you show a legend on your scope block or System object, you can use the legend to filter which signals are shown. Left-clicking a signal in the legend hides all other signals in the scope. Right-clicking a signal in the legend toggles whether the scope shows or hides the signal.

Stem plot option for Scope block

In the Scope block, you can visualize your signal as a stem plot. From the **View** > **Style** menu, select **Plot type** > **Stem**.

Simulink Blocks: Simulink implements same workflow when adding a block through the user interface or the command line

When you add a block through the command line, Simulink now executes the block copy function first and then sets the specified parameter values for the block.

Before R2017a, when you add a block through the command line, the parameter values are set first and then the block copy function is executed.

Slider Gain block: Minimum and Maximum values must not be same

In R2017a and higher, the minimum values specified for the Slider Gain block must be less than the maximum value specified. The minimum and maximum values must not be same.

Default input signal attributes for MATLAB System block

In R2017a and higher, the default input signal attributes are defined if a MATLAB System block has one or more inputs that are unconnected to another block's output port or connected to a port that has underspecified attributes.

Additional calls to Propagation Methods `getOutputDataTypeImpl`, `getOutputSizeImpl` and `isOutputComplexImpl` during the model pre-compile phase

In addition to the model compile phase, propagation methods `getOutputSizeImpl`, `getOutputDataTypeImpl` and `isOutputcomplexImpl` are called during the model pre-compile phase.

Math Function block `rem`, `mod`, and `pow` function changes

The Math Function block `rem` and `mod` functions have these changes:

- For generated code, Simulink now applies output signedness for the `rem` and `mod` functions when the output is 0.

- You may experience improved performance when using the `rem` and `mod` functions.

The Math Function block `pow` function has this change:

- For generated code and simulation, the `pow` function now returns `1.0` for these cases:

  - `1^inf`
  - `(-1)^inf`
  - `(1)^(-inf)`
  - `(-1)^(-inf)`

  In previous releases, the `pow` function returned `NaN` for these cases.

Trigonometric Function block `asin`, `asinh`, `acos`, and `acosh` function changes

The Trigonometric Function block `asin`, `asinh`, `acos`, and `acosh` functions now perform correct branch cut behavior for generated code and simulation. In previous releases:

- The `asin` and `acos` functions returned `NaN` in the real part of a complex number.

- The `asinh` and `acosh` functions returned `NaN` in the imaginary part of a complex.

Dynamic memory allocation for unbounded arrays and large arrays

In R2017a, simulation and C/C++ code generation support dynamic memory allocation for arrays in a System object associated with a MATLAB System block. Dynamic memory allocation allocates memory as needed at run time, instead of allocating memory statically on the stack. Dynamic memory allocation is beneficial when:

- You do not know the upper bound of an array.
- You do not want to allocate memory on the stack for large arrays.

By default, dynamic memory allocation is enabled. To disable it, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, clear the **Dynamic memory allocation in MATLAB Function blocks** check box.

When dynamic memory allocation is enabled, the code generator uses dynamic memory allocation for arrays whose size is equal to or greater than a threshold. The default value of this threshold is 64 kilobytes. To change the threshold, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, set the **Dynamic memory allocation threshold in MATLAB Function blocks** parameter.

Dynamic memory allocation does not apply to:

- Input and output signals. Variable-size input and output signals must have an upper bound.
- Parameters or global variables. Parameters and global variables must be fixed-size.
- Fields of bus arrays. Bus arrays cannot have variable-size fields.
- Discrete state properties of System objects associated with a MATLAB System block.

See "Control Memory Allocation for Variable-Size Arrays in a MATLAB Function Block" and "Use Dynamic Memory Allocation for Variable-Size Arrays in a MATLAB Function Block".

Better handling of promoted parameter

From R2017a, any change to the path of a promoted mask parameter is resolved automatically. If the underlying block of the promoted mask parameter is deleted or is moved within another mask, Simulink displays a warning during model load, model simulation, and while using the set_param command on the promoted parameter.

Connection to Hardware

Wireless Connectivity: Use UDP and TCP/IP blocks to let Simulink hardware targets communicate with each other

The following support packages now support wireless communication using UDP and TCP/IP blocks:

- Simulink Support Package for Android™ Devices
- Simulink Support Package for Apple iOS Devices
- Simulink Support Package for Arduino® Hardware
- Simulink Support Package for LEGO® MINDSTORMS® NXT Hardware
- Simulink Support Package for Raspberry Pi™ Hardware

When you add UDP or TCP/IP blocks to models created in two of these support packages, the models can communicate and transfer of data to each other directly during run time on the hardware.

Support for print and println on Arduino Serial Transmit block

The Serial Transmit block now supports the print and println options so that you can print data to the serial port.

Hardware plugin detection for Arduino boards in MATLAB, Simulink

Hardware plugin detection feature helps you to identify the devices that you can use within MATLAB and Simulink when you plug in a new hardware into the host computer.

Blocks added to LEGO EV3 support package

This table lists the support for these new blocks.

Block	Usage
TCP/IP Receive	Receive TCP packets from a remote host.
TCP/IP Send	Send TCP packets to a remote host.
UDP Receive	Receive UDP packets from a remote host.

Block	Usage
UDP Send	Send UDP packets to a remote host.

Blocks added to Raspberry Pi support package

This table lists the support for these new blocks.

Block	Usage
I2C Master Write	Write data to I2C slave device or I2C slave device register.
I2C Master Read	Read data from I2C slave device or I2C slave device register.
SPI Master Transfer	Write data to and read data from SPI slave device.
SPI Register Write	Write data to registers of an SPI slave device.
SPI Register Read	Read data from registers of SPI slave device.
Serial Write	Write data to serial device.
Serial Read	Read data from serial device.
TCP/IP Receive	Receive TCP packets from a remote host.
TCP/IP Send	Send TCP packets to a remote host.
HTS221 Humidity Sensor	Measure relative humidity and ambient temperature (Sense HAT block).
LPS25h Pressure Sensor	Measure barometric air pressure and ambient temperature (Sense HAT block).
LSM9DS1 IMU Sensor	Measure linear acceleration, angular rate and magnetic field along X, Y, and Z axis (Sense HAT block).
Joystick	Read the state of five-position joystick (Sense HAT block).
8x8 RGB LED Matrix	Control pixel color of 8x8 RGB LED Matrix (Sense HAT block).

Support for all Android smartphones and tablets

The Simulink Support Package for Android Devices supports all Android smartphones and tablets using Android version 4.2 and higher.

Blocks added to Android support package

This table lists the new blocks available in the Simulink Support Package for Android Devices.

Block	Usage
TCP/IP Send	Send TCP packets to a remote host.
TCP/IP Receive	Receive TCP packets from a remote host.
Audio File Read	Use an audio file from your desktop computer on your mobile app.

Blocks added to Apple iOS support package

This table lists the new blocks available in the Simulink Support Package for Apple iOS Devices.

Block	Usage
TCP/IP Send	Send TCP packets to a remote host.
TCP/IP Receive	Receive TCP packets from a remote host.
Audio File Read	Use an audio file from your desktop computer on your mobile app.

Support for Scope block on Apple iOS and Android apps

The following support packages now support the Scope block:

- Simulink Support Package for Android Devices
- Simulink Support Package for Apple iOS Devices

When you add a Scope block to a model created using one of these support packages, an equivalent real-time scope display is added to the mobile app user interface.

MATLAB Function Blocks

Dynamic memory allocation for unbounded arrays and large arrays

In R2017a, simulation and C/C++ code generation support dynamic memory allocation for arrays in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block. Dynamic memory allocation allocates memory as needed at run time, instead of allocating memory statically on the stack. Dynamic memory allocation is beneficial when:

- You do not know the upper bound of an array.
- You do not want to allocate memory on the stack for large arrays.

By default, dynamic memory allocation is enabled for GRT-based targets and disabled for ERT-based targets. To change the setting, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, clear or select the **Dynamic memory allocation in MATLAB Function blocks** check box.

When dynamic memory allocation is enabled, the code generator uses dynamic memory allocation for arrays whose size is equal to or greater than a threshold. The default value of this threshold is 64 kilobytes. To change the threshold, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, set the **Dynamic memory allocation threshold in MATLAB Function blocks** parameter.

Dynamic memory allocation does not apply to:

- Input and output signals. Variable-size input and output signals must have an upper bound.
- Parameters or global variables. Parameters and global variables must be fixed-size.
- Fields of bus arrays. Bus arrays cannot have variable-size fields.
- Discrete state properties of System objects associated with a MATLAB System block.

See "Control Memory Allocation for Variable-Size Arrays in a MATLAB Function Block" and "Use Dynamic Memory Allocation for Variable-Size Arrays in a MATLAB Function Block".

Nested functions

In R2017a, you can use nested functions in MATLAB code in a MATLAB Function block. When you use nested functions, adhere to these restrictions:

- If the parent function declares a persistent variable, it must assign the persistent variable before it calls a nested function that uses the persistent variable.
- A nested recursive function cannot refer to a variable that the parent function uses.
- If a nested function refers to a structure variable, you must define the structure by using `struct`.
- If a nested function uses a variable defined by the parent function, you cannot use `coder.varsize` with the variable in either the parent or the nested function.
- You cannot use nested functions in MATLAB action language used by a Stateflow chart.

Also, you must adhere to the code generation restrictions for value classes and handle classes.

Handle classes in value classes

In R2017a, in MATLAB code in a MATLAB Function block, you can use value classes that contain handle classes. The handle class can be one that you define or a predefined handle class that is available with MATLAB or a MATLAB toolbox. Predefined handle classes, such as toolbox System objects, must be supported for C/C++ code generation. See "Functions and Objects Supported for C/C++ Code Generation — Category List".

For example, suppose that `myclass` is a value class and `myhandle` is a handle class. A MATLAB Function block can contain code such as:

```
obj = myclass;
obj.p1 = myhandle;
obj.p2 = dsp.Mean;
```

The code generation limitations for handle class objects apply to handle class objects in value classes. See Code generation limitations for handle class objects.

Constant folding of value classes

In R2017a, you can use `coder.const` to constant-fold value classes.

The code generator tries to fold constant expressions into the generated code. Constant folding uses the value of a constant expression instead of the expression in the generated code. Constant folding can improve execution time because the generated code does not have to evaluate the expression multiple times. You can try to force the code generator to constant- fold an expression by using `coder.const`.

To constant-fold a value class object `obj`, use this syntax:

```
coder.const(obj)
```

To constant-fold the property `prop`, use this syntax:

```
coder.const(obj.prop)
```

Class properties and structure fields passed by reference to external C functions

To pass arguments by reference to an external C function, you use `coder.ref`, `coder.rref`, or `coder.wref` in a `coder.ceval` call. For example:

```
...
x = 1;
y = coder.ceval('myCFunction', coder.ref(x));
...
```

In previous releases, the argument that you passed by reference had to be a scalar variable or an element of an array. To pass a class property or structure field, you had to first assign the property or field to a variable. For example:

```
...
x = myClass;
x.prop = 1;
v = x;
coder.ceval('foo', coder.ref(v));
...
```

In R2017a, you can directly pass a class property or structure field by reference. For example:

- Pass a class property

  ```
  ...
  x = myClass;
  ```

```
x.prop = 1;
coder.ceval('foo', coder.ref(x.prop));
...
```

- Pass a structure field

```
...
s = struct('s1', struct('a', [0 1]));
coder.ceval('foo', coder.wref(s.s1.a));
...
```

- Pass a field of an element of an array of structures

```
...
s = struct('c', [1 2], 'd', 2);
s1 = struct('a', [s s]);
coder.ceval('foo', coder.rref(s1.a(1).d));
...
```

Function specialization prevention with `coder.ignoreConst`

At compile time, if an input argument to a function call evaluates to a constant, the code generator can use the constant value to produce function specializations. A function specialization is a version of a function in which the input type, size, complexity, or value is customized for a particular invocation of the function. To prevent function specializations due to constant arguments, instruct the code generator to treat the value of the argument as a nonconstant value by using `coder.ignoreConst`.

With compile-time recursion, the code generator produces function specializations instead of a recursive call. If the specializations are due to a constant input argument to the recursive function, you might be able to force run-time recursion by using `coder.ignoreConst`. See "Force Code Generator to Use Run-Time Recursion".

New `coder.unroll` syntax for more readable code

In R2017a, `coder.unroll` has a new syntax that helps make your code more readable.

In previous releases, you put `coder.unroll` inside a `for`-loop. For example:

```
...
for i = coder.unroll(1:n)
    y(i) = rand();
end
```

```
...
```

With the new syntax, you put `coder.unroll` on a line by itself, immediately before the loop that it unrolls. For example:

```
...
coder.unroll();
for i = 1:n
    y(i) = rand();
end
...
```

Here is an example of the new syntax with the `flag` argument:

```
...
unrollflag = n < 10;
coder.unroll(unrollflag);
for i = 1:n
    y(i) = rand();
end
...
```

Both the new syntaxes and the syntaxes from previous releases are supported. For more readable code, use the new syntax.

For more information about `coder.unroll` and for-loop unrolling, see `coder.unroll` and "Unroll for-Loops".

Size argument for coder.opaque

In R2017a, you can specify the size of a variable that you declare with `coder.opaque`. The syntax with the size argument is:

```
x = coder.opaque(type,value,'Size', size)
```

Specify the size in bytes. For example, declare `x1` to be a 4-byte integer with initial value 0.

```
x1 = coder.opaque('int','0', 'Size', 4);
```

Code generation for more MATLAB functions

• `cholupdate`

- `histcounts`
- `ismethod`

Code generation for more Audio System Toolbox System objects

audioPlayerRecorder

For C/C++ code generation usage notes and limitations, see the reference page.

Code generation for more Communications System Toolbox System objects

comm.RBDSWaveformGenerator

For C/C++ code generation usage notes and limitations, see the reference page.

Code generation for more DSP System Toolbox System objects

- dsp.HampelFilter
- dsp.AsyncBuffer

For C/C++ code generation usage notes and limitations, see the System object reference page.

Code generation for more Phased Array System Toolbox System objects

- `bw2range`
- `diagbfweights`
- `scatteringchanmtx`
- `waterfill`
- phased.BackScatterSonarTarget
- phased.DopplerEstimator
- phased.IsoSpeedUnderWaterPaths
- phased.IsotropicHydrophone
- phased.IsotropicProjector

- phased.MultipathChannel
- phased.RangeEstimator
- phased.RangeResponse
- phased.ScatteringMIMOChannel

For C/C++ code generation usage notes and limitations, see the function or System object reference page.

Code generation for more Robotics System Toolbox functions and classes

- robotics.AimingConstraint
- robotics.Cartesianbounds
- robotics.GeneralizedInverseKinematics
- robotics.InverseKinematics
- robotics.Joint
- robotics.JointPositionBounds
- robotics.PoseTarget
- robotics.PositionTarget
- robotics.OrientationTarget
- robotics.RigidBody
- robotics.RigidBodyTree
- transformScan

For C/C++ code generation usage notes and limitations, see the function or class reference page.

Code generation for more Signal Processing Toolbox functions

- alignsignals
- cconv
- convmtx
- corrmtx
- envelope
- finddelay

- `hilbert`
- `sgolayfilt`
- `sinc`
- `xcorr2`
- `xcov`

For C/C++ code generation usage notes and limitations, see the function or class reference page.

Statistics and Machine Learning Toolbox Code Generation: Generate C code for prediction by using linear models, generalized linear models, decision trees and ensembles of classification trees

You can generate C code that predicts responses by using trained linear models, generalized linear models (GLM), decision trees, or ensembles of classification trees. The following prediction functions support code generation.

- predict — Predict responses or estimate confidence intervals on predictions by applying a linear model to new predictor data.
- predict or `glmval` — Predict responses or estimate confidence intervals on predictions by applying a GLM to new predictor data.
- predict or predict — Classify observations or estimate classification scores by applying a classification tree or ensemble of classification trees, respectively, to new data.
- predict — Predict responses by applying a regression tree to new data.

For C/C++ code generation usage notes and limitations, see the function reference page.

Enhancement to synchronous subsystem support

For a MATLAB Function block inside a synchronous subsystem, you can now use the combinational and sequential logic portions in one MATLAB function. Previously, you created two separate MATLAB Function blocks, one for the combinational logic, and the other for the sequential logic.

To use the combinational and sequential logic portions inside one MATLAB Function block, in the Ports and Data Manager dialog box, select the **Allow direct feedthrough** checkbox. The output function can then depend on inputs and persistent variables. For

example, you can now use this MATLAB function that has two outputs, with one output depending on the input, and the other output depending on a persistent variable.

```
function [y1, y2] = fcn(u, v)

persistent p;
if isempty(p)
    p = uint8(0);
end

y1 = p;
y2 = v;

p = u;
```

If you have HDL Coder™, you can use the MATLAB Function block inside a synchronous subsystem to generate cleaner HDL code and use fewer hardware resources. See also State Control.

Support for tunable structure array parameters

You can now use a tunable structure array parameter with the MATLAB Function block. Previously, you used a tunable structure of scalar values or nontunable parameters with structure arrays.

State behavior specification for function-call input events

If you define a function-call input event for a MATLAB Function block, you can now specify the state behavior when this event reenables the block. To specify this behavior, use the **States When Enabling** block parameter located in either:

- The **Properties** > **Advanced** section of the Property Inspector
- The Ports and Data Manager dialog box

When you set **States When Enabling** to held, the simulation maintains the most recent values of the states when the function-call event reenables the MATLAB Function block. If you set **States When Enabling** to reset, the function-call event reverts states to their initial conditions.

S-Functions

Functionality being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
Level-1 Fortran S-Functions	Still runs.	Level-2 Fortran S-Functions	Level-1 Fortran S-Functions will continue to work but no documentation support will be provided. Use the Level-2 Fortran S-Functions instead.

R2016b

Version: 8.8

New Features

Bug Fixes

Compatibility Considerations

Simulation Analysis and Performance

Just-in-Time Acceleration Builds: Quickly build the top-level model for improved performance when running simulations in Accelerator mode

When you simulate a model in accelerator mode, Simulink now uses Just-in-Time (JIT) acceleration to speed up the building of the accelerator target for the top-level model. Using JIT, Simulink generates an execution engine in memory instead of generating C code or MEX files during simulation. JIT provides the best performance for the generation of the accelerator target for a model.

For more information, see Accelerator Mode, or watch this video to learn more.

If you want to simulate your model using the classic, C-code generating, accelerator mode, run the following command:

```
set_param(O, 'GlobalUseClassicAccelMode', 'on');
```

Dataset Signal Plot: View and analyze dataset signals directly from the MATLAB command line

Use plot to plot dataset signals in the Signal Preview window for Simulink.SimulationData.Dataset and Simulink.SimulationData.DatasetRef objects.

Watch this video to learn more.

Multi-State Image Dashboard Block: Display different images based on the signal value

The Dashboard block library now includes a Multi-state Image block that you can use to display changes in signal values. After adding the block to your model, double-click the block and select signals whose states you want to capture visually. For each state, select a value and an image to display. When the value of a signal changes, the block displays the image that you set up for that particular value.

Simplified tasking mode setup

Previously, you set the tasking mode for your model using the **Tasking mode for periodic sample times** drop-down in the **Configuration Parameters** dialog box.

You were also able to set the tasking mode programmatically using the `SolverMode` parameter.

In **R2016b**, a simple check box labeled **Treat each discrete rate as a separate task** replaces the drop-down. The command-line equivalent of this parameter is `EnableMultiTasking`.

	Tasking mode for periodic sample times	Treat each discrete rate as a separate task
Command-line parameter	`SolverMode`	`EnableMultiTasking`
Parameter values	`Auto`, `Multitasking`	`On` (enables multitasking mode)
	`SingleTasking`	`Off` (enables single-tasking mode)

If you select the check box, single-tasking mode is used in these cases:

- Your model contains one sample time.
- Your model contains a continuous and a discrete sample time, and the fixed step size is equal to the discrete sample time.

For more information, see Treat each discrete rate as a separate task.

Compatibility Considerations

By default, new models now have the single-tasking mode enabled, whereas they had the `Auto` tasking mode enabled previously.

Further, if you have references to `SolverMode` in your scripts, replace those references with `EnableMultiTasking`. Use the permissible values for `EnableMultiTasking` to specify the tasking mode.

Diagnostic Suppressor: Suppress specific simulation warnings on particular blocks

The Diagnostic Viewer in Simulink now includes an option to suppress certain diagnostics. This feature enables you to suppress warnings for specific objects in your model. Click the **Suppress this warning** button next to the warning in the Diagnostic

Viewer to suppress the warning from the specified source. You can restore the warning from the source by clicking **Restore this warning**.
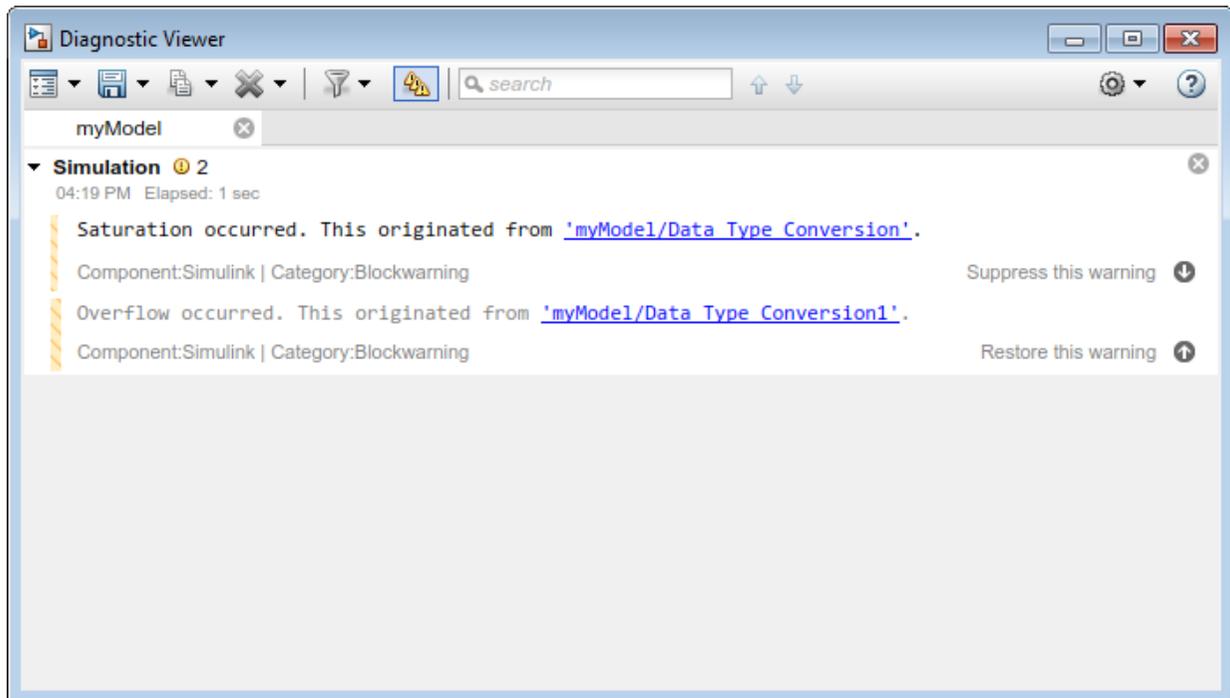
You can also control the suppressions from the command line. For more information, see Suppress Diagnostic Messages Programmatically.

Diagnostic Viewer: Improved build diagnostics display

Diagnostic Viewer detects compiler errors and warnings generated during model build and reports them with the ability to navigate to the source of the problem.

Export functions allow periodic function calls

Export functions have been enhanced to allow periodic function calls. In addition, you can now use the sample time of each export function call in your model to control the scheduling of triggers. The order of the function-call triggers is determined by the sample

times of their corresponding export-function blocks. Using the sample time provides more fine-grained control over how function-call triggers are scheduled in your model.

These enhancements provide the following advantages:

- Multiple exported functions can have the same execution period.
- Simulink now checks for sample time consistency in export functions.
- Export-function models support `auto` step size and dataset logging.
- Sample time optimization results in more efficient code.

For more information, see Export-Function Models.

Compatibility Considerations

As a result of this enhancement, the root outport logging behavior for export-function models in standalone simulations has changed. Previously, logging of root output ports for export-function models was inconsistent with the multitasking behavior of regular Simulink models, resulting in missing periodic rates in the logged output. Now, export-function models supports multitasking logging and dataset logging, which enable correct logging of all periodic rates from root output ports.

If you rely on the previous logging behavior, you can enable dataset logging to get the old output back. Note that dataset logging does not support asynchronous rate. In those cases, specify the base rate in the root input ports.

Simulink Editor

Property Inspector: Edit parameters and properties of model elements using a single interface

The Simulink Editor enables you to use a single interface to edit parameters and properties for any model element. When you display the Property Inspector, it becomes part of the model editing window. Select **View** > **Property Inspector** to open this interface. You can leave the Property Inspector open as you build your model, and it updates with your selection. To learn more about setting properties, see Setting Properties and Parameters, or watch this video to learn more.

You can use the Property Inspector to edit:

- Block parameters and properties
- Stateflow elements
- Annotations, areas, and images
- Signals
- Model properties

Edit-Time Checking: Detect and fix potential issues in your model at design time

The **Errors and Warnings** option enables edit-time checking, providing you with visual cues about issues with your model. To display information about an issue, hover the cursor over the highlighted block and click the error or warning icon.

Simulink detects the following block errors and warnings:

- Goto and From block mismatches.
- Duplicate data store blocks. The value of the Duplicate data store names parameter determines if there is a visual cue, and if the cue is an error or warning.

The **Errors and Warnings** option, is enabled by default. To turn off this option, in the model window, select **Display** > **Errors & Warnings**.

For more information, see Model Design Error Detection.

Finder: Search for model elements using improved interface

The interface for searching by way of **Edit** > **Find** (**Ctrl+F)** is improved in R2016b. To learn how to search using this interface, see Find Model Elements in Simulink Models.

Annotations in Libraries: Add annotations from libraries into models

In R2016b, you can add annotations to custom libraries for the user of the library to add to their model. To learn how to add an annotation to a library that appears in the Library Browser, see Create a Custom Library. Add annotations to your model from a library using the same techniques you use for adding blocks.

Library Browser: Expand or collapse libraries by default

You can customize the display of any library in the Library Browser tree to expand or collapse by default. See Customize Library Browser Appearance.

Library Browser API: Programmatically refresh the Library Browser

Using the `refresh` method with the `Library.LibraryBrowser2` class, you can programmatically refresh the library browser. To learn more, see LibraryBrowser.LibraryBrowser2.

Compatibility Considerations

In previous releases, these two commands returned `LibraryBrowser.LibraryBrowser2`:

```
lb = slLibraryBrowser
lb = LibraryBrowser.LibraryBrowser2
```

In R2016b, each returns `LibraryBrowser.LBStandalone`. However, the API to access Library Browser operations such as show and refresh has not otherwise changed.

Annotations: Click once to select annotation

In previous releases, a single click on an annotation prompted you to edit the annotation text instead of selecting the annotation frame. In R2016b, click one time to select the

annotation frame, which you can then drag or right-click to access the content menu. Click a selected annotation to edit the text.

Default Model Font: Specify default font for model elements

You can now set the default font for models. Setting the default font affects any existing model elements whose font you have not set manually as well as new model elements. In your model, with no selection, select **Diagram** > **Format** > **Font Styles for Models**. See Specify Fonts in Models. To use that font in new models, set the default font in the default template. See "Default Model Template: Use your own customized settings when creating new models" on page 2-15.

Simulink Preferences: Simplified and reorganized interface

Simulink Preferences now includes three panes:

- **General** pane for generated file folders, background colors for printing and exporting, and model display options
- **Editor** pane for customizing the Simulink Editor
- **Model File** pane for file saving options, including version notifications

Settings that are saved with the model no longer appear in Preferences. Use default templates for these settings instead of preferences. For more information, see "Default Model Template: Use your own customized settings when creating new models" on page 2-15.

For more information on the new Simulink Preferences panes, see Set Simulink Preferences.

Simulink Editor Fonts: FreeType font engine replaces Windows GDI font engine

On Windows® system, Simulink now uses the FreeType font engine. In previous releases, it used the Windows graphics device interface (GDI) font engine. This change improves the appearance of text in the user interface of the Simulink Editor.

If you are using a language that uses non-Latin characters, you might need to change the font in the editor to one that supports your language. To set defaults for a model, with no

selection in the Simulink Editor, select **Diagram** > **Format** > **Font Styles for Model**. Change the font using the dialog box.

Component-Based Modeling

Initialize and Terminate Function Blocks: Respond to events to model dynamic startup and shutdown behavior

Use an Initialize Function block with a State Writer block to set the state of a block in response to an initialize event. By default, the Initialize Function block includes a State Writer block and an Event Listener block with the **Event** parameter set to `Initialize`.

Use the Terminate Function block with a State Reader block to read the state of a block in response to a terminate event. By default, the Terminate Function block includes a State Reader block and an Event Listener block with the **Event** parameter set to `Terminate`.

Applications for these new blocks include:

- Starting up and shutting down an application component
- Initial condition calculations
- Save and restore state from nonvolatile memory

For more information, see Initialize Function, Terminate Function, Event Listener, State Reader, and State Writer block reference. Watch this video to learn more

Variant Subsystem Condition Propagation: Automatically assign variant conditions to blocks outside the subsystem for improved performance

When you specify variant conditions in models containing Variant Subsystem blocks, Simulink propagates these conditions outside to determine which components of the model are active during simulation.

A variant condition can be either a condition expression or a variant object.

For more information, see Condition Propagation with Variant Subsystem, or watch this video to learn more.

Simulink Units Updates

These are changes to the Simulink Units:

- Allowed Units list:

- The ASCII symbol of the reduced_Planck_constant unit has changed to h_bar. In R2016a, this symbol was hbar. Starting in R2016b, hbar refers to hectobar.

- Starting in R2016b, the dalton unit no longer allows the use of SI prefixes. In R2016a, the dalton unit incorrectly allowed the use of SI prefixes.

- Starting in R2016b, the deg unit is listed in both the English and SI (extended) units lists. In R2016a, the deg unit was listed in only the SI (extended) list.

- Starting in R2016b, the au unit refers to astronomical unit. In R2016a, au incorrectly referred to attodalton.

- Unit matching — As you type your unit in the **Unit** parameter, the software now suggests close, but not exact, unit matches highlighted with a star.

- Unit mismatching handling — Now when you click on the mismatched units warning badge, the dialog lets you go to either the source or destination where the unit is specified.

- Undefined units handling — The Model Advisor check now has a column with suggested unit alternatives that are a close match to the Simulink unit database.

Additional `SimStruct` Functions to Specify Units for Input and Output Ports

To specify units for S-function input and output ports, use these `SimStruct` functions:

- ssGetInputPortUnit
- ssGetOutputPortUnit
- ssRegisterUnitFromExpr
- ssSetInputPortUnit
- ssSetOutputPortUnit

Additional heterogeneous targets supported for concurrent execution

When configuring a model for concurrent execution, you can now build and download partitions of the model to these additional heterogeneous targets:

- Altera® Cyclone® SoC Rev. C development kit target
- Altera Cyclone SoC Rev. D development kit target
- Arrow® SoCKit development board target

For more information, see Specify a Target ArchitectureExport-Function Models.

Simulink.BusElement: SamplingMode property removed to support having blocks specify whether to treat inputs as frame-based signals

The `SamplingMode` property of `Simulink.BusElement` objects has been removed in R2016b. Specify the sampling mode (sample-based and frame-based) of input signals at the block level instead of at the signal level.

Compatibility Considerations

Scripts that use the `SamplingMode` property of `Simulink.BusElement` objects continue to work in R2016b. However, support for `SamplingMode` will be removed in a future release.

Export functions allow periodic function calls

Export functions have been enhanced to allow periodic function calls. In addition, you can now use the sample time of each export function call in your model to control the scheduling of triggers. The order of the function-call triggers is determined by the sample times of their corresponding export-function blocks. Using the sample time provides more fine-grained control over how function-call triggers are scheduled in your model.

These enhancements provide the following advantages:

- Multiple exported functions can have the same execution period.
- Simulink now checks for sample time consistency in export functions.
- Export-function models support `auto` step size and dataset logging.
- Sample time optimization results in more efficient code.

For more information, see Export-Function Models.

Compatibility Considerations

As a result of this enhancement, the root outport logging behavior for export-function models in standalone simulations has changed. Previously, logging of root output ports for export-function models was inconsistent with the multitasking behavior of regular Simulink models, resulting in missing periodic rates in the logged output. Now, export-

function models supports multitasking logging and dataset logging, which enable correct logging of all periodic rates from root output ports.

If you rely on the previous logging behavior, you can enable dataset logging to get the old output back. Note that dataset logging does not support asynchronous rate. In those cases, specify the base rate in the root input ports.

Variant Refresh: Improved performance with removal of live refresh

Simulink models offer better performance with the removal of live refresh of active models that contain Variant Subsystem blocks.

To update or refresh such models, click **Diagram** > **Refresh Blocks** (Ctrl + K) or **Simulation** > **Update Diagram** (Ctrl + D) in Simulink.

Variant Subsystem: Convert Subsystems with physical ports to Variant

You can convert a Subsystem block that contains physical ports to a Variant Subsystem block.

To do so, right-click the Subsystem block, in the context menu, click **Subsystem & Model Reference** > **Convert Subsystem to** > **Variant Subsystem**.

Variant Reducer: Additional model reduction modes in Variant Reducer (requires SLDV product license)

Use the Variant Reducer to reduce a variant model based on any one of these reduction modes:

- Variant control variable values in the global workspace.
- Variant control values specified as a comma-separated list.
- Variant configuration data object.

For more information, see Reduce Models Containing Variant Blocks.

Enhanced find_mdlrefs function: Keep models loaded that the function loads

The find_mdlrefs function loads models in the model reference hierarchy of the model that you specify. By default, the function closes those models, except for models

that were already loaded before execution of the function. You can now use the
`KeepModelsLoaded` name-value pair to keep all the models loaded that the function
loads.

Subsystem conversion to referenced models: Automatic subsystem wrapper and improved Goto and From block handling

The Model Reference Conversion Advisor and
`Simulink.SubSystem.convertToModelReference` function conversion creates a
wrapper subsystem automatically if the conversion modifies the Model block interface by
adding ports. The wrapper subsystem preserves the layout of the top model.

The conversion process no longer generates an error when Goto and From block pairs
cross model reference boundaries. The conversion automatically fixes the model,
regardless of how you set the **Fix errors automatically** (`'Autofix'`) option. The
conversion now produces a message highlighting the changes it made, so that you can
determine whether the changes produce the desired results.

For more information, see Convert a Subsystem to a Referenced Model and
Simulink.SubSystem.convertToModelReference.

Disallow multiple iterations of root Inport function-call with discrete sample time

For a function-call root Inport block with a discrete sample time vector, you cannot have
multiple times at the same sample time.

Project and File Management

Default Model Template: Use your own customized settings when creating new models

In R2016b, you can specify a model template to use for all new models. Create a model with the configuration settings and blocks you want, then export the model to a template. To reuse these settings in every new model, make the new template your default model template using the Simulink start page or the Simulink.defaultModelTemplate function.

In R2016b, after setting a default model template, every new model uses that template, for example, when you press **Ctrl+N**, when you use new model buttons, or when you use new_system. In the Simulink Editor, the default template name appears at the top of the list when you select **File** > **New** > *MyDefaultTemplateName*.

For details, see Use Customized Settings When Creating New Models, or watch this video to learn more.

Compatibility Considerations

Action	R2016a	R2016b
Create new models using: • **Ctrl+N** in the Editor or start page. • New model toolbar buttons in the Editor and Library Browser • new_system	Factory default blank model	You can specify your default template. Until you choose a new default template, the default template remains the blank model.
Specify configuration setting defaults	You could use Simulink Preferences to specify default parameters for new models.	The configuration defaults, font defaults, and display defaults are removed from the Simulink Preferences dialog box in R2016b. Use a model template instead. If you previously saved configuration defaults in the Simulink Preferences, in R2016b, Simulink converts these settings

Action	R2016a	R2016b
		to a model template, called `Converted Simulink Preferences` in the start page. If you previously specified configuration defaults and ran scripts on the root block diagram, then to use the old behavior, run the script `enablePreferencesCompatibilityMode`.
Specify model fonts	You could not change fonts in existing models.	In the Simulink Editor, with no model element selected, choose **Diagram** > **Format** > **Font Styles for Model**. To use the specified fonts as defaults for new models, export the model to a template.

Upgrade Advisor API: Automate the process of upgrading large model hierarchies

In R2016b, you can call `upgrade` or `analyze` on the output argument of the `upgradeadvisor` function. You can use the API to analyze only or also perform automatic fixes where available.

For details, see upgradeadvisor, or watch this video to learn more.

Project-Wide Search: Search inside all models and supporting files

Search across all your project files in one place. You can find matches inside model files, MATLAB files, and other project files such as PDF and Microsoft® Word files. On the **Simulink Project** tab, click **Search** and enter some characters to search for.

Open search results to locate specific items, e.g., highlight blocks in models, specific lines in MATLAB files. Filter results by file type, status, or label.

For details, see Search Inside Project Files and Filter File Views.

Refactoring Tools: Rename folders and automatically replace all references

In a Simulink project, when you rename a folder, the project checks for impact in other project files and offers fixes. You can find and fix impacts and avoid refactoring pain tracking down other affected files. Automatic renaming is helpful in preventing errors that result if you change names or paths manually and overlook or mistype one or more instances of the name.

For details, see Automatic Updates When Renaming, Deleting, or Removing Files.

Project Toolbox Analysis: Find products and toolboxes used by a project

In a Simulink project, in the Impact graph view, you can find the required toolboxes for the whole project or for selected files. You can see which products a new team member requires to use the project, or find which file is introducing a product dependency.

For details, see Find Required Toolboxes.

Project Derived File Analysis: Find out-of-date `.p`, `.slxp`. and `.mex` files in a project

In a Simulink project, you can run checks to ensure your derived files are up to date, and choose to rebuild the files.

After running dependency analysis, the Impact view shows:

- Relationships between source and derived files (such as `.m` and `.p` files, `slx`/`slxp`, `ssc`/`sscp`, `c`/`mex` files).
- Warnings when any derived file is out of date compared to its source file.

New project checks help you manage derived files:

- You can automatically rebuild out-of-date P-code files by running the project checks.
- If you rename a source file, the project detects impact in the derived file and prompts you to update it.

For details, see Check Dependency Results and Resolve Problems.

Project Export Profiles: Share specified files to zip archive

You can specify export profiles to control the project files you send to a zip archive. You can choose to exclude files, for example, when sharing a release version.

For details, see Archive Projects in Zip Files.

Project Batch Job Report: Archive results in a document

After running a batch job on project files, you can view and archive results in a batch job file. For details, see Run a Simulink Project Batch Job and Publish Report.

Git Submodules: Include submodules in your project

You can specify Git submodules to include in your project, to reuse code from another repository.

For details, see Add Git Submodules.

C/C++ file dependency analysis: View dependencies between C/C++ source and header files in the Impact graph

Use Simulink Project dependency analysis to explore dependencies between C/C++ source and header files in the impact graph.

For details, see Perform Impact Analysis.

Updated source control SDK: Write a source control integration providing file-based actions and annotations

You can use the source control Software Development Kit (SDK) to integrate Simulink Project with third-party source control tools. You can now use the updated SDK to write a source control integration that can provide file-based actions and annotations. An integration written using this API works with R2016b onwards. If you require a backwards-compatible integration, you can continue to use the old interfaces.

For details, see Write a Source Control Integration with the SDK.

Diff Tools: Customize external source control tools to use MATLAB to compare and merge

Use MATLAB Comparison tool to review changes to files such as MAT data and data dictionaries from your external source control client.

For details, see Customize External Source Control to Use MATLAB for Diff and Merge.

SVN Cleanup: Fix problems with working copy locks

Remove stale working copy locks using SVN Cleanup. See Get SVN File Locks.

Command-line Impact Analysis: Update and analyze the dependencies graph programmatically

In R2016b, you can access the dependency graph using the Simulink project API. You can perform command-line project impact analysis and get required files.

For details, see Get File Dependencies.

Data Management

Model Data Editor: Configure model data properties using a table within the Simulink Editor

The Model Data Editor allows you to inspect and edit data items (signals, data stores, and parameters) in a list that you can sort, group, and filter. You can then configure properties and parameters without having to locate the items in the block diagram.

While creating and debugging a model, you can configure multiple data items at once by selecting the corresponding signals and blocks in the block diagram. Work with the selected items in the Model Data Editor instead of opening individual dialog boxes.

Use the Model Data Editor to configure:

- Instrumentation for signals and data stores, which means you want to view and collect the simulation values. For example, you can configure signals and data stores to stream values to the Simulation Data Inspector.
- Design attributes such as data type, minimum and maximum value, and physical unit. You specify these attributes to control:
  - The values of numeric block parameters, such as the **Gain** parameter of a Gain block.
  - The interaction (interface) between components through Inport and Outport blocks and data stores.
- Storage classes for code generation. See Model Data Editor for applying storage classes to Inport blocks, Outport blocks, signals, and Data Store Memory blocks.

To use the Model Data Editor, in a model, select **View** > **Model Data**. See Configure Data Properties by Using a Table, or watch this video to learn more.

Output Logging: Log data incrementally, with support for rapid accelerator mode and variant conditions

For logging output (**Configuration Parameters** > **Data Import/Export** > **Output**) using `Dataset` format:

- If you enable **Configuration Parameters** > **Data Import/Export** > **Log Dataset data to file**, output data is logged to a MAT-file, using only a small constant amount

of memory. Before R2016b, when you accessed the output logged data, Simulink loaded the entire data at once. Now Simulink loads that data into the workspace incrementally. For information about logging to a MAT-file, see Log Data Using Persistent Storage.

- In addition to the simulation modes previously supported, you can now log to memory or to a MAT-file in rapid accelerator mode.
- For the active variant condition, Simulink creates a `Dataset` object with the logged data. For inactive variant conditions, Simulink creates MATLAB `timeseries` with zero samples.

Logging Inside For Each Subsystem: Log signals inside a For Each subsystem by marking lines with antennas

Performing signal logging in For Each subsystems no longer requires adding an Outport block outside of the subsystem for each signal that you want to log. You can mark for signal logging nonbus signals inside For Each subsystems. Directly logging signals in a For Each subsystem simplifies the model layout and editing. For details, see Log Signals in For Each Subsystems.

Logged Dataset Data Analysis: Call same function for all timeseries objects in logged Dataset data

Use the new Simulink.SimulationData.forEachTimeseries function to call a built-in or user-defined function on each MATLAB `timeseries` object that you specify as input. For example, you can use this function to resample every element of a structure of `timeseries` objects obtained by logging a bus signal. You can use `forEachTimeseries` with a function that returns a scalar. You can specify a `timeseries` object, an array of `timeseries`, a structure with `timeseries` at leaf nodes, or an array of structures with `timeseries` at leaf nodes.

Scalar expansion of initial value for data store

Before R2016b, if a data store (Data Store Memory block or `Simulink.Signal` object) represented an array (nonscalar) signal, you specified the initial value of the data store by using an array of the same dimensions. You could not use scalar expansion by specifying a scalar initial value. For several other blocks, scalar expansion enables you to specify the same initial value for each element of a nonscalar signal.

In R2016b, you can take advantage of scalar expansion for data stores with some exceptions. For more information, see Specify Initial Value for Data Store.

Technique to determine whether signal has variable size

You can use the programmatic parameter `CompiledPortDimensionsMode` of a block output port to determine whether the associated signal is a variable-size signal. For more information, see Programmatically Determine Whether Signal Line Has Variable Size.

View your model configuration parameters as a group on the All Parameters tab

Previously, in the Configuration Parameters dialog box, the **All Parameters** tab grouped configuration parameters only by the category that was displayed in the first column. In R2016b, this tab groups the parameters for each category into the same groups used on the **Commonly Used Parameters** tab. Headings that describe the category and group replace the **Category** column. The new **Advanced Parameters** group for each category contains parameters only available on the **All Parameters** tab.

For more information, see Configuration Parameters Dialog Box Overview.

Enhanced error reporting and extended syntax for specifying argument dimensions for function specifications in Legacy Code Tool

In R2016b, these features enhance the usability of the Legacy Code Tool:

- Improved error reporting — provides more specific information about argument specification errors.
- Extended syntax for specifying argument dimensions — In addition to the size function, you can specify function argument dimensions using expressions with:
  - The numel function
  - Arithmetic operators: +, −, *, /
  - Integer and floating-point literals

Class to package and share breakpoint and table data for lookup tables

You can use the new classes `Simulink.LookupTable` and `Simulink.Breakpoint` to store table and breakpoint data for lookup tables.

When you share data between lookup table blocks by using variables in a workspace or data dictionary, use this technique to improve model readability by:

- Clearly identifying the data as parts of a lookup table.
- Explicitly associating breakpoint data with table data.

For more information, see Package Shared Breakpoint and Table Data for Lookup Tables.

Simulink Coder™ and Embedded Coder® enable you to use this technique to package the data in the generated code for calibration according to the ASAP2 and AUTOSAR standards. See Storage of lookup tables for calibration according to ASAP2 and AUTOSAR standards.

These blocks have additional parameters to support the use of `Simulink.LookupTable` and `Simulink.Breakpoint` objects:

- n-D Lookup Table
- Prelookup
- Interpolation Using Prelookup

Root Inport Mapping Tool Updates

The Root Inport Mapping Tool has the following updates:

- Select a subset of scenarios for mapping and choose a mapping mode different from other scenarios (for more information, see Map Signal Data to Root Inports).
- If you import from a sheet whose name does not follow MATLAB variable name rules, the Root Inport Mapping tool now uses a modified sheet name. For more information, see Supported Microsoft Excel File Formats.

Option to disable resolution of signals and states to `Simulink.Signal` objects

To configure signal or state properties such as data type, minimum and maximum values, and storage class, you can create a `Simulink.Signal` object in a workspace or data dictionary. The signal property **Signal name must resolve to Simulink signal object**, the block parameter **State name must resolve to Simulink signal object**,

and the model configuration parameter **Signal resolution** control how the signal or state name corresponds to the object.

In R2016b, **Signal resolution** has an additional option, `None`. When selected, this setting prevents the signals, states, Stateflow data, and MATLAB Function block data in the model from acquiring settings from `Simulink.Signal` objects. Use this setting to reduce the dependency of the model on variables and objects in workspaces and data dictionaries, which can improve model portability, readability, and ease of maintenance.

However, selecting `None` does not affect data stores that you define by creating `Simulink.Signal` objects (instead of Data Store Memory blocks).

For more information about **Signal resolution**, see Signal resolution.

Help fixing configuration errors from Diagnostic Viewer

For some configuration error messages, the Diagnostic Viewer provides these methods for fixing the error:

- To fix simple configuration errors, click the **Fix** button. The software updates the parameter. Green or red text in the diagnostic viewer reports if the change is successful. To see the highlighted parameter in the Configuration Parameters dialog box before fixing, click the parameter link.

  Note: If you are modifying a referenced configuration, or if the parameter change results in a model reference incompatibility, a dialog box warning allows you to cancel the operation.

- To fix a configuration error with a more complex solution or multiple solutions, click the **Open** button next to your chosen solution. Then, update the highlighted parameter in the Configuration Parameters dialog box.

Metadata for Logging to Persistent Storage: Simulation metadata contains persistent storage logging settings to facilitate analysis of data from multiple simulations

R2016a introduced the option for you to log `Dataset` format simulation data to persistent storage in a MAT-file. To do so, select **Configuration Parameters** > **Data Import/Export** > **Log Dataset data to file**. In R2016b, Simulink stores metadata about logging to persistent storage. When you run multiple simulations using parallel

simulations or batch processing, specify a different persistent storage MAT-file for each simulation. Then you can use the metadata to provide context for analyzing the logged data stored in each file.

If you use single simulation output, Simulink creates a `Simulink.SimulationMetadata` object. In R2016b, this object includes in its `ModelInfo` structure a new `LoggingInfo` structure that contains two fields:

- `LoggingToFile` — Indicates whether logging to persistent storage is enabled (`'on'` or `'off'`)
- `LoggingFileName` — Specifies the resolved file name for the persistent storage MAT-file (if `LoggingToFile` is `'on'`)

The MAT-file used for persistent storage contains a new `SimulationMetadata` variable that stores the same simulation metadata as the `Simulink.SimulationMetadata` object.

For details, see Save Logged Data from Successive Simulations.

Improved display of large arrays by Model Explorer and `Simulink.Parameter` property dialog boxes

Before R2016b, for numeric MATLAB variables and `Simulink.Parameter` objects, the Model Explorer **Value** column displayed large arrays as read-only text, such as `<3x5x3 double>`. The property dialog box for `Simulink.Parameter` objects also used this read-only text.

In R2016b, the Model Explorer and property dialog boxes display the entire value of large arrays. To modify the elements in the array, you can edit the displayed text.

Arrays with three or more dimensions appear as an expression that contains a call to the `reshape` function. To edit the values in the array, modify the arguments of this `reshape` call.

For more information, see Edit and Manage Workspace Variables Used by Models.

Configuration set in base workspace resolves variables in base workspace

For a model that references a configuration set in the base workspace, if the configuration set uses a variable, Simulink resolves the variable in the base workspace.

Connection to Hardware

Raspberry Pi 3 Support: Run Simulink models on Raspberry Pi 3 hardware

The Simulink Support Package for Raspberry Pi Hardware now supports Raspberry Pi 3 hardware.

Watch this video to learn more.

Arduino: Improved External mode over serial communication

The external mode feature on the Simulink Support Package for Arduino Hardware is now improved with a faster serial communication protocol. The new protocol reduces data drop during data logging. With this change, increasing the baud rate also increases the data logging performance.

Simulink Support Package for Samsung GALAXY Android Devices renamed to Simulink Support Package for Android Devices

The Simulink Support Package for Samsung GALAXY® Android Devices has been renamed to the Simulink Support Package for Android Devices.

Google Nexus Support: Run Simulink models on Google Nexus Android devices

The Simulink Support Package for Android Devices now supports Google® Nexus Android devices.

Watch this video to learn more.

Block Enhancements

State Reader and Writer Blocks: Reset and record states during model execution

Use the State Writer block with an Initialize Function block to set the state of a block in response to an initialize event. Use the State Reader block with a Terminate Function block to read the state of a block in response to a terminate event.

Blocks with state include:

- Discrete State-Space
- Discrete-Time Integrator
- Discrete Transfer Fcn
- Discrete Filter
- Delay
- Unit Delay
- S-Function (with one data type work vector declared as a discrete-state vector)

For more information, see Initialize Function, Terminate Function, Event Listener, State Reader, and State Writer block reference.

MATLAB System Block Support for Global Data: Access Simulink data stores from System objects using global variables

The MATLAB System block now has ability to share data between multiple MATLAB System blocks. For more information, see Data Sharing with the MATLAB System Block.

MATLAB System block now supports enumerated data types

The MATLAB System block now supports enumerated data types.

MATLAB System Block Support for LAPACK: Generate faster standalone code for linear algebra in a MATLAB System block

For improved simulation speed of certain linear algebra function calls in a System object associated with a MATLAB System block, the simulation software can call LAPACK

functions. In R2016b, if you use Simulink Coder to generate C/C++ code from your model, you can specify that the code generator produce LAPACK function calls. If you specify that you want to generate LAPACK function calls, and the input arrays for the linear algebra functions meet certain criteria, the code generator produces calls to relevant LAPACK functions. The code generator uses the LAPACKE C interface.

LAPACK is a software library for numerical linear algebra. MATLAB uses this library in some linear algebra functions, such as `eig` and `svd`. Simulink uses the LAPACK library that is included with MATLAB. Simulink Coder uses the LAPACK library that you specify. If you do not specify a LAPACK library, the code generator produces code for the linear algebra function instead of generating a LAPACK call.

To generate LAPACK function calls and link to a specific LAPACK library, follow the procedure in Speed Up Linear Algebra in Code Generated from a MATLAB Function Block in the Simulink Coder documentation.

In R2016a, when you configured your model to generate LAPACK calls for standalone code, the configuration applied to MATLAB Function blocks and Stateflow charts. In R2016b, the configuration also applies to MATLAB System blocks. If you previously configured your model to generate LAPACK calls for MATLAB Function blocks or Stateflow charts, the code generator now also generates LAPACK calls for qualifying linear algebra calls in MATLAB System blocks.

Simpler way to call System objects

Instead of using the `step` method to perform the operation defined by a System object, you can call the object with arguments, as if it were a function. The `step` method will continue to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `dsp.FFT` System object named `fft1024`, then you call the System object as a function with that name.

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...
    'FFTLength',1024);
fft1024(x)
```

The equivalent operation using the `step` method is:

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...
    'FFTLength',1024);
```

```
step(fft1024,x)
```

When the `step` method has the System object as its only argument, the function equivalent has no arguments. This function but must be called with empty parentheses. For example, `step(sysobj)` and `sysobj()` perform equivalent operations.

System objects support for additional inputs, global variables, and enumeration data types

- System objects in code generated using MATLAB Coder can have up to 1024 inputs.
- You can use global variables declared in System objects to exchange data with the Data Store Memory block in Simulink. You can use these variables in generated code.
- Enumeration data types for System objects included in Simulink using the MATLAB System block is supported. Enumerations restrict data to a finite set of data values that inherit from `int8`, `uint8`, `int16`, `uint16`, `int32`, or `Simulink.IntEnumType` data types, or a data type you define using `Simulink.defineIntEnumType`.

Prelookup and Interpolation Using Prelookup Block Bus Support: Simplify and extend use of index and fraction signals

The Prelookup and the Interpolation Using Prelookup blocks use two signals as output and input, respectively:

- An index of the breakpoint set element
- An interval fraction that represents the normalized position on the breakpoint interval between the index and the next index value if the input is in range.

In R2016b, you can use a bus to combine index and fraction signals. Benefits of using a bus for these signals include:

- Simplifies the model by tying these two related signals together.
- Creates a testpoint `DpResult` structure for the AUTOSAR 4.0 library.
- For the AUTOSAR 4.0 library, avoids the creation of extra copies during code generation when the Prelookup and Interpolation Using Prelookup blocks are in separate models.

For more information, see the Prelookup and Interpolation Using Prelookup blocks.

From Spreadsheet block updates

The From Spreadsheet block now has a **Range** parameter. Use this parameter to select ranges of data in your spreadsheet.

Property inspector available for Simulink blocks

Property inspector view is now available for Simulink blocks. For more information on the property inspector, see Setting Properties and Parameters.

Manual Variant Source and Sink: Switch manually between different variants without using conditions

Use the Manual Variant Source and Sink blocks to switch between choices at input and output ports. You can double-click the block to toggle between the choices.

For more information, see Manual Variant Source and Manual Variant Sink.

Block Mask: Improved performance while evaluating mask parameter in fast restart mode

Once a model is initialized, only the tunable mask parameters are evaluated thus resulting in improved performance while evaluating the mask parameters.

Slider Range Parameter: Dynamically change the range of slider and dial parameter

You can change the range of the slider and dial parameters on a mask using callback code.

For more information, see the `Change Slider range Dynamically` model in slexMaskParameterOptionsExample.

Some types of unit delay blocks obsoleted

These unit delay blocks were removed from the Discrete library in R2016b. In new models, use the Delay block (with parameters set appropriately). For example, you can

use the Resettable Delay block to replace the Unit Delay Resettable External IC block. The Resettable Delay is the Delay block configured to reproduce the behavior of the Unit Delay Resettable External IC block.

Existing models that contain these blocks continue to work for backward compatibility.

- Unit Delay Enabled
- Unit Delay Enabled Resettable
- Unit Delay Enabled External IC
- Unit Delay Enabled Resettable External IC
- Unit Delay External IC
- Unit Delay Resettable
- Unit Delay Resettable External IC
- Unit Delay With Preview Enabled
- Unit Delay With Preview Enabled Resettable
- Unit Delay With Preview Enabled Resettable External RV
- Unit Delay With Preview Resettable
- Unit Delay With Preview Resettable External RV

Enhanced discrete block behavior

Previously, if your model used a variable-step solver, Simulink reported an error when it was unable to assign an appropriate sample time to the discrete blocks.

In this release, this discrepancy has been addressed so that discrete blocks always receive an appropriate sample time. The error message no longer appears.

In addition, for the following blocks, the default sample time setting has changed from 1 to −1:

- Discrete-Time Integrator
- Zero-Order Hold
- Discrete Zero-Pole
- Discrete PID Controller
- Discrete PID Controller (2DOF)
- Discrete State-Space

MATLAB Function Blocks

MATLAB Language Support: Use recursive functions and anonymous functions in a MATLAB Function block

Recursive functions

In R2016b, you can use recursive functions in MATLAB code that is intended for code generation. To generate code for recursive functions, MATLAB Coder uses *compile-time recursion* or *run-time recursion*. With compile-time recursion, the code generator creates multiple copies of the function in the generated code. The inputs to the copies have different sizes or constant values. With run-time recursion, the code generator produces recursive functions in the generated code. You can influence whether the code generator uses compile-time or run-time recursion by modifying your MATLAB code. You can disallow recursion or disable run-time recursion by modifying configuration parameters. See Code Generation for Recursive Functions.

The block-level function in a MATLAB Function block cannot be a recursive function, but it can call a recursive function.

Anonymous functions

In R2016b, you can use anonymous functions in a MATLAB Function block. For example, a MATLAB Function block can contain the following MATLAB code that defines an anonymous function that finds the square of a number:

```
sqr = @(x) x.^2;
a = sqr(5);
```

Anonymous functions are useful for creating a function handle to pass to a MATLAB function that evaluates an expression over a range of values. For example, this MATLAB code uses an anonymous function to create the input to the `fzero` function:

```
b = 2;
c = 3.5;
x = fzero(@(x) x^3 + b*x + c,0);
```

You cannot use anonymous functions for Simulink signals, parameters, or data store memory. For additional limitations, see Code Generation for Anonymous Functions.

Variable-Size Cell Array Support: Use `cell` to create a variable-size cell array in a MATLAB Function block

To create a variable-size cell array in a MATLAB Function block, you can use the cell function. For example:

```
function z = mycell(n, j)
assert(n < 100);
x = cell(1,n);
for i = 1:n
    x{i} = i;
end
z = x{j};
end
```

See Definition of Variable-Size Cell Array by Using cell.

Error for testing equality between enumeration and character array in a MATLAB Function block

For MATLAB code in a MATLAB Function block, an enumeration class must derive from a built-in numerical class. In R2016b, MATLAB introduces a new behavior for testing equality between these enumerations and a character array or cell array of character arrays. In previous releases, MATLAB compared the enumeration and character array character-wise. The behavior in a MATLAB Function block matched the MATLAB behavior. In R2016b, MATLAB compares the enumeration name with the character array. In R2016b, if a MATLAB Function block tests the equality of an enumeration and a character array, simulation or code generation ends with this error message:

```
Code generation does not support comparing an enumeration to a
character array or cell array with the operators '==' and '~='
```

Consider this enumeration class:

```
classdef myColors < int8
    enumeration
        RED(1),
        GREEN(2)
    end
end
```

The following code compares an enumeration with the character vector `'RED'`:

```
mode = myColors.RED;
z = (mode == 'RED');
```

In previous releases, the answer in MATLAB and generated code was:

```
0    0    0
```

In R2016b, the answer in MATLAB is:

```
1
```

In R2016b, simulation or code generation ends with an error.

Compatibility Considerations

If you want the behavior of previous releases, cast the character array to a built-in numeric class. For example, use the built-in class from which the enumeration derives.

```
mode = myColors.RED;
z = (mode == int8('RED'));
```

Incremental build for relocation of MATLAB program files on the MATLAB path

In previous releases, for MATLAB files called by a MATLAB Function block, incremental builds treated a relocated file on the MATLAB path as an updated file, even if the contents or timestamp did not change. The relocation of an unchanged file called by a MATLAB Function block caused a rebuild. In R2016b, incremental builds treat unchanged, relocated MATLAB files on the MATLAB path as unchanged files. Relocation of an unchanged MATLAB file called by a MATLAB Function block does not cause a rebuild.

To specify incremental builds, set **Simulation target build mode** to `Incremental build`.

Additional I/O Support: Generate code for fseek, ftell, fwrite

- fseek
- ftell
- fwrite

See Data and File Management in MATLAB in Functions and Objects Supported for C/C++ Code Generation — Category List.

Code generation for additional MATLAB functions

- cplxpair
- fminbnd
- inpolygon
- isenum
- polyeig
- repelem

See Functions and Objects Supported for C/C++ Code Generation — Alphabetical List.

Code generation for additional Audio System Toolbox functions

- integratedLoudness
- octaveFilter
- weightingFilter

See Audio System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

Code generation for additional Computer Vision System Toolbox functions

- cameraPoseToExtrinsics
- extrinsicsToCameraPose

See Computer Vision System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

Statistics and Machine Learning Toolbox Code Generation: Generate code for prediction by using SVM and logistic regression models

You can generate C code that classifies new observations by using trained, binary support vector machine (SVM) or logistic regression models, or multiclass SVM or logistic regression via error-correcting output codes (ECOC).

- saveCompactModel compacts and saves the trained model to disk.
- loadCompactModel loads the compact model in a prediction function that you declare. The prediction function can, for example, accept new observations and return labels and scores.
- predict classifies and estimates scores for the new observations in the prediction function.

  - To classify by using binary SVM models, see predict.
  - To classify by using binary logistic regression models, see predict.
  - To classify by using multiclass SVM or logistic regression via ECOC, see predict.

Communications and DSP Code Generation: Generate code for additional functions

Communications System Toolbox

- iqimbal
- comm.BasebandFileReader
- comm.BasebandFileWriter
- comm.EyeDiagram
- comm.PreambleDetector

See Communications System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

DSP System Toolbox

- dsp.MovingAverage
- dsp.MovingMaximum
- dsp.MovingMinimum
- dsp.MovingRMS
- dsp.MovingStandardDeviation
- dsp.MovingVariance
- dsp.MedianFilter
- dsp.BinaryFileReader
- dsp.BinaryFileWriter

- dsp.Channelizer
- dsp.ChannelSynthesizer

See DSP System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

Phased Array System Toolbox

- musicdoa
- pambgfun
- taylortaperc
- phased.GSCBeamformer
- phased.WidebandBackscatterRadarTarget
- phased.WidebandTwoRayChannel
- phased.MUSICEstimator
- phased.MUSICEstimator2D

See Phased Array System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

Conditional breakpoints for run-time debugging

To help you debug code, you can enter a MATLAB expression as a condition on a breakpoint inside a MATLAB Function block. Simulation then pauses on that breakpoint only when the condition is true. To set a conditional breakpoint, in the MATLAB Function block editor, right-click beside the line of code and select **Set Conditional Breakpoint**. Type the condition in the pop-up window. You can use any valid MATLAB expression as a condition. This condition expression can include numerical values and any data that is in scope at the breakpoint.

When you right-click a breakpoint, you can choose:

- `Set/Modify the condition`
- `Disable breakpoint`
- `Clear breakpoint`

You can also perform these actions from the **Breakpoints** menu in the MATLAB Function block editor.

Compiler optimization parameter support for faster simulation

In R2016b, Simulink applies the setting for the configuration parameter **Compiler optimization level** to MATLAB Function blocks. To speed up the simulation runtime for your model, set the configuration parameter to **Optimizations on (faster runs)**. The application of compiler optimizations might consume extra time during the build process. The default setting, **Optimizations off (faster build)** disables compiler optimizations and provides the fastest build times.

Run-Time error stack in Diagnostic Viewer

In the Diagnostic Viewer, some error messages display the run-time stack information and include the line numbers for the MATLAB code. You can trace these errors that occur in MATLAB Function blocks to the line of code. When you select the line number, the link opens the editor and highlights the corresponding line of code. Previously, the error message did not provide information about the location of the error.

Modeling Guidelines

Modeling guidelines for high-integrity systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0033: Usage of Lookup Table blocks
- hisl_0034: Usage of Signal Routing blocks
- hisl_0036: Configuration Parameters > Diagnostics > Saving
- hisl_0037: Configuration Parameters > Model Referencing
- hisl_0038: Configuration Parameters > Code Generation > Comments
- hisl_0039: Configuration Parameters > Code Generation > Interface
- hisl_0047: Configuration Parameters > Code Generation > Code Style
- hisl_0049: Configuration Parameters > Code Generation > Symbols

R2016a

Version: 8.7

New Features

Bug Fixes

Compatibility Considerations

Simulation Analysis and Performance

Automatic Solver Option: Set up and simulate your model more quickly with automatically selected solver settings

Starting with R2015b, you can use the auto solver to select a solver and step size for simulating a model. The auto solver suggests a fixed-step or variable-step solver along with maximum step size based on the dynamics of the model. Select the auto solver in the solver pane and accept recommended settings in the solver information dialog box. For more information, see Use Auto Solver to Select a Solver.

Starting with R2016a, auto solver calculates the stiffness of a model. For stiff models, auto solver selects `ode15s`. For more information, see Auto Solver Heuristics.

One-Click Display: Click a signal line when the simulation is running to view the current value

When you simulate your model, you can now display the port value label for a signal by clicking it. This option is selected by default.

For more information, see Display Value for a Specific Port or watch this video to learn more.

Simulation Metadata Diagnostics: Understand why a simulation has stopped in batch or individual runs

You can now use diagnostics in `SimulationMetadata` to understand why a simulation stopped. The metadata object has a new property called `ExecutionInfo`. This property contains information about stop events, error, and warning diagnostics. You can use the information in `ExecutionInfo` to troubleshoot individual runs or each run in a batch simulation.

See Simulink.SimulationMetadata to learn more.

Multi-Input Root Inport Mapping: Connect multiple sets of input signals to your Simulink model for interactive or batch simulation

The Root Inport Mapping tool now supports the connection of multiple sets of input signals to your Simulink model for interactive or batch simulation.

For more information, see Map Root Inport Signal Data or watch this video to learn more.

Simulation for Mixed Targets: Simulate system-level designs that integrate referenced models targeting an assembly of heterogeneous embedded devices

You can simulate a parent system model that includes referenced models configured for heterogeneous embedded devices. For example, you can simulate a parent model that has a referenced model **Configuration Parameters** > **Hardware Implementation** > **Device vendor** parameter set to ARM Compatible and another referenced model with that parameter set to ASIC/FPGA. Simulink models can now simulate system engineering level models that target multiple hardware devices.

Time Out feature for Performance Advisor run time

You can now specify a run-time duration for Performance Advisor by selecting the **Time Out** option. For models with long simulation times, use this option to limit the time period for Performance Advisor runs. For more information, see Specify Runtime for Performance Advisor.

Solver Profiler to speed up simulation performance

You can use Solver Profiler to examine solver behavior of variable step solvers and model behavior to identify issues that can contribute to poor simulation performance. Run Solver Profiler on models that fail to simulate or that have long simulation times. When you pause or stop the simulation, Solver Profiler displays information gathered during runtime including an analysis of conditions that can slow down simulation. Use this data to examine the model simulation for solver resets, zero crossings and solver exceptions.

Solver Profiler highlights states in the model that contain solver errors. You can also launch States Explorer from Profiler to further investigate the state plot of a model state.

For more information, see Examine Model Dynamics Using Solver Profiler.

Diagnostic Viewer performance improvement

The Diagnostic Viewer performs better and faster when handling large number of similar warnings from Simulink operations.

Component-Based Modeling

Variant Source and Sink Blocks with Condition Propagation: Design variant choices and automatically remove unneeded functionality based on block connectivity

Simulink provides two blocks to visualize implementations of variant choices in a model graphically—Variant Source and Variant Sink.

When you compile the model, Simulink determines which variant control evaluates to `true`. Simulink then deactivates blocks that are not tied to the variant control being true and visualizes the active connections.

When you specify variant conditions in models containing Variant Source and Variant Sink blocks, Simulink propagates these conditions backward and forward from these blocks to determine which components of the model are active during simulation.

See Variant Condition Propagation with Variant Sources and Sinks or watch this video to learn more.

Scoping Simulink Functions: Call Simulink Function blocks within a subsystem hierarchy

Defining the scope of Simulink functions in a Simulink model allows you to modularize your model by limiting the visibility and access of functions. Simulink uses subsystems to define the local scope of a function. See Define Scope of Simulink Function Blocks and Simulink Functions in Simulink Models, or watch this video to learn more.

Simulink Units: Specify, visualize, and check consistency of units on interfaces

Simulink now supports the specification of physical units as part of signal attributes. This capability enables you to specify unit attributes at the boundaries of components such as subsystems and models. To learn more, see Unit Specification in Simulink Models or watch this video.

For a list of supported unit systems and their units, see Allowed Unit Systems.

Units in Simulink.Parameter and Simulink.Signal Objects

The `DocUnits` property is now `Units`. Use this property to specify units for signals.

Note: You can still continue to use the `DocUnits` field to access or set the property. This capability maintains backward compatibility for existing MATLAB code, MAT-files, and Simulink data dictionaries that use the `DocUnits` field.

Specifying units in MATLAB Function blocks

Simulink supports the specification of a unit property for data inputs and outputs of MATLAB Function blocks. Specify units for input and output data by using the **Unit (e.g., m, m/s^2, N*m)** parameter on the Ports and Data Manager.

During model update, Simulink checks for inconsistencies in units between input and output data ports and the corresponding Simulink signals.

Units for logging and loading signal data

You can include units in signal data that you log or load.

Units for logging and loading are specified using `Simulink.SimulationData.Unit` objects. When you log using `Dataset` or `Timeseries` format, Simulink stores the unit information using `Simulink.SimulationData.Unit` objects. If you create MATLAB timeseries data to load, you can specify `Simulink.SimulationData.Unit` object for the `Units` property of the `timeseries` object.

For details, see Log Signal Data That Uses Units and Load Signal Data That Uses Units.

New units blocks

The following blocks are new:

- Unit Conversion
- Unit System Configuration

Configuration parameters

The following configuration parameters are new, available on the **All Parameters** tab:

- Allowed unit systems
- Unitsinconsistency messages
- Allow automatic unit conversions

Model Advisor checks

The following Model Advisor checks are new for units:

- Identify undefined units in the model
- Identify unit mismatches in the model
- Identify disallowed unit systems in the model
- Identify automatic unit conversions in the model

Updated example

The Exploring the Solver Jacobian Structure of a Model has been updated with unit specifications in all components.

Compatibility Considerations

- The `DocUnits` property is now `Unit` for `Simulink.Parameter` or `Simulink.Signal` objects. If, in a previous release, you used the `DocUnits` parameter of a `Simulink.Parameter` or `Simulink.Signal` object to contain text that does not now comply with units specifications, simulation returns a warning when the model simulates.

  To suppress these warnings, set the configuration parameter Unitsinconsistency messages to `none`. This setting suppresses all units inconsistency check warnings.

- If you have a class that derives from `Simulink.Parameter`, `Simulink.Signal`, or `Simulink.BusElement` with a previously defined `Unit` property, Simulink returns an error like the following:

  ```
  Cannot define property 'Unit' in class 'classname' because the property
  has already been defined in the superclass 'superclass'.
  ```

  If you use this property to represent the physical unit of the signal, delete the `Unit` property from the derived class in the R2016a or later release. Existing scripts continue to work, unless you are assigning incorrectly formed unit expressions to the

Unit field. In this case, replace the use of `Unit` with `DocUnits` to continue to be able to assign the unit expression.

Note: If you store existing data in a MAT- or `.sldd` file, in a release prior to R2016a, copy the contents of the `Unit` property to the `DocUnits` first. Then, save the file in the earlier release before loading the model in R2016a or later release.

Mask Dialogs: Create masks with flexible layout options and new control parameters

You can use these controls from the Mask Editor dialog box to create a rich mask dialog box:

- Collapsible panel: Expand or collapse sections in the mask dialog box.
- Dynamic mask dialogs: Change the menu options of a dependent dialog box based on the value of the controlling parameter.
- Spinbox step size: Fine-tune and control a parameter value in the mask dialog box.
- Horizontal stretch property: Create a more flexible mask dialog box layout.
- Slider and dial parameter: Use variables to specify value in dial and slider parameters.

For more information, see Mask Editor Overview.

Mask Images: Quickly add images to masks and while keeping the port names visible

- Mask icons can leave port names visible.
- Mask icon preview is available in the Mask Editor.

For more information, see Mask Editor Overview.

Tracing Simulink Functions: Display connections between all Function Callers and a Simulink Function

Visually display connections between Simulink functions and their callers with tracing lines that connect callers to functions:

- Turning on/off tracing lines — From the **Display** menu, select **Function Connectors**.

- Direction of tracing lines — Lines connected at the bottom of a block are from a function caller. Lines connected at the top of a block are to a Simulink function or a subsystem containing the function.

- Navigation to functions — If a function caller is within a subsystem, you can open the subsystem, and then click a link to the called Simulink function. If the function is at the root level of a model, the function opens. If the function is within a subsystem, the subsystem containing the function opens.

Signal Label Propagation for Referenced Models: Propagate signal labels out of referenced models by default

To propagate signal labels out of referenced models more easily, in R2016a for new models, the **Configuration Parameters** > **Data Model Referencing** > **Propagate all signal labels out of the model** parameter is now enabled by default. The default for the `PropagateSignalLabelsOutOfModel` parameter is now `'on'`. For details, see Propagate all signal labels out of the model.

Compatibility Considerations

If you open a model created before R2016a that has the **Propagate all signal labels out of the model** parameter cleared, that setting is preserved.

The new default can require updates to scripts written before R2016a that rely on the previous default `PropagateSignalLabelsOutOfModel` setting of `'off'`. To have the scripts work as expected with models created in R2016a or later, update the code to reflect that the default for that parameter is now `'on'`.

`Simulink.SubSystem.convertToModelReference` function for multiple subsystem conversion: Convert multiple subsystems with one command

You can convert multiple subsystems in a model to referenced models in one `Simulink.SubSystem.convertToModelReference` command.

Subsystem to Model Reference Conversion: Insert subsystem wrapper to preserve model layout

When you convert a subsystem to a referenced model, you can have the conversion process insert a wrapper subsystem to preserve the layout of a model. The subsystem wrapper contains the Model block from the conversion.

In the Model Reference Conversion Advisor, enable the **Create a wrapper subsystem** input parameter. In the `Simulink.SubSystem.convertToModelReference` function, use the `CreateWrapperSubsystem` name and value pair.

Model Reference Conversion Automatic Fix for Goto Blocks: Convert subsystems with Goto blocks more easily

The Model Reference Conversion Advisor and the `Simulink.SubSystem.convertToModelReference` function include additional checks and automatic fixes to make it easier to convert subsystems with Goto blocks to referenced models.

Virtual Bus Signals Across Model Reference Boundaries: Use virtual bus signals as inputs or outputs of a referenced model

In R2016a, when you specify that a bus signal input or output for a referenced model is a virtual bus, Simulink sets up the model so that, compared to previous releases, generated code generally:

- Has fewer copies of bus signals
- Executes faster

Use the Block Parameters dialog box for an Inport or Outport block to specify virtual bus output.

- Inport block — Clear the **Output as nonvirtual bus** parameter.
- Outport block — Clear the **Output as nonvirtual bus in parent model** parameter.

For information about the changes to the code that you generate from models, see Model Block Virtual Buses: Interface to Model blocks by using virtual buses, reducing data copies in the generated code.

Compatibility Considerations

The behavior of models that meet these criteria is different than it is in R2016a:

- The model was saved in a release earlier than R2016a.
- The model has referenced models with bus inputs and outputs configured to be treated as virtual buses.

Use the Upgrade Advisor Check for virtual bus across model reference boundaries check to avoid errors that the new behavior can trigger. Run the **Analyze model hierarchy and continue upgrade sequence** check on the top-level model and then down through the model reference hierarchy.

Bus Selector and Bus Assignment Block Signals: Display full signal path while editing a model

You can now display full paths to bus signals for Bus Selector and Bus Assignment blocks in the Simulink Editor, without opening the Block Parameters dialog boxes for the blocks. Interactively viewing the full signal path can simplify the process of editing signals with duplicate leaf names, by eliminating the need to match the block ports with the associated signals. Also, interactively viewing the full path for the signal provides a quick way to get context for understanding the model.

- To see the full path of a Bus Selector block output signal, hover over the output signal label.
- To see the full path of up to the first ten output signals of a Bus Selector block, hover over the block.
- To see the full path of up to the first ten signals in a Bus Assignment block, hover over the block.

Multi-Input Bus-Capable Block Ports: Simulate unconnected multi-input bus-capable block ports without error

Before R2016a, simulating a model that contains a multiport bus-capable block with an unconnected input port causes an error if one or more of the input ports is connected directly to *either*:

- A bus signal

- A Ground block

In R2016a, you can simulate under the same circumstances without error for these multiport bus-capable blocks:

- Manual Switch
- Multiport Switch
- S-Function
- Switch
- Variant Source
- Vector Concatenate

Using a Merge block in `Simplified` initialization mode in still causes an error for those situations.

Outport Blocks with Bus Output: Simulate Outport blocks with a bus output without error

Before R2016a, simulating a model that contains an Outport block whose output data type is specified by a bus object caused an error if *either* of these conditions applied:

- The Outport block has an unconnected port.
- The Outport block is connected directly to a Ground block.

In R2016a, you can simulate without error in these situations.

Function-Call Split block with multiple outputs

Before R2016a, the Function-Call Split block was a circular block with one input and two outputs. In R2016a, this block can split an incoming function-call signal into more than two output signals. You can also change the shape of the block.

Function-Call Split block with no input signal

Before R2016a, the Function-Call Split block had to have an input signal. In R2016a, you can simulate your model without supplying an input signal to the Function-Call Split block.

Trigger port with inherited periodic function-call signal

Before R2016a, if the Trigger block of a triggered subsystem had the **Trigger type** parameter set to `function-call` and **Sample time type** set to `periodic`, you could not set **Sample time** to `-1`. In R2016a, you can specify a sample time of `-1`.

Standalone code generation for models with asynchronous function-call inputs

Before R2016a, you had to build a top model from which to reference asynchronous function-call inputs. In R2016a, models with asynchronous function-call inputs support code generation on standalone models. You can now build these models as is.

Additional component parameters saved with `Simulink.ConfigSet.saveAs`

In R2016a, the `Simulink.ConfigSet.saveAs` function saves all enabled parameters in the base configuration set. In addition, the function saves enabled parameters for the following components:

- Coder Target
- HDL Coder
- Polyspace®
- SimEvents®
- Simscape™
- Simulink Coverage
- Simulink Design Verifier™
- Simulink PLC Coder™
- Target Hardware Resources

Project and File Management

Start Page: Get started or resume work faster by accessing templates, recent models, and featured examples

The Simulink start page helps you get started faster by offering model and project templates and examples. You can use common design patterns or learn about new features.

If you want to pick up where you left off, the start page shows a list of recent models, and opens the project automatically if your selected model is part of a project. You can now also open recent models from the Simulink Editor or from the Library Browser.

For details, see Create Models and Open Existing Models, or watch this video to learn more.

Compatibility Considerations

In previous releases, you could not access a list of recent models, and you could not find model and project templates and examples in the same place. The Simulink Template Gallery and the project template browser are merged into the start page.

Functionality	What Happens When You Use This Functionality?	Compatibility Considerations
simulink	Opens the start page	simulink no longer opens the Library Browser, but instead opens the start page. To open the Library Browser, use slLibraryBrowser instead, or click the Library Browser button in the Editor: .
Simulink	Opens the start page	The Simulink button in MATLAB no longer opens the Library Browser, but instead opens the start page.

Functionality	What Happens When You Use This Functionality?	Compatibility Considerations
Simulink button on the MATLAB Home tab.		
Creating a new model, chart, library, or project from the MATLAB Home tab **New** menu, from the Editor, from the Library Browser, or from Simulink Project.	Opens the start page	Choose a template from the start page. The start page shows an appropriate filtered list of models, charts, libraries, or projects. To recreate the previous workflow: • In the start page, select Blank Model, Blank Library, Blank Project, or Chart, (or press **Ctrl+N** for a blank model), • In the Editor or Library Browser, select a new **Blank Model** (or press **Ctrl+N**). To create a project using source control or from an archive, use the start page options.

Functionality	What Happens When You Use This Functionality?	Compatibility Considerations
Use model templates.	Find all your templates on the start page.	The menu item **From Template** is no longer in the Editor or Library Browser. Instead, in the Simulink Editor, select **File** > **New** > **Model** and select your template in the start page. In the Library Browser, click the **New Model** button arrow and select **Model**. The Simulink Template Gallery is now merged into the start page.
Use project templates.	Find all your templates on the start page.	Project templates are no longer in the MATLAB Home tab **New** menu or in the Create Project dialog box. Instead, open the start page from MATLAB, or from the Editor by selecting **File** > **New** > **Project**, and select your template in the start page.
Simulink project templates created in R2014a or earlier (`.zip` files)	You cannot browse to legacy templates in the start page	Upgrade legacy templates to `.sltx` files using Simulink.exportToTemplate.

Automatic Renaming: Update all references in a project when you rename models, libraries, or MATLAB files

In a Simulink project, when you rename, delete, or remove a file, the project checks for impact in other project files. You can find and fix impacts such as changed library links, model references, and model callbacks. This tooling can help you avoid refactoring pain tracking down other affected files. Automatic renaming is helpful in preventing errors that result if you change names or paths manually and overlook or mistype one or more instances of the name.

Automatic renaming helps you refactor MATLAB code. Simulink project dependency analysis now finds dependencies on MATLAB code in packaged functions, classes, and superclasses. You can view the dependencies in the Impact graph and if you refactor the files, automatic renaming prompts you. For example, when renaming a class, the project offers to automatically update all classes that inherit from it. If you rename a `.m` or `.mlx` file, the project offers to automatically update any files and callbacks that call it.

For details, see Automatic Updates When Renaming, Deleting, or Removing Files.

Three-Way Model Merge: Resolve conflicts between revisions and ancestor models using Simulink projects

In a Simulink project under source control, if your changes in a model conflict with another user, you can open the Three-Way Model Merge tool to resolve the conflicts. You can examine your local file compared to the conflicting revision and the base ancestor file, and decide which changes to keep. You can resolve the conflict and submit your changes. Three-Way Model Merge requires Simulink Report Generator™.

For details, see Resolve Conflicts.

Template API: Programmatically create models and projects from custom templates

In R2016a, you can programmatically create models and projects from templates, and create custom templates. Model and project templates are starting points to apply common modeling approaches. They help you reuse settings and block configurations and share knowledge. Use model and project templates to apply best practices and take advantage of previous modeling solutions.

To use templates programmatically, see Simulink.createFromTemplate, Simulink.findTemplates, and Simulink.exportToTemplate.

Export function: Export to previous version using `Simulink.exportToVersion`

In R2016a you can use a new function, Simulink.exportToVersion, to export a model so that you can open it in a previous version of Simulink. The new function makes it easier to distinguish between exporting to a previous version and saving with a different name.

In previous releases you could use `save_system` with the `'ExportToVersion'` option. This option will also continue to work.

Dirty Model Management: Identify, save, or discard unsaved changes in project models

In a Simulink project, you can check for unsaved changes in project models, and decide whether to save or discard changes. In the previous release, you had to close the project to see warnings about unsaved changes. Now you can manage unsaved changes before closing the project.

For details, see Manage Shadowed and Dirty Model Files.

Source Control API: Programmatically get modified files and revision information

In R2016a, you can programmatically get source control status information about Simulink project files. Use `simulinkproject` to get a project object, then you can get a list of modified files in the project. You can get the local status of a file (modified, unmodified, not under source control) and the revision if available.

For details, see listModifiedFiles.

Source Control Notifications: List changed files on update (SVN); find out if your branch is behind the origin (Git)

In R2016a, Simulink Project provides more notifications to help you with source control operations. With SVN, when you update a project from the repository, you see a list of all changed files in a dialog box. With Git, when you fetch from a remote repository, you can see if your current branch is behind or ahead of the remote tracking branch (the origin).

For details, see Update Revisions with SVN and Fetch and Merge.

SVN Externals: Include files in projects from other repositories or repository locations

Use SVN externals to get files into your Simulink project from another repository or from a different part of the same repository. Right-click a project folder and select **Manage**

SVN Externals. The project provides a dialog to help you browse, specify and validate the externals definitions. After you define the externals setting on a folder, other project users can get the same files included in their project.

For details, see Manage SVN Externals.

Custom Shortcut Icons: Personalize frequent task buttons on the toolstrip

In R2016a, you can specify an icon to use for your shortcut buttons on the Project Shortcuts tab. Icons such as "build" can aid other project users to recognize frequent tasks. For details, see Customize Shortcut Icons.

Simplified Configuration Parameters: Configure model more easily using streamlined category panes

In the Configuration Parameters dialog box, streamlined category panes display only configuration parameters that you are most likely to use when configuring your model.

The category panes, previously referred to as the Category view, are now available on the **Commonly Used Parameters** tab. The **All Parameters** tab, previously referred to as the List view, provides the complete list of parameters in the model configuration set.
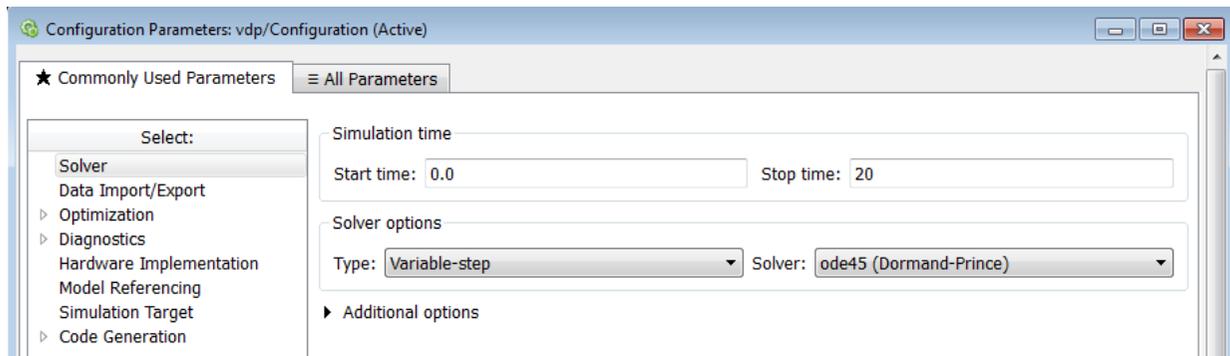
Compatibility Considerations

Following are the configuration parameters that have moved to the **All Parameters** tab or moved to a different pane.

Note: Parameters that are removed from a pane are still available for configuration on the **All Parameters** tab. To locate a parameter on this tab, use either the search box or the **Category** filter.

Data Import/Export Pane

The **Enable live streaming of selected signal to Simulation Data Inspector** parameter is moved to the **All Parameters** tab.

The **Save simulation output as single object** parameter is now called **Single simulation output**

The following parameters are available by clicking **Additional Parameters** at the bottom of the pane:

- **Limit data points to last**
- **Decimation**
- **Output options**
- **Refine factor**

Diagnostics Pane

The following parameter is moved to the **All Parameters** tab:

- **Solver data inconsistency**

Diagnostics > Data Validity Pane

The following parameters are moved to the **All Parameters** tab:

- **Array bounds exceeded**
- **Model verification block enabling**
- **Check preactivation output of execution context**
- **Check runtime output of execution context**
- **Check undefined subsystem initial output**
- **Detect multiple driving blocks executing at the same time step**
- **Underspecified initialization detection**

Diagnostics > Saving Pane

The pane is removed and its parameters are moved to the **All Parameters** tab:

- **Block diagram contains disabled library links**
- **Block diagram contains parameterized library links**

Diagnostics > Solver Pane

The following parameters are moved to the **Diagnostics > Sample Time** pane:

- **Sample hit time adjusting**
- **Unspecified inheritability of sample time**

The following parameter is moved to the **Diagnostics > Compatibility** pane:

- **SimState object from earlier release**

Optimization Pane

The following parameters are moved to the **All Parameters** tab:

- **Remove code from floating-point to integer conversions with saturation that maps NaN to zero**
- **Compiler optimization level**
- **Verbose accelerator builds**
- **Implement logic signals as Boolean data (vs. double)**
- **Block reduction**
- **Conditional input branch execution**
- **Use memset to initialize floats and doubles to 0.0**

Optimization > Signals and Parameters Pane

The following parameters are moved to the **All Parameters** tab:

- **Signal storage reuse**
- **Enable local block outputs**
- **Reuse local block outputs**
- **Optimize global data access**

- **Reuse global block outputs**
- **Eliminate superfluous local variables (Expression folding)**
- **Simplify array indexing**

Simulation Target Pane

The following parameters are moved to the **All Parameters** tab:

- **Echo expressions without semicolons**
- **Simulation target build mode**
- **Ensure responsiveness**
- **Generate typedefs for imported bus and enumeration types**
- **Ensure memory integrity**

Simulation Target > Custom Code Pane

The pane is removed and its parameters are moved to the **Simulation Target** pane:

- **Header file**
- **Initialize function**
- **Source file**
- **Terminate function**
- **Parse custom code symbols**
- **Include directories**
- **Libraries**
- **Source files**
- **Defines**

Simulation Target > Symbols Pane

The pane is removed and its parameter is moved to the **Simulation Target** pane:

- **Reserved names**

Simulink Editor

Single-Selection Actions: Access commonly used editing actions when clicking a block or signal line

When you select a block or a signal line in a Simulink model, a cue appears that lets you select a common action to perform. When you move your cursor over the cue, an action bar appears. Click the action you want to perform.

- For blocks, you can comment or uncomment the block or hide or display the block name using this cue.
- For signal lines, you can autoroute the line or enable or disable signal logging.

The figure shows how to use the cue to comment out a block.
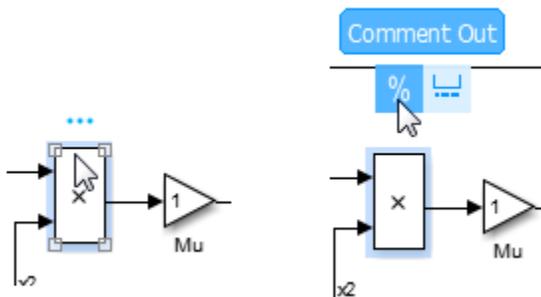
Watch this video to learn more.

Multiple-Selection Cue: Selecting multiple blocks in the Simulink Editor shows new cue

When you select multiple blocks in your model, the Simulink Editor now displays a smaller cue, as shown in the figure. Move your mouse over the cue to display the action bar of common commands.
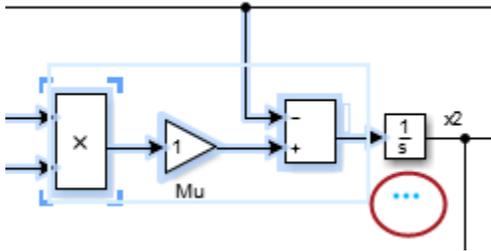
Single Click for Quick Insert: Click block name once to insert block from list

If you know the name of the block you want to add to a model, you can click in the model where you want to add the block and start typing the block name. The blocks that match appear in a list. In previous releases, you double-clicked the block name to insert it in the model. In R2016a, you click the block name once to insert it. For an example, see Add More Blocks.

Interactive Library Unlocking: Click lock symbol in custom libraries to unlock

Clicking the lock symbol in a custom library now unlocks the library so you can edit it. For information about library locks, see Create a Custom Library.

Key Parameter Preference: Turn off parameter prompt during block creationWhen you add a block to a model, a prompt appears so that you can enter a key parameter. To prevent this prompt from appearing, you can set a preference. In your model, select FileSimulink Preferences. In the Editor Preferences pane, clear the Edit key parameter when adding new blocks check box.

Improved block search usability

The Quick Block Insert search displays a description of the library making it easier to recognize the block in the Library Browser.

Data Management

Signal and State Logging to File: Log data directly to a MAT-file for long simulations

When logging large amounts of data that can cause memory issues for a long simulation, use the new **Configuration Parameters** > **Data Import/Export** > **Log Dataset data to file** parameter. With this feature enabled, Simulink stores in a MAT-file data that is in `Dataset` format for these kinds of logging:

- Signal logging
- States
- Final states
- Output
- Data stores

Alternatively, you can enable this feature using the `LoggingToFile` and `LoggingFileName` name-value pairs with either the `sim` command or `set_param` command.

Using a `Simulink.SimlationData.DatasetRef` object to access signal logging and states logging data loads data into the model workspace incrementally. Loading and accessing data for other kinds of logging loads all of the data into memory at once.

For details, see Log Data Using Persistent Storage.

Preserve symbolic constants in propagated signal dimensions

Previously, Simulink treated signal and parameter dimension specifications as numeric constants. In R2016a, you can use a `Simulink.Parameter` object as a symbol in a MATLAB expression to represent a dimension value. During simulation, Simulink propagates dimension symbols throughout the model and preserves these symbols in the propagated signal dimensions.

For example, Inport blocks `In1` and `In2` have symbolic constant dimension specifications. In the **Signals Attribute** tab in the Source Block Parameters dialog box for `In1`, the Port dimension parameter has the `Simulink.Parameter` A, which has a value of `2`. In the **Signals Attribute** tab in the Source Block Parameters dialog box for `In2`, the

Port dimension parameter has the `Simulink.Parameter` B, which has a value of 3. When you simulate the model, you see the symbolic constants and their values propagate throughout the model.
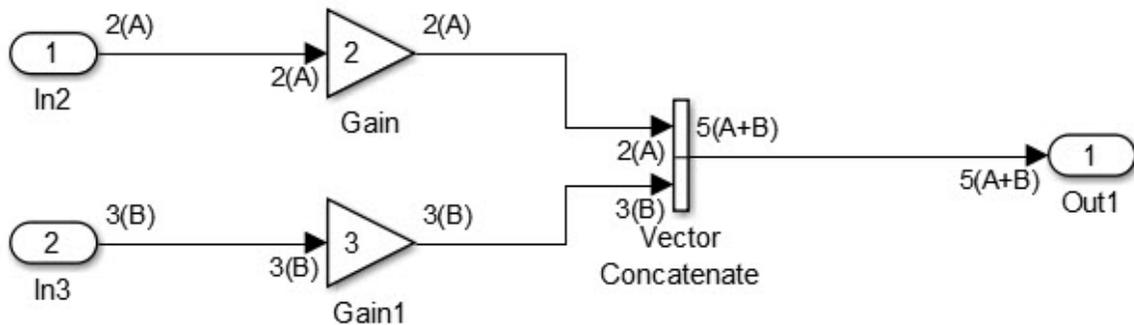
Embedded Coder preserves these dimension symbols and expressions in the generated code. If you use Embedded Coder, you can compile the same generated code into multiple applications of different sizes. For more information on generating code with symbolic constants, see Compile-Time Dimensions: Generate compiler directives (#define) for implementing signal dimensions and Implement Dimension Variants for Array Sizes in Generated Code.

Dataset Format for Signal Logging: Log signals in format used for other logging

When you open a model in R2016a, signal logging uses the `Dataset` format for the logged data. The `SignalLoggingSaveFormat` parameter is set to `Dataset`.

`Dataset` is the default format for logging output and state information. Using the same format for signal logging data as well as other logged data simplifies writing scripts to process that data.

Compatibility Considerations

The `ModelDataLogs` class is supported for backwards compatibility. Starting in R2016a, you cannot log data in the `ModelDataLogs` format. In R2016a or later, when you open a model from an earlier release that had used `ModelDataLogs` format, the model is converted to use `Dataset` format.

You can convert signal logging data from `ModelDataLogs` to `Dataset` format. For more information, see Convert Logged Data to Dataset Format.

If you have legacy code that uses the `Simulink.ModelDataLogs` API, you can encounter situations that require updates to your code or model. See Migrate Scripts That Use ModelDataLogs API.

Unlimited Number of Data Points for Logging by Default: Log all data points by default

To simplify logging time, output, and state data, the default behavior is to have no limit on the number of data points logged during a simulation. In R2016a, by default the **Configuration Parameters** > **All Parameters** > **Limit data points** is now cleared.

Compatibility Considerations

The changed default can require updates to scripts written before R2016a that rely on the previous default of limiting the logged data points to 1000 or to a specified number of data points and that use:

- The `new_system` function to create an empty Simulink system
- `Simulink.ConfigSet` to access the model configuration set

Root Inport Mapping Tool Updates

The Root Inport Mapping Tool has the following updates:

- New interface
- Support for Excel® spreadsheets on all platforms
- Support for the linking in of multiple scenario files at the same time

For more information, see Map Root Inport Signal Data

Function to convert MAT-file contents to `Simulink.SimulationData.Dataset` object

New convertToSlDataset function converts the contents of a MAT-file that contains Simulink inputs to a `Simulink.SimulationData.Dataset` object.

Functions to identify and close data dictionaries

Some commands and functions, such as
`Simulink.data.dictionary.cleanupWorkerCache`, cannot
operate when data dictionaries are open. Use the new function
Simulink.data.dictionary.getOpenDictionaryPaths to identify open data dictionaries.

You can use the new function Simulink.data.dictionary.closeAll to close all connections
to all open data dictionaries. However, it is a best practice to close each individual
connection.

Navigation to variables from additional block dialog boxes

Beginning in R2015a, you can navigate to or create variables directly from blocks in
the base Simulink libraries such as Discrete and Continuous. You can navigate to or
create the variables in a workspace or data dictionary, or navigate to mask parameters,
by right-clicking the variable name in the block dialog box. For more information, see
Manage Variables from Block Parameter.

In R2016a, the blocks in these additional libraries allow you to navigate to variables from
block dialog boxes:

- **Communications System Toolbox**
- **Communications System Toolbox Support Package for RTL-SDR Radio**,
  except for the **Radio address** parameter in the RTL-SDR Receiver block
- **Communications System Toolbox Support Package for Xilinx FPGA-Based
  Radio**
- **SimEvents**, except for the **Port number** parameter in the Conn legacy block
- **Simulink Coder**
- **Simulink Control Design**
- **Simulink Design Optimization**
- **Simulink Design Verifier**
- **Simulink Extras**
- **Simulink Support Package for Apple iOS Devices**
- **Simulink Support Package for BealgeBoard Hardware**
- **Simulink Support Package for LEGO MINDSTORMS EV3 Hardware**
- **Simulink Support Package for LEGO MINDSTORMS NXT Hardware**

- Simulink Support Package for Raspberry Pi Hardware
- Simulink Support Package for Samsung GALAXY Android Devices
- System Identification Toolbox

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations

The programmatic parameter `DefaultDataPackage`, which corresponds to the **Package** option on the **Data Management Defaults** pane in the Simulink Preferences dialog box.	Warning. For example, this command generates a warning:
	
	`set_param(0,...`
	`'DefaultDataPackage',...`
	`'mpt')`
		To select default classes for creating signal and parameter objects, on the Model Explorer toolbar, click the arrows next to the Simulink Signal and Simulink Parameter buttons.
		
		To select a default package when you apply a custom storage class in a Signal Properties dialog box or in a block dialog box, use the new drop-down list **Signal object class**.
		
		In the Data Object Wizard, click **Change Class** to select a class for each data object that the wizard proposes.
			In your scripts, remove references to the programmatic parameter `DefaultDataPackage`.

Connection to Hardware

Hardware implementation parameters enabled by default

In R2016a, the **Enable hardware specification** button is removed from the **Configuration Parameters** > **Hardware Implementation** pane. By default, the parameters on the pane are enabled.

Mac Support for LEGO EV3: Run Simulink models on LEGO EV3 hardware from a Mac

You can use the Simulink Support Package for LEGO MINDSTORMS EV3 Hardware on the Apple Mac OS X platform.

Block Enhancements

From Spreadsheet Block Updates

The From Spreadsheet has the following changes:

- The block has moved from the Simulink Extras/Additional Sources sublibrary to the Simulink Sources sublibrary.
- The block now supports Rapid Accelerator mode on all platforms.

System object enhancements to MATLAB System block

To implement these classes and methods for defining your own System objects, add them to your object's class definition file.

- Fixed-point data tab — The showFiSettingsImpl method adds a Data Types tab to the MATLAB System block dialog box. This tab includes options for fixed-point data settings.
- Model reference discrete sample time inheritance — The allowModelReferenceDiscreteSampleTimeInheritanceImpl method lets you specify whether a System object in a referenced model can inherit the sample time of the parent model. If your object uses discrete sample time in its algorithm, you set this method to `true` to allow inheritance.

Unit Delay block does not accept rate transitions

In R2016a, if the Unit Delay block input and output signals are at different rates, the block produces an error. For the rate transitions workflow, use a Rate Transition block.

Matrix Interpolation Block for Multidimensional Lookup Table Data

The Matrix Interpolation block performs interpolation (or extrapolation) on a multidimensional table, where each data point can be a matrix. The block resides in the Simulink Extras library.

Enhanced System Object Development with MATLAB Editor

Create System objects in the MATLAB Editor using code insertion and visualization options.

- Define your System object with options to insert properties, methods, states, inputs, and outputs.
- View and navigate the System object code with the Analyzer.
- Develop System block and preview block dialog box interactively (with Simulink only).
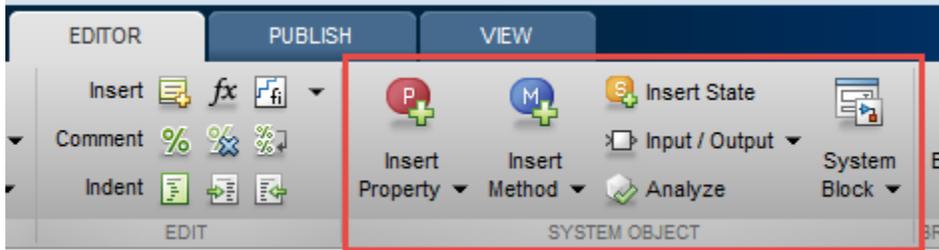
These coding tools are available when you open an existing System object or create a new System object with **New** > **System object**.

Scope Block and Signal Viewer Enhancements

- Added ability to select signals for individual displays. This change removes an incapability from previous releases. See Connect Signals to Floating Scope Block or Scope Viewer
- Expand scope window layout from 4 x 4 to 16 x 16 displays. See Select Number of Displays and Layout.

MATLAB Function Blocks

Cell Array Support: Use additional cell array features in a MATLAB Function block

In R2016a, support for cell arrays in a MATLAB Function block includes:

Use of {end + 1} to grow a cell array

You can write code such as X{end + 1} to grow a cell array X. For example:

```
X = {1 2};
X(end + 1} = 'a';
```

When you use {end + 1} to grow a cell array, follow the restrictions described in Growing a Cell Array by Using {end + 1}.

Value and handle objects in cell arrays

Cell arrays can contain value and handle objects. You can use a cell array of objects as a workaround for the limitation that code generation does not support objects in matrices or structures.

Function handles in cell arrays

Cell arrays can contain function handles.

Non-Power-of-Two FFT Support: Generate code for fast Fourier transforms for non-power-of-two transform lengths

In previous releases, code generation required a power of two transform length for fft, fft2, fftn, ifft, ifft2, and ifftn. In R2016a, code generation allows a non-power-of-two length for these functions.

Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific LAPACK library

To improve the simulation speed of MATLAB Function block algorithms that call certain linear algebra functions, the simulation software can call LAPACK functions. In R2016a,

if you use Simulink Coder to generate C/C++ code for these algorithms, you can specify that the code generator produce LAPACK function calls. If you specify that you want to generate LAPACK function calls, and the input arrays for the linear algebra functions meet certain criteria, the code generator produces calls to relevant LAPACK functions. The code generator uses the LAPACKE C interface.

LAPACK is a software library for numerical linear algebra. MATLAB uses this library in some linear algebra functions, such as `eig` and `svd`. Simulink uses the LAPACK library that is included with MATLAB. Simulink Coder uses the LAPACK library that you specify. If you do not specify a LAPACK library, the code generator produces code for the linear algebra function instead of generating a LAPACK call.

See LAPACK Calls for Linear Algebra in a MATLAB Function Block.

To specify that you want to generate LAPACK function calls and link to a specific LAPACK library, see Speed Up Linear Algebra in Code Generated from a MATLAB Function Block in the Simulink Coder documentation.

Concatenation of variable-size, empty arrays

In R2016a, the MATLAB Function block treatment of an empty array in a concatenation more closely matches the MATLAB treatment.

For concatenation of arrays, MATLAB and the MATLAB Function block require that corresponding dimensions across component arrays have the same size, except for the dimension that grows. For horizontal concatenation, the second dimension grows. For vertical concatenation, the first dimension grows.

In MATLAB, when a component array is empty, the sizes of the nongrowing dimensions do not matter because MATLAB ignores empty arrays in a concatenation. In previous releases, the MATLAB Function block required that the sizes of nongrowing dimensions of an variable-size, empty array matched the sizes of the corresponding dimensions in the other component arrays. A dimension size mismatch resulted in a simulation error.

In R2016a, for most cases of empty arrays in concatenation, the MATLAB Function block behavior matches MATLAB behavior. In some cases, if the MATLAB Function block does not recognize the empty array and treats it as a variable-size array, a dimension size mismatch results in a simulation error.

Consider the function `myconcat` that concatenates two arrays A and B.

```
function D = myconcat(n, B)
%#codegen
assert(n <= 5);
A = zeros(n,3);
C = [A, B];
D = numel(C);
end
```

The size of A is :5-by-3. The first dimension has a variable size with an upper bound of 5. The second dimension has fixed size 3. Suppose that B is a 5-by-5 array. If n is 0, A is an empty array whose size is 0-by-3. In previous releases, myconcat in a MATLAB Function block resulted in a size mismatch error because the size of dimension 1 of A did not match the size of dimension 1 of B. In R2016a, in a MATLAB Function block, the output from myconcat is 25. This output is the same as the output from myconcat in MATLAB.

Compatibility Considerations

When the result of the concatenation is assigned to a variable that must be a fixed-size variable, support for a variable-size, empty array in a concatenation introduces an incompatibility.

In previous releases, it was possible that a concatenation that included a variable-size array produced a fixed-size array because concatenation rules were stricter in the MATLAB Function block than in MATLAB. In R2016a, a concatenation that includes a variable-size array produces a variable-size array. If the result of the concatenation is assigned to a variable that must be a fixed-size variable, an error occurs.

Consider the function myconcat1 that concatenates two arrays X and Y.

```
function c = myconcat1(n, Y)
%#codegen
assert(n <= 2);
X = zeros(n,2);
Z.f = [X Y];
c = numel(Z.f);
```

X has size :2-by-2. The first dimension has a variable size with an upper bound of 2. The second dimension has fixed size 2. Suppose that Y is a 2-by-4 array. In previous releases, the MATLAB Function block determined that the result of [X Y] had fixed size 2-by-6. In R2016a, the result of [X Y] has a size of 2-by-:6. The first dimension has a fixed size of 2 and the second dimension has a variable size with an upper bound of 6. This size

accommodates an empty and nonempty X. If X is empty, the size of the result is 2-by-4. If X is nonempty, the size of the result is 2-by-6.

Consider the function `myconcat2`.

```
function c = myconcat2(n, Y)
%#codegen
assert(n <= 2);
X = zeros(n,2);
Z.f = ones(2,6);
myfcn(Z);
Z.f = [X Y];
c = numel(Z.f);

function myfcn(~)
```

`myconcat2` assigns a 2-by-6 value to `Z.f`. The size of `Z.f` is fixed at 2-by-6 because `Z` is passed to `myfcn`. The result of the concatenation `[X Y]` is variable-size. The assignment `Z.f = [X Y]` results in an error because the left side of the assignment is fixed-size and the right side is variable-size.

To work around this incompatibility, you can use `coder.varsize` to declare that `Z.f` is variable-size.

```
function c = myconcat2(n, Y)
%#codegen
assert(n <= 2);
coder.varsize('Z.f');
X = zeros(n,2);
Z.f = ones(2,6);
myfcn(Z);
Z.f = [X Y];
c = numel(Z.f);

function myfcn(~)
```

xcorr Code Generation: Generate faster code for xcorr with long input vectors

For long input vectors, code generation for xcorr now uses a frequency-domain calculation instead of a time-domain calculation. The resulting code can be faster than in previous releases.

To use the `xcorr` function, you must have the Signal Processing Toolbox™ software.

More keyboard shortcuts for the MATLAB Function report

In R2016a, you can use keyboard shortcuts to perform the following actions in a MATLAB Function report.

Action	Default Keyboard Shortcut
Zoom in	**Ctrl+Plus**
Zoom out	**Ctrl+Minus**
Evaluate selected MATLAB code	**F9**
Open help for selected MATLAB code	**F1**
Open selected MATLAB code	**Ctrl+D**
Step backward through files that you opened in the code pane	**Alt+Left**
Step forward through files that you opened in the code pane	**Alt+Right**
Refresh	**F5**
Find	**Ctrl+F**

Your MATLAB preferences define the keyboard shortcuts associated with these actions. You can also select these actions from a context menu. To open the context menu, right-click anywhere in the report.

Zoom In	Ctrl+Plus
Zoom Out	Ctrl+Minus
Evaluate Selection	F9
Help on Selection	F1
Open Selection	Ctrl+D
Back	Alt+Left
Forward	Alt+Right
Refresh	F5
Find...	Ctrl+F
Page Source	

Code generation for Audio System Toolbox functions and System objects

See Audio System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Computer Vision System Toolbox functions and objects

See C code generation support in the Computer Vision System Toolbox™ release notes.

Image Processing Toolbox Code Generation: Generate code for additional functions

See C-code generation support for more than 20 functions, including regionprops, watershed, bweuler, bwlabel, bwperim, and multithresh using MATLAB Coder in the Image Processing Toolbox™ release notes.

Code generation for additional MATLAB functions

Specialized Math in MATLAB

- airy
- besseli
- besselj

See Specialized Math in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Trigonometry in MATLAB

- deg2rad
- rad2deg

See Trigonometry in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Interpolation and Computational Geometry in MATLAB

- interpn

See Interpolation and Computational Geometry in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Changes to code generation support for MATLAB functions

- Code generation now supports the `nanflag` option for sum, mean, median, min, max, cov, var, and std.
- Code generation for ismember no longer requires that the second input be sorted.

See Functions and Objects Supported for C and C++ Code Generation — Alphabetical List.

Code generation for additional Communications System Toolbox functions

- convenc
- dpskdemod

- dpskmod
- qammod
- qamdemod
- vitdec

See Communications System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional DSP System Toolbox

- audioDeviceWriter
- dsp.Differentiator
- designMultirateFIR
- dsp.SubbandAnalysisFilter
- dsp.SubbandSynthesisFilter

DSP System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Phased Array System Toolbox functions

- fogpl
- gaspl
- rainpl
- phased.BackscatterRadarTarget
- phased.LOSChannel
- phased.WidebandLOSChannel

See Phased Array System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for WLAN System Toolbox functions and System objects

See WLAN System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List

Units for MATLAB Function blocks

Simulink supports the specification of a unit property for data inputs and outputs of MATLAB Function blocks. Specify units for input and output data by using the **Unit (e.g., m, m/s^2, N*m)** parameter on the Ports and Data Manager.

During model update, Simulink checks for inconsistencies in units between input and output data ports and the corresponding Simulink signals.

In/Out Arguments: Specify same variable name for in/out arguments

In a MATLAB Function block, you can specify the same name for input and output arguments. For example, the block supports this code.

```
function y = fcn(y)
%#codegen

y = y + 1;
```
The corresponding input and output ports on the block have the same name, y.

`UserData` parameter available for storing values

Use `set_param()` and `get_param()` to store and retrieve values in the `UserData` parameter for MATLAB Function blocks.

Modeling Guidelines

High-Integrity Systems: Model object, file, and folder names

When you develop models for high-integrity systems, use the following guidelines for model object, file, and folder names:

- hisl_0031: File and folder names
- hisl_0032: Model object names

If you have a Simulink Verification and Validation™ license, you can use the following Model Advisor standards checks to verify compliance with high-integrity guideline hisl_0032: Model object names. The checks are available in the applicable Model Advisor **By Task** folder.

- DO-178C/DO-331: Check model object names
- IEC 61508, EN 50128 and ISO® 26262: Check model object names

Model Advisor

Additional functionality for Model Advisor check that checks for usage of partial structure

In R2016a, the Model Advisor check **Check for partial structure parameter usage with bus signals** has a new name, **Check structure parameter usage with bus signals**. The check uses a new programmatic ID, `mathworks.design.MismatchedBusParams`.

If you have Simulink Coder software, before you generate code from a model, use this check to discover potential inefficient typecasts due to mismatched data types. For information about the changes to the check functionality, see Model Advisor check for data type mismatches between bus elements and structure fields.

Compatibility Considerations

If you have scripts that programmatically run this check, consider using the new programmatic ID. The old programmatic ID is `mathworks.design.PartialBusParams`. For example, your scripts might use this command:

```
SysResultObjArray = ModelAdvisor.run({'myModel'},{'mathworks.design.PartialBusParams'})
```

Your scripts continue to work with the old ID. The new ID is `mathworks.design.MismatchedBusParams`.

In a future release, the check will appear in the **By Product** > **Simulink Coder** folder instead of the **By Product** > **Simulink** folder. You will need Simulink Coder software to run the check. If you do not have Simulink Coder software, consider removing the check from your scripts in R2016a.

S-Functions

`ssSetSolverNeedsReset` updates

The ssSetSolverNeedsReset macro is now needed for both fixed-step and variable-step ODE solvers. Simulink now monitors S-function continuous state changes without solver resets in normal and accelerated simulation modes for fixed-step simulation.

To improve performance, consider updating existing S-functions that correctly use `ssSetSolverNeedsReset` with the new macro, ssSetSkipContStatesConsistencyCheck.

`ssSetSkipContStatesConsistencyCheck`

The ssSetSkipContStatesConsistencyCheck macro is a new macro that lets you skip continuous state consistency checks. Consider using this macro for an S-function with continuous states that either does not change the state vector or changes it only in conjunction with ssSetSolverNeedsReset.

R2015aSP1

Version: 8.5.1

Bug Fixes

R2015b

Version: 8.6

New Features

Bug Fixes

Compatibility Considerations

Simulink Editor

Signal Line Healing: Click once to repair broken signal lines after deleting blocks

You can connect signal lines that broke when you deleted a block. When you delete a block that was connected to one input signal and one output signal, a prompt appears in its place. Click the prompt to connect the broken input and output signal lines to each other.
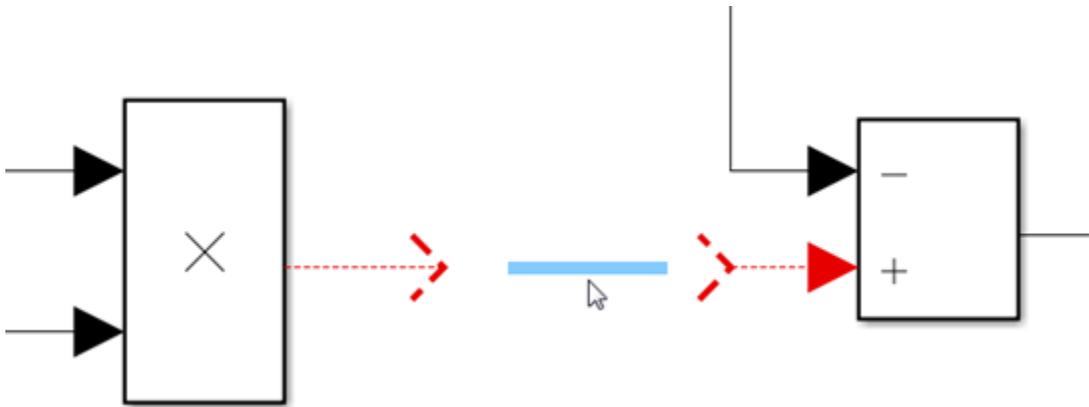
Multilingual Names and Comments: Use any language to write block names, signal names, and MATLAB Function comments

You can use Unicode characters for most textual content in Simulink, including names of blocks and signals as well as comments in MATLAB scripts or functions.

Programmatic removal of mask dialog box controls and mask parameters

You can use the following functions to remove dialog control elements and parameters from the mask dialog box:

- `removeParameter`: Removes a parameter from mask dialog box. For more information, see Simulink.Mask.removeParameter.

- `removeDialogcontrol`: Removes a dialog control element from mask dialog box. For more information, see Simulink.Mask.removeDialogControl.

Alternative view of library contents in Library Browser

You can now view blocks in the Simulink Library Browser in alphabetical order or in the order specified by the developer of the library. Right-click the blocks pane in the Library Browser, and select **Sort in library model order**. The blocks appear in the order specified by the developer of the library in the underlying library model. This setting takes effect on all your libraries and stays in effect from session to session.

To return to alphabetical order, right-click and select **Sort in alphabetical order**. For more information, see Reorder Blocks in Libraries.

Prompt to set key parameter when dragging a block from the Library Browser

When you drag a block from the Library Browser into your model, you are prompted to enter the key parameter. Setting the value this way lets you set the parameter without opening the dialog box.

Printing to Postscript and EPS file formats

You now cannot print Simulink models or Stateflow charts to postscript and EPS file formats.

Compatibility Considerations

If you want to print models or charts to a file, specify a .pdf extension for the output file.

Programmatic addition of areas and images in models

You can now add labeled areas and images to your Simulink model programmatically. For more information, see add_block.

Redesigned interface for Model Dependency Viewer

The Model Dependency Viewer tool has a new interface to view a model's dependencies on models and libraries. For more information, see Model Dependency Viewer.

Visual cue for undo and redo of block parameter value changes

After you undo or redo block parameter changes, a visual cue appears that shows the current values of the affected parameters. For example:

1 In the vdp model, for an Integrator block, change the **Initial condition** parameter value to 2 and select the **Wrap state** check box. Apply the changes.

2 Select **Edit** > **Undo Parameter Changes**. The cue shows that the parameters returned to their original values.

van der Pol Equation

Copyright 2004-2013 The MathWorks, Inc.

| InitialCondition | 0 |
| WrapState | off |
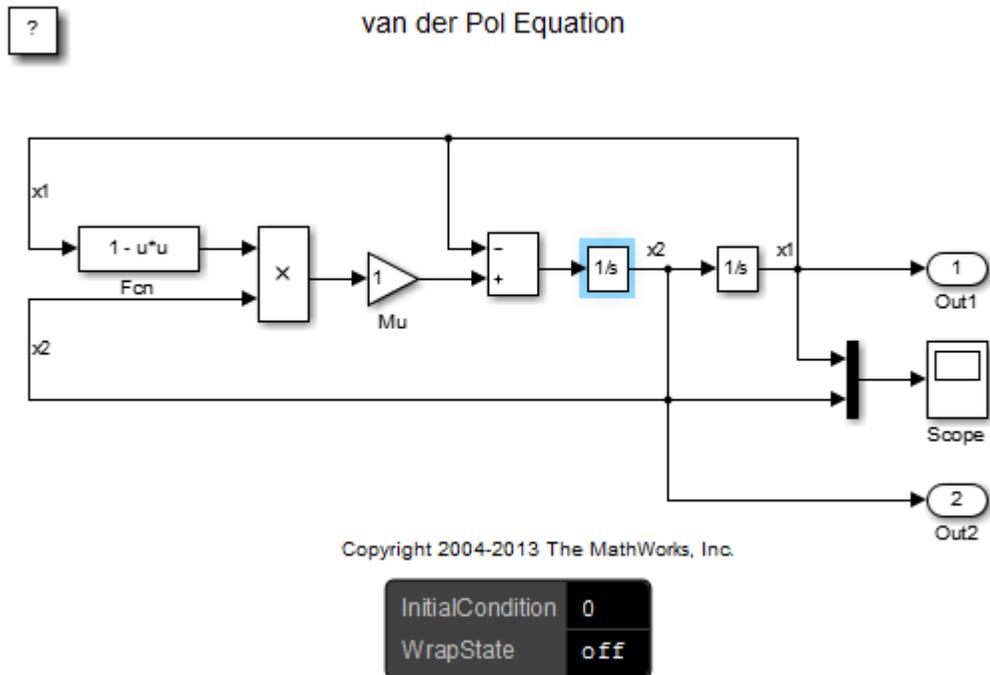
3 Select **Edit** > **Redo Parameter Changes**. The cue shows the restored parameters values.

van der Pol Equation

x1

1 - u*u
Fcn

×

1
Mu

−
+

x2
1/s

x1
1/s

1
Out1

x2

Scope

2
Out2
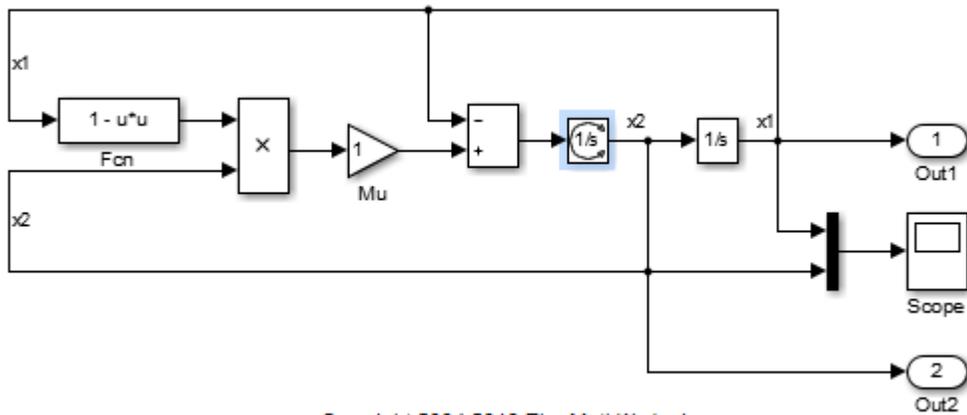
Copyright 2004-2013 The MathWorks, Inc.

| InitialCondition | 2 |
| WrapState | on |

For more information on Undo and Redo, see Interactive Model Building.

Simulation Analysis and Performance

New Interface for Scopes: View and debug signals with cursors and measurements

The Simulink Scope and Floating Scope blocks with the Scope Viewer were significantly upgraded. Simulink upgrades the scope blocks in a model that you previously created and saved when you reopen it.

Major new features include simulation data analysis and debugging tools directly within the Scope window, and tools similar to those of modern hardware bench-top oscilloscopes. See Scope block reference for a list of features.

Note: The Signal Statistics, Bilevel Measurements, and Peak Finder panels require a DSP System Toolbox or Simscape license.

Features from the previous scope were retained with some differences.

Capability	Previous Simulink Scope block	New Simulink Scope block
Number of input ports	No limit	Maximum 96
Number of axes (displays)	No limit	Maximum 16
Mapping axes (displays) to input ports	Number of axes equal to the number of input ports.	Number of displays and number of input ports are independent. Ports are assigned to displays sequentially, one port per display. If there are more ports than displays, the extra signals are plotted in the last display.
Mapping axes (displays) to signals	For the Floating Scope, select a specific axis and assign signals to axis using the Signal Selector.	For the Floating Scope, the number of displays and the number of attached signals are independent.

Capability	Previous Simulink Scope block	New Simulink Scope block
		Signals are assigned to displays sequentially, one signal per display. If there are more signals than displays, the extra signals are plotted in the last display.
Default logging format	`Structure with time`	`Dataset`
Default line color	Color order yellow, magenta, cyan, red, green, light gray.	Color order yellow, blue, orange, green, purple, cyan, magenta.
Model Explorer Search	Search `by Property Name`, `by Property Value`, or `by Dialog Prompt`.	Not possible to search property names or values.
Block and dialog parameters	The command `get_param(<scope block path>,'ObjectParameters')` returns a list of block parameters and dialog parameters.	The command `get_param(<scope block path>,'ObjectParameters')` returns a list of block parameters. There are no dialog parameters. The command `get_param(<scope block path>,'DialogParameters')` returns empty.
Last buffer Data Archiving	Last data buffer in external mode data archiving saved to the workspace by default	Data saved to workspace when Write Intermediate Results to WorkSpace is selected
Decimation and Sample Time	Sample time and decimation settings are not independent.	Sample time and decimation settings are independent.

Capability	Previous Simulink Scope block	New Simulink Scope block
Select signals for Floating Scope	Several Signal Selectors opened at once. Selection from the model diagram and from the Signal Selector poorly coordinated.	Only one Signal Selector opened at a time. Selecting signals with the Signal Selector disables selection from the model diagram. To select signals from the model diagram, unlock the Floating Scope, and then select signals.
Convert Scope block to Floating Scope block and back	Convert Scope to Floating Scope using a **Floating Scope** check box parameter.	Conversion not available. Get a Scope or a Floating Scope from the sinks library.
`Floating` property	Possible to convert a Scope block to a Floating Scope block using the command `set_param([model '/Scope'], 'Floating', 'on')`	The `Floating` property is read-only and set to `'off'`. Not possible to convert a Scope to a Floating Scope.
Hybrid signal plot style	Continuous elements of a hybrid signal (mux of continuous and discrete) plotted using a line and discrete elements plotted using stairs.	Signal elements of a hybrid signal plotted using lines. Behavior is consistent with a `Dataset` logging timeseries plot of hybrid signals.
Programmatic access and scripting	Use `set_param` and `get_param`. Changing Scope dialog parameters using `set_param` can put the Scope in an undefined state.	Use Scope Configuration object. For backward compatibility, `set_param` and `get_param` continue to support dialog parameters in the old Scope. However, these parameters are hidden.
Legend location	Location of a legend on a scope serialized and restored when a model is re-opened.	Placed legends anywhere within a scope, but the location of the legend is not serialized.

Capability	Previous Simulink Scope block	New Simulink Scope block
Zoom Mode	ZoomMode serialized and restored when a model is re-opened.	ZoomMode not serialized and not restored when a model is re-opened.
Scope blocks in conditionally executed subsystems	Scope data is not acquired when subsystem is not enabled. Lines drawn connecting gaps in plotted data.	Scope data is not acquired when subsystem is not enabled. Gaps in plotted data visible.

Fast Restart API: Programmatically run consecutive simulations more quickly

You can now enable fast restart from the command line using `set_param`. See Get Started with Fast Restart for more details.

You can also simulate a model in fast restart using `sim` and `cvsim` commands.

Previously, you could not use `cvsim` or `signalbuilder` with fast restart. In R2015b, you can now use these commands.

Auto solver that chooses solver for a model

Starting with R2015b, you can use the auto solver to select a solver and step size for simulating a model. The auto solver suggests a fixed-step or variable-step solver along with maximum step size based on the dynamics of the model. Select the auto solver in the solver pane and accept recommended settings in the solver information dialog box. For more information, see Use Auto Solver to Select a Solver.

Tunability of struct parameters in rapid accelerator mode

You can now tune struct parameters when you simulate a model in rapid accelerator mode. Previously you tuned struct parameters only in normal and accelerator modes. You do not need to regenerate the rapid accelerator code when you tune struct parameters.

Port value labels for nonvirtual buses and bus signals

To monitor bus signal data during simulation, you can use port value labels for nonvirtual buses. You can also display port labels for individual signals in a bus through a new interface. For more information, see Display Value for a Specific Port.

Visualization of inserted rate transition blocks

When Simulink performs automatic rate transition, you can now see the rate transition blocks that Simulink inserts in your model. You can also set the **Initial Condition** of these blocks and change block parameters for rate transfers. For more information, see Visualize Inserted Rate Transition Blocks.

Common format for saving states, output, and final states data and other logging and loading techniques

The default for the **Configuration Parameters** > **Data Import/Export** > **Format**> > parameter is now `Dataset`, which:

- Simplifies postprocessing of logged data
- Makes it easier to take advantage of features that require `Dataset` format

Extended support for root Inport loading using Dataset format in rapid accelerator

Loading root Inport data using Dataset format in rapid accelerator mode is now supported for underspecified and complex buses and arrays of buses.

Free MinGW-w64 compiler for running simulations on 64-bit Windows®

You can now use the MinGW-w64 compiler from TDM-GCC to run simulations on 64-bit Windows® hosts. You can run simulations in Accelerator and Rapid Accelerator modes, build model reference simulation targets and S-functions, and simulate MATLAB Function blocks. To download and install the compiler, see Install MinGW-w64 Compiler.

Component-Based Modeling

More flexible configuration of Application lifespan (days) parameter in a model reference hierarchy for simulation

The **Configuration Parameters** > **Optimization** > **Application lifespan (days)** parameter can now be different for a parent and child model in a model reference hierarchy during simulation. This flexibility can be useful during model reference development. However, for code generation, the settings for this parameter must be consistent throughout a model reference hierarchy.

Model Advisor checks for simplified initialization mode

The Model Advisor **Check consistency of initialization parameters for Outport and Merge blocks** check has been replaced with four new checks. These checks help to migrate your model to simplified initialization mode. For more information, see the Model Advisor task Migrating to Simplified Initialization Mode Overview.

Changes to export-function models

Export-function models include these changes for R2015b.

Configuration Parameter for Scheduling Checks

A new model referencing configuration parameter, **Enable strict scheduling checks for referenced export-function models**, lets you enable or skip checks on scheduling order and sample-time consistency. This check applies to referenced export-function models. For more information, see Enable strict scheduling checks for referenced export-function models.

Triggered Sample Time for Function-Call Subsystems

You can now specify a discrete sample time for the function-call root-level Inport block for function-call subsystems whose trigger block has **Sample time type** set to `Triggered`. For more information, see Sample Time for Function-Call Subsystems in Export-Function Models.

Execution Order of Function-Call Root-Level Inport Blocks

The algorithm to determine block execution order has changed. Root-level function-call Inport block execution order is now determined first by priority, then by sample time,

and then by the port number. For more information, see Execution Order for Function-Call Root-level Inport Blocks.

Compatibility Considerations

The algorithm to determine block execution order has changed. Specify low priority on the blocks you want to execute first.

Saving of list view parameters with `Simulink.ConfigSet.saveAs`

Previously, `Simulink.ConfigSet.saveAs` saved a configuration set that included the enabled parameters that appear in the category view in the Configuration Parameters dialog box. In R2015b, the method also saves the enabled parameters that appear only in the list view.

Project and File Management

Referenced Projects: Create reusable components for large modeling projects

You can organize large projects into components through the use of referenced projects. Componentization facilitates reuse, modular development, unit testing, and independent release of components. In R2015b, you can:

- Add new components to your project by referencing other projects.
- Use shortcuts to view and run files that belong to the referenced project.
- Extract a folder from a project and convert the folder into a referenced project.

For more information, see:

- Componentization of Large Projects
- Airframe Project Reference Example

Configuration Parameters List View: List, edit, and search all configuration parameters within your model

The Configuration Parameters list view provides a complete list of parameters in the model configuration set. The list view includes parameters available in the category view and parameters previously available only from the command line. To use the list view, click the **List** button at the top of the Configuration Parameters dialog box.
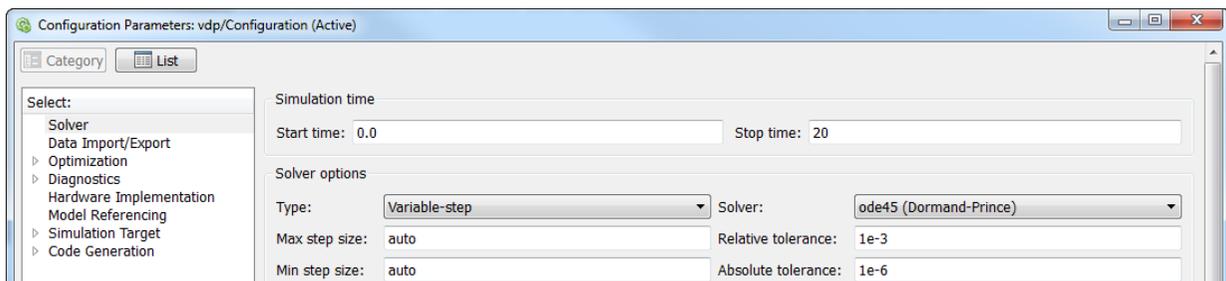
You can use the list view to:

- Search for specific parameters or filter parameters by category.

- Sort parameters.
- Edit parameter values.
- View parameter dependencies.
- Get parameter names for use in scripts.
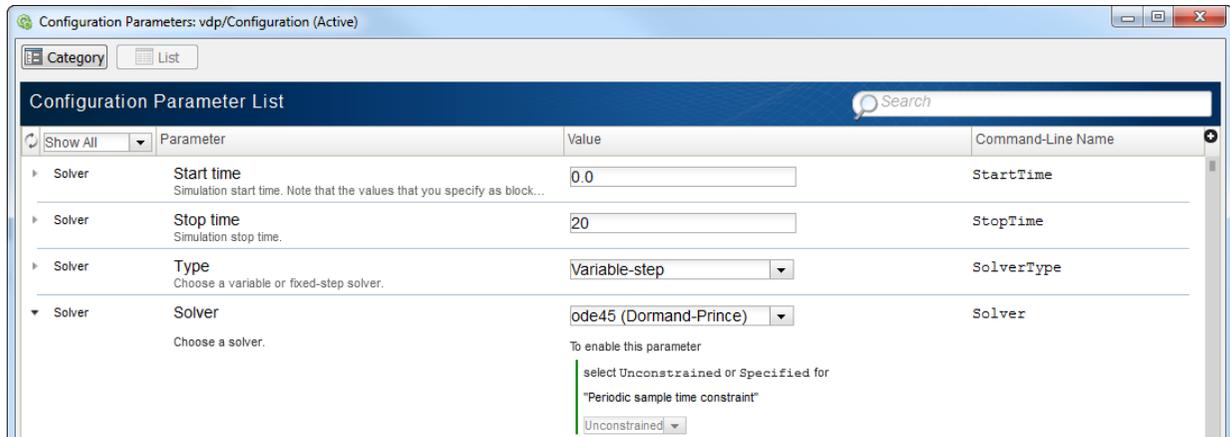
For more information on the Configuration Parameters dialog box and the list view, see Configuration Parameters Dialog Box Overview.

Project Creation from a Model: Quickly organize your model and all dependent files

Put your model and all dependent files into a Simulink project in three clicks. If your model has supporting files, then a project makes it easier to manage. If you have more than one model, version, or user, a project can help you. The project provides these advantages:

- Automate setup and shutdown tasks such as loading data and running scripts.
- Manage any required files (not just models and libraries, but all your supporting files, such as data, images, scripts, functions, data dictionaries, requirements docs, etc.).
- Manage your path when you open and close the project.
- Visualize all file dependencies.
- Use source control tools and compare versions.

In a Simulink model, select **File** > **Simulink Project** > **Create Project from Model**.

Simulink runs dependency analysis on your model to identify required files and suggest a project root location that contains all dependencies.

For details, see Create a Project from a Model.

Faster, Improved Dependency Analysis: Analyze projects several times faster, identify referenced project files, and view library blocks

Dependency analysis of typical projects is faster, and includes new capabilities:

- Identify dependencies on files in referenced projects.
- Trace library link dependencies to view library block names in the graph.

Analyzing all project files is now the default, which helps to ensure that you explore all dependencies.

The dependency analysis workflow is simplified so that you can perform all operations in the graph view using one toolstrip tab. The Simulink Project tree now has a single **Dependency Analysis** view instead of three views separating file selection for analysis, the table of results, and the Impact graph. The table view of files is still available as an option, but you can perform all operations from the Impact graph view.

For details, see Perform Impact Analysis.

Management of shadowed and dirty files

Simulink Project now helps you to avoid working on the wrong files by detecting shadowed files when you open a project. If loaded model files shadow your project model files, you are prompted to inspect or close them. To avoid working on the wrong files, close the files that shadow your project files before continuing to open your project. You can choose to show or close individual files, or close all files that shadow with one click. Previously, you only saw a warning in the command window when you opened a model that another loaded file shadowed.

Simulink Project also now helps you to manage unsaved changes when closing a project. When you close a project, it closes any project models that are open, unless they are dirty. If model files have unsaved changes, then a prompt appears . You can see all dirty files, grouped by project if you have referenced projects. To avoid losing work, you can save or discard changes by file, by project, or globally.

For details, see Manage Shadowed and Dirty Model Files.

Comparison of any pair of file revisions

In a Simulink project under source control, you can now select any pair of revisions to compare. This ability is useful for investigating differences in older changelists. You can now sort revisions by column headers in the Compare to Revision dialog box (e.g., by date, revision number, or author). The Compare to Revision dialog box has an improved layout, placing revision columns above the details pane, making it much easier to read.

Previously you could compare a revision only with your local file, making it more difficult to investigate older changes. You could not compare between other revisions, and you could not sort revisions.

For details, see Compare Revisions.

Updated power window example

The Power Window Control Project example has been updated to exercise more of the logic in the power window controller. The example uses the From Spreadsheet block to provide multiple sets of inputs to the controller. For more information, see Power Window.

Case-sensitive model and library names

Function Element Name	What Happens When You Use This Function Element	Use This Instead	Compatibility Considerations
Case-insensitive model or library name	Simulink uses the closest case-insensitive match, and warns	Specify exact case	Case-insensitive matching will be removed in a future release.

Warning for Model Info Configuration Manager

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
Model Info block Configuration Manager	When a model's configuration manager is active,	For a more flexible interface to source control tools, use	The Configuration Manager will be removed in a future

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
	you see a new warning when loading or saving the model, or when you access the Model Info block.	Simulink Project instead of the Model Info block.	release. For help, see the Upgrade Advisor checks: Identify Model Info blocks that use the Configuration Manager and Identify Model Info blocks that can interact with external source control tools.

Data Management

Interval Logging: Specify start and stop time intervals to log only the data you need

You can use the new **Configuration Parameters** > **Data Import/Export** > **Logging intervals** parameter to specify an interval for logging data for:

- Time
- States
- Output
- Signal logging
- The To Workspace block
- The To File block

Limiting logging to specified intervals:

- Allow you to view specific logged data without changing the model or adding complexity to a model
- Simplifies analysis of logged data
- Reduces simulation time
- Reduces memory consumption

Always-On Tunability: Tune all block parameters and workspace variables during a simulation

Prior to R2015b, you selected **Configuration Parameters** > **Optimization** > **Signals and Parameters** > **Inline parameters** to increase simulation speed by treating block parameter values as constants. In this case, you could not tune these parameters during simulation. To preserve tunability for individual block parameters, you used data objects or the Model Parameter Configuration dialog box. Alternatively, clearing **Inline parameters** enabled block parameter tunability by default while reducing simulation speed.

In R2015b, you can tune block parameters during simulation, and you retain the simulation speed benefit. You do not need to use data objects or the Model Parameter

Configuration dialog box to preserve parameter tunability during simulation. **Inline parameters** is now called **Default parameter behavior**. This configuration parameter does not affect simulation.

Prior to R2015b, when you displayed sample time colors, magenta indicated constant sample time. Now, magenta indicates blocks whose output values are constant. The term *constant* refers to blocks whose output values change only when you tune block parameters.

The table describes some block and system level changes caused by these enhancements.

Description of change	Behavior in R2015a	Behavior in R2015b
Constant block with sample time set to `inf`	**Inline parameters** selected: The block receives constant sample time. When you display sample time colors, the block appears magenta in color.	The block computes its output at the start of simulation and whenever you tune a block parameter. When you display sample time colors, the block appears magenta in color.
	Inline parameters cleared: The block inherits a sample time from its context. The block color indicates the inherited sample time.	
Relationship between sample time and parameter tunability	If you use a tunable parameter, such as a `Simulink.Parameter` object with a storage class other than `Auto`, to specify a numeric block parameter, the block inherits a sample time from its context.	The block computes its output at the start of simulation and whenever you tune a block parameter. When you display sample time colors, the block appears magenta in color.
Weighted sample time math block	**Inline parameters** selected: If any of the block inputs are constant, Simulink displays an error.	The block inherits a sample time from its context.
	Inline parameters cleared: The block inherits	

Description of change	Behavior in R2015a	Behavior in R2015b
	a sample time from its context.	

For more information about changes to the configuration parameter **Inline parameters** with respect to code generation, see the R2015b Simulink Coder release notes.

For more information about constant sample time, see Constant Sample Time.

Compatibility Considerations

- An error occurs in conditionally executed subsystems where:

  - A divide-by-zero operation occurs and all the blocks in the computation have constant output values.

  - The control input of an Index Vector block specifies an index that is out of range, and the block input has a constant value.

  Workaround: Use Upgrade Advisor to run **Check model for parameter initialization and tuning issues**. For more information, see Check model for parameter initialization and tuning issues.

- Prior to R2015b, if a model contained Level-2 MATLAB S-Function blocks, and you selected **Inline parameters**, these blocks allowed constant sample time by default. In R2015b, Level-2 MATLAB S-Function blocks inherit a sample time from their context by default.

  Workaround: To enable these blocks to allow constant sample time, explicitly set `SetAllowConstantSampleTime` to `true`.

- If you set **Default parameter behavior** to `Inlined`, and you log blocks that have tunable parameters, a mismatch can occur in the number of data points logged between the simulation and code generation workflows. However, the value logged remains constant where extra points are logged.

- In the code generation workflow, when the Merge block receives a constant value and non-constant sample times, one of these conditions must hold. Otherwise Simulink displays an error.

  - The source of the constant value is a grounded signal.

  - The source of the constant value is a constant block with a non-tunable parameter.

- There is only one constant block that feeds the Merge block.
- All other input signals to the Merge block are from conditionally executed subsystems.
- The Merge block and outport blocks of all conditionally executed subsystems should not specify any initial outputs.

For more information, see Merge.

- If you generate subsystem code for a subsystem that has any constant inputs, an error can occur.

  Workaround: Change the sample time at the source to an inherited or periodic sample time.

- Models that had **Inline parameters** cleared and simulated without error in releases prior to R2015b can have rate transition issues in R2015b. These issues occur because all constant blocks now receive constant sample time regardless of the setting for **Default parameter behavior**. This change can produce different sample time propagation results.

  Workaround: Use Upgrade Advisor to run **Check model for parameter initialization and tuning issues**. For more information, see Check model for parameter initialization and tuning issues.

- Suppose you generate code for a model where these conditions are true:

  - The default parameter behavior is tunable.
  - A constant block executes with a downstream rate that is not the fastest rate.

  The logged data appears to be delayed. There is a delay equal to one step of the slower rate that the block executes at. Simulink displays a warning during code generation.

  This issue does not affect signal logging during simulation.

- If you write TLC code to generate code from an inlined S-function, and if the TLC code contains an `Outputs` function, you must modify the TLC code if all of these conditions are true:

  - An output port uses or inherits constant sample time. The output port has a constant value.
  - The S-function is a multirate S-function or uses port-based sample times.

In this case, the TLC code must generate code for the constant-valued output port by using the function `OutputsForTID` instead of the function `Outputs`. For more information, see Specifying Constant Sample Time (Inf) for a Port.

- Previously, if you selected **Inline parameters**, parameters that used the storage class `Auto` (including numeric MATLAB variables) were not tunable during simulation. The code generated for models referenced in accelerator mode and the code generated for top models or freestanding models in rapid accelerator mode inlined the numeric values of the variables. In R2015b, these parameters are tunable, and the code preserves them.

    - These parameters must exist throughout simulation. For example, if you reference a model in accelerator mode, you cannot use the referenced model `CloseFcn` callback to clear the parameters. Consider using the callback of the parent model instead. Alternatively, store the parameters in a Simulink data dictionary.

    - Suppose that you use a variable `myVar` in an inconsistent way, for example:

        - You use `myVar` as the value of the **Gain** parameter of a Gain block in a referenced model `submodel1`. The Gain block applies a data type, such as `int32`, to `myVar`.

        - You also use `myVar` in a Gain block in the referenced model `submodel2`. This Gain block applies a different data type, such as `int16`, to `myVar`.

        In R2015b, you cannot use `myVar` in an inconsistent way across these referenced models because `myVar` is tunable. To simulate the models, resolve the inconsistent usage. For example, modify the referenced models `submodel1` and `submodel2` so they apply the same data type to `myVar`.

- Previously, the code generated in rapid accelerator mode did not preserve expressions, such as `myVar + myOtherVar * 5`, that you used to specify block parameter values. In R2015b, the code preserves these expressions.

    - If a block parameter value references workspace variables, you cannot change the block parameter value during rapid accelerator simulation. Instead, you can tune the values of the referenced variables. If you use a script to change the block parameter value during simulation, modify the script so it changes the values of the variables instead.

    - Suppose that a For Each subsystem has a mask parameter `a` whose value is the three-element array `[1 2 3]`. Suppose that the subsystem partitions the parameter `a` three times so that each iteration of the subsystem uses one of the

elements in the array. You cannot use the expression `15.23^a` to specify a block parameter value inside the subsystem because the expression contains an operator that the code generator does not support.

To simulate the model, consider implementing the expression as a block algorithm instead. For more information about the operators that the code generator supports in tunable expressions, see Tunable Expression Limitations.

- Previously, you could select **Inline parameters** to prevent mask initialization code from executing during simulation. This technique prevented code that made changes to the model from executing. If the changes generated errors during simulation, for example by attempting to add blocks to the model, selecting **Inline parameters** prevented the errors.

  In R2015b, all mask initialization code executes during simulation. To avoid errors during simulation, consider wrapping the mask initialization code in additional code that prevents it from executing during simulation.

Arrays of structures as parameters

You can use arrays of structures as parameters to:

- Initialize arrays of bus signals that pass through blocks like Unit Delay. Arrays of buses help you to reduce signal line density in model diagrams. For more information, see Specify Initial Conditions for Bus Signals.
- Specify the **Constant value** parameter in a Constant block. You can use this technique to compactly represent multiple constant-valued signals as an array of buses. For an example, see the Constant block.
- Group workspace variables into a single variable whose value is an array of structures. You can use this technique to organize related variables and reduce workspace clutter. For more information about structure parameters, see Organize Related Parameters in Structures and Arrays of Structures.
- Parameterize a For Each Subsystem, which can help you repeat an algorithm over multiple inputs. The subsystem can partition mask parameters that are arrays of structures. For an example, see Repeat an Algorithm Using a For Each Subsystem.

Improved methods to create custom data objects

Model Explorer

Previously, to create data objects such as `Simulink.Parameter` and

`Simulink.Signal`, you used the Model Explorer buttons Add Parameter [image] and Add

Signal [image]. To create data objects from a package other than `Simulink`, you specified the package using Simulink Preferences.

In R2015b, to select a data class other than those in the package `Simulink`, you can use the new arrows beside these buttons. You can select any data class whose definition is on the MATLAB path, including custom classes such as `myPackage.myParameter`.

For more information about creating data objects, see Data Objects.

Data Object Wizard

Previously, to create custom data objects using the Data Object Wizard, you specified a default data class package using Simulink Preferences.

In R2015b, to create objects from data classes in any package, including your own packages, you can select the classes using the Data Object Wizard.

For more information about creating data objects using the Data Object Wizard, see Create Data Objects for a Model Using Data Object Wizard.

No creation of parameter objects for mask initialization code

You now cannot use mask initialization code to create parameter objects. Parameter objects are objects of the class `Simulink.Parameter` and of any of its subclasses that you create. For more information about block masking and using MATLAB code to initialize a mask, see Initialize Mask.

Compatibility Considerations

If you use existing mask initialization code that creates parameter objects, you must edit the code so that it does not create parameter objects.

Sample time for signal logging

In the Signal Properties dialog box, you can use the new **Sample Time** option for a signal marked for signal logging. This option:

- Maintains the separation of design and testing, because you do not need to insert a Rate Transition block to have a consistent sample time for logged signals
- Reduces the amount of logged data for continuous time signals, for which setting decimation is not relevant
- Eliminates the need to postprocess logged signal data for signals with different sample times

For details, see Set Sample Time for a Logged Signal.

Same format for logging states, output, and final states as used for other logging and loading techniques

The default for the **Configuration Parameters** > **Data Import/Export** > **Format** parameter is now `Dataset`, which:

- Simplifies post processing of logged data
- Makes it easier to take advantage of features that require `Dataset` format

Root Inport loading in rapid accelerator mode using Dataset format

In rapid accelerator mode, using `Dataset` format to load root Inport blocks now supports:

- Specifying a single `Dataset` object, as an alternative to specifying a long comma-separated list using the **Configuration Parameters** > **Data Import/Export** > **Input** parameter
- Loading of buses and arrays of buses, including underspecified buses and arrays of buses
- Loading of fixed-point and `enum` data
- No code regeneration for new data

Logged signals with propagated names

Signal logging and root Outport block logging data for a signal captures the propagated signal name if the logging format is `Dataset` and:

- For signal logging, you:

  - Mark one or more signals for signal logging and in the Signal Properties dialog box select **Show Propagated Signals**.
  - Enable **Configuration Parameters** > **Data Import/Export** > **Signal logging**.

- For root Outport block logging, you select **Configuration Parameters** > **Data Import/Export** > **Output**.

The `Simulink.SimulationData.Signal` class has a new `PropagatedName` property for displaying the propagated signal name. The `Simulink.SimulationData.Dataset` class displays propagated signal names as a comment (the propagated signal name preceded by a percent sign (%)). In the logged data, the propagated signal name does not include angle brackets (<>).

Tolerance for data type mismatch between bus elements and structure fields

Previously, when you used a MATLAB structure to initialize a bus signal, or to drive a bus signal using a Constant block, you matched the data types of the structure fields with those of the bus signal elements.

In R2015b, you do not need to match the data types when you simulate a model. You can use doubles to specify the structure field values, and use the bus signal elements to control the data types. Prior to simulation, the bus elements cast the structure field values.

However, for other applications such as creating tunable initial conditions in the generated code, you must match the data types.

To decide whether to explicitly specify field data types, see Decide Whether to Specify Data Types for Structure Fields.

Summary of changes made to data dictionary

If you use a Simulink data dictionary to store the variables that a model uses, you can display a summary of all of the unsaved changes that you made to the dictionary. You can use this technique to:

- Track your changes while you create and modify dictionary entries.
- Decide which changes to keep or discard before you close an unsaved dictionary.

- Recover variables that you deleted.
- Recover dictionary references that you removed.

For an example, see View and Revert Changes to Entire Dictionary.

Rename All in Goto blocks

You can use the **Rename All** button in Goto block dialog boxes to quickly rename the corresponding tag in From and Goto Tag Visibility blocks. For more information, see the Goto block.

Change to visibility of `SamplingMode` property of signal objects

The `Simulink.Signal` property `SamplingMode` is hidden if you set it to `'auto'`, the default value. Previously, when you used a `Simulink.Signal` object to define a data store, you set `SamplingMode` to `'Sample based'`. You can now leave the property at the default value, `'auto'`.

If you use existing code that creates `Simulink.Signal` objects and changes the value of the `SamplingMode` property, the property is not hidden for these objects.

This change supports recent changes to frame-based processing with DSP System Toolbox. For more information, see Sample- and Frame-Based Concepts in the DSP System Toolbox documentation.

Continued availability of `Simulink.saveVars`

The R2014a Release Notes state that the function `Simulink.saveVars` will be removed in a future release. As of R2015b, you can continue to use this function. Existing scripts that use `Simulink.saveVars` do not generate warnings.

However, `Simulink.saveVars` is not recommended. To save workspace variables to a MATLAB script, use matlab.io.saveVariablesToScript instead of `Simulink.saveVars`.

`Simulink.SimulationData.Dataset` updates

Simulink.SimulationData.Dataset.find is a new method that finds elements with specified property names and values. For documentation on this method, in the MATLAB Command Window, type `help Simulink.SimulationData.Dataset.find`.

Edit Input button is now Connect Input

The **Input** > **Edit Input** button on the Configuration Parameters Data Import/Export pane is now **Connect Input**.

Legacy Code Tool support for conditional outputs

When you use the legacy_code function, you can now specify whether the legacy code conditionally writes the output ports. Use the `outputsConditionallyWritten` S-function option. If `true`, the generated S-function specifies that the memory associated with each output port cannot be overwritten and is global (`SS_NOT_REUSABLE_AND_GLOBAL`). If `false`, the memory associated with each output port is reusable and is local (`SS_REUSABLE_AND_LOCAL`). By default, the value is `false` (`0`).

Connection to Hardware

Raspberry Pi 2 Support: Run Simulink models on Raspberry Pi 2 Model B hardware

You can use the Simulink Support Package for Raspberry Pi Hardware with Raspberry Pi 2 Model B hardware.

Arduino Yun: Design and run Simulink models on Arduino Yun hardware

You can use the Simulink Support Package for Arduino Hardware with Arduino Yun hardware.

Hardware Implementation Selection: Quickly generate code for popular embedded processors

Specification of hardware configurations has been simplified. Top-level Configuration Parameters dialog box panes, **Run on Target Hardware** and **Coder Target**, have been removed. Parameters previously available on those panes now appear on the **Hardware Implementation** pane. A parameter has also moved from the **Code Generation** pane to the **Hardware Implementation** pane.

This list summarizes the R2015b changes and new behavior:

- By default, the **Hardware Implementation** pane lists **Hardware board**, **Device vendor**, and **Device type** parameter fields only.

- If you use Simulink without a Simulink Coder license, initially parameters on the **Hardware Implementation** pane are disabled. To enable them, click **Enable hardware specification**. The parameters remain enabled for the current MATLAB session.

- By default, the **Hardware board** list includes: None or Determine by Code Generation system target file, and Get Hardware Support Packages. After installing a hardware support package, the list also includes corresponding hardware board names.

- If you select a hardware board name, parameters for that board appear in the dialog box display.

- Lists for the **Device vendor** and **Device type** parameters have been updated to reflect hardware that is available on the market. The default **Device vendor** and **Device type** are `Intel` and `x86-64 (Windows64)`, respectively.

- If Simulink Coder is installed, the revised **Hardware Implementation** pane identifies the system target file that you selected on the **Code Generation** pane.

- A **Device details** option provides a way to display parameters for setting details such as number of bits and byte ordering.

- To specify target hardware for a Simulink support package, select a value from **Configuration Parameters** > **Hardware Implementation** > **Hardware board**. Before R2015b, you selected **Tools** > **Run on Target Hardware** > **Prepare to run**. Then, you selected a value from **Configuration Parameters** > **Run on Target Hardware** > **Target hardware**.

- To specify target hardware for an Embedded Coder support package, select a value from **Configuration Parameters** > **Hardware Implementation** > **Hardware board**. Before R2015b, you selected a value from **Configuration Parameters** > **Code Generation** > **Target hardware**.

- The **Test hardware** section was removed. Configure test hardware from the Configuration Parameters list view. Set `ProdEqTarget` to `off`, which enables parameters for configuring test hardware details.

- If you set **Configuration Parameters** > **Code Generation** > **System target file** to `ert.tlc`, `realtime.tlc`, or `autosar.tlc`, the default setting for **Configuration Parameters** > **Hardware Implementation** > **Hardware board** is `None`. If you set **System target file** to value other than `ert.tlc`, `autosar.tlc`, or `realtime.tlc`, the default setting for **Hardware board** is `Determine by Code Generation system target file`.

For more information, see Hardware Implementation Pane.

Compatibility Considerations

Starting in R2015b:

- By default, the **Hardware Implementation** pane lists **Hardware board**, **Device vendor**, and **Device type** parameter fields only. To view parameters for setting details, such as number of bits and byte ordering, click **Device details**.

- The following devices appear on the **Hardware Implementation** pane only for models that you create with a version of the software earlier than R2015b. These devices are considered legacy devices.

Generic, 32-bit Embedded Processor
Generic, 64-bit Embedded Processor (LP64)
Generic, 64-bit Embedded Processor (LLP64)
Generic, 16-bit Embedded Processor
Generic, 8-bit Embedded Processor
Generic, 32-bit Real-Time Simulator
Generic, 32-bit x86 compatible
Intel, 8051 Compatible
Intel, x86–64
SGI, UltraSPARC Iii

In R2015b, if you open a model configured for a legacy device and change the **Device type** setting, you cannot select the legacy device again.

- Device parameter **Signed integer division rounds to** is set to `Zero` instead of `Undefined`. For some cases, numerical differences can occur in results produced with `Zero` versus `Undefined` for simulation and code generation.

  This change does not apply to legacy devices.

- To associate a new model with an existing configuration set that has the following characteristics, configure the model to use the same hardware device as the existing model.

  - The model consists of a model reference hierarchy. Models in the hierarchy use different configuration sets.
  - The existing configuration set was saved as a script and associated with a configuration set variable.

  If the code generator detects differences in device parameter settings, a consistency error occurs. To correct the condition, look for differences in the device parameter settings, and make the appropriate adjustments.

Signal Management

Virtual bus signal inputs to blocks that require nonbus or nonvirtual bus input

Starting in R2015b, virtual bus signal input to blocks that require nonbus or nonvirtual bus input can cause an error. The error occurs when a virtual bus input signal is generated by a block that specifies a bus object as its output data type. Examples of blocks that can specify a bus object as its output data type include a Bus Creator block or a root Inport block. The blocks that cause an error when they have a virtual bus input in this situation are:

- Assignment
- Delay

  This block causes an error only if you set an initial condition from the dialog that is a MATLAB structure or zero and you specify a value for **State name**.

- Permute Dimension
- Reshape
- Selector
- Unit Delay

  This block causes an error only if you set an initial condition from the dialog that is a MATLAB structure or zero and you specify a value for **State name**.

- Vector Concatenate

Generating an error when this situation occurs helps to promote consistent output by requiring proper input to these blocks.

To check for proper virtual bus usage and for Mux blocks used to create bus signals, use the new Upgrade Advisor **Check bus usage** check.

The new Upgrade Advisor check inserts a Bus to Vector block to attempt to convert virtual bus input signals to vector signals. For issues that the Upgrade Advisor identifies but cannot fix, modify the model manually. For details, see Correct Buses Used as Muxes and Prevent Bus and Mux Mixtures.

Compatibility Considerations

In R2015b, a virtual bus input signal to the affected blocks causes an error message, regardless of the **Configuration Parameters** > **Diagnostics** > **Connectivity** > **Bus signal treated as vector** setting.

In models created in releases earlier than R2015b, if the **Bus signal treated as vector** diagnostic setting is `error`, there is no compatibility issue when you run the model in R2015b.

To help you to address this issue, use the Upgrade Advisor **Check bus usage** check.

Entire nested bus assignment for Bus Assignment block

If a bus signal input to a Bus Assignment block contains a nested bus signal, then you can assign or select the nested bus signal as a whole. However, the nested bus cannot be nested inside of an array of buses. Before R2015b, you had to assign each signal in the nested bus.

Support for entire nested bus signal assignment reduces:

- The number of blocks in a diagram
- The maintenance effort

For details, see Bus Assignment.

Block Enhancements

Waveform Generator Block: Define and output arbitrary waveform signals

The Waveform Generator block outputs waveforms using signal notations. This block is located in the Sources sublibrary.

From Spreadsheet Block: Read signal data into Simulink from a spreadsheet

The From Spreadsheet block reads data from spreadsheets. This block is located in the Simulink Extras/Additional Sources sublibrary.

MATLAB System block support for nonvirtual buses

The MATLAB System block now supports nonvirutal buses. For more information, see Nonvirtual Buses and MATLAB System Block. For an example, see Using Buses with MATLAB System Blocks.

Inport block update

The Inport block now has a **Connect Input** button. Use this button to import, visualize, and map signal and bus data to root-level inports using the Root Inport Mapping tool.

From File updates for file name and signal preview

The From File block **File name** parameter has been updated to include:

- A file browse button
- A view button that lets you plot and inspect signals.

Inheriting of continuous sample time for discrete blocks

The **Configuration Parameters** > **Diagnostics** > **Sample Time** > **Discrete used as continuous** diagnostic was removed. If a discrete block (such as the Unit Delay block) inherits a continuous sample time, the block returns an error.

Evenly spaced breakpoints in Lookup Tables

A new format is available to specify evenly spaced breakpoints in the Prelookup and n-D Lookup Table blocks. For more information, see the **Specification** parameter in the Prelookup block reference page and the **Breakpoints specification** parameter in the n-D Lookup Table reference page.

Integrator block: Wrapped states for modeling rotary and cyclic state trajectories

The Integrator block has been enhanced to support wrapped states when modeling rotary, cyclic, or periodic state trajectories. This support for wrapping states provides these advantages.

- It eliminates simulation instability when your model approaches large angles and large state values.
- It reduces the number of solver resets during simulation and eliminates the need for zero-crossing detection, improving simulation time.
- It eliminates large angle values, speeding up computation of trigonometric functions on angular states.
- It improves solver accuracy and performance and enables unlimited simulation time.

Variant Subsystem block: Enhanced option for generating preprocessor conditionals

The option **Generate preprocessor conditionals** in the Variant Subsystem block parameters dialog box has been replaced with the option **Analyze all choices during update diagram and generate preprocessor conditionals**. When you select this option, Simulink analyzes all variant choices during an update diagram or simulation. This analysis provides early validation of the code generation readiness of all variant choices.

Compatibility Considerations

Previously, when the option to generate preprocessor conditionals was switched on, Simulink analyzed all variant choices only during the code generation phase. Now, Simulink performs this analysis during the update diagram phase. As a result, errors

that you would normally see during code generation appear earlier, during an update diagram.

Constant sample time in S-function blocks

MATLAB S-function blocks no longer support a constant sample time (`Inf`) for their ports by default.

Compatibility Considerations

To allow ports in your MATLAB S-function blocks to have a sample time of `Inf`, use the SetAllowConstantSampleTime command.

MATLAB Function Blocks

Calling of Simulink Functions

You can call a Simulink Function block from inside of a MATLAB Function block. You can also call Stateflow functions with **Export Chart Level Functions (Make Global)** and **Allow exported functions to be called by Simulink** checked in the chart Properties dialog box.

Nondirect feedthrough in MATLAB Function blocks

By default, MATLAB Function blocks have direct feedthrough enabled. To disable, in the Ports and Data Manager, you can now clear the **Allow direct feedthrough** check box. Nondirect feedthrough enables semantics to ensure that outputs rely only on current state.

To use nondirect feedthrough, do not program outputs to rely on inputs or updated persistent variables. For example, do not use the following code in a nondirect feedthrough block:

```
counter = counter + 1;   % update state
output = counter;        % compute output based on updated state
```

Instead, use code such as:

```
output = counter;        % compute output based on current state
counter = counter + 1;   % update state
```

Also, nondirect feedthrough semantics require function inlining. Do not disable inlining.

Using nondirect feedthrough enables you to use MATLAB Function blocks in a feedback loop and prevent algebraic loops.

Overflow and data range detection settings unified with Simulink

Previously, you controlled overflow detection in MATLAB Function blocks with the configuration parameter **Detect wrap on overflow**. This parameter was on the **Simulation Target** pane in the Model Configuration Parameters dialog box.

You now control the overflow detection for MATLAB Function blocks with the configuration parameter **Wrap on overflow**. This parameter is on the **Diagnostics:**

Data Validity pane in the Model Configuration Parameters dialog box. Choose one of these settings: `none`, `warning`, and `error`.

Previously, you controlled data range error checking in MATLAB Function blocks in the editor, with **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Data Range**.

You now control data range checking with the configuration parameter **Simulation range checking**. This parameter is on the **Diagnostics: Data Validity** pane in the Model Configuration Parameters dialog box. Choose one of these three settings: `none`, `warning`, and `error`.

See Diagnostics Pane: Data Validity.

Compatibility Considerations

When you open a model with a MATLAB Function block saved in a previous release, a change in behavior is possible. These options are no longer valid.

- **Detect wrap on overflow** on the **Simulation Target** pane of the Model Configuration Parameters dialog box
- **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Data Range** in the Stateflow editor

In R2015b, the software determines overflow detection and data range by the setting of these Simulink options on the **Diagnostics: Data Validity** pane in the Model Configuration Parameters dialog box.

- **Wrap on overflow**
- **Simulation range checking**

Set each configuration parameter to `none`, `warning`, or `error`. The software displays a warning if the previous MATLAB Function block options saved with the model were set differently than the current Simulink options.

The command-line parameter `SFSimOverflowDetection` is no longer valid. Use `IntegerOverflowMsg` instead. The API parameter `Debug.RunTimeCheck.DataRangeChecks` is no longer valid. Use the command-line parameter `SignalRangeChecking` instead.

When you save a current model with a MATLAB Function block in a previous version, the current Simulink parameters are saved. Both the MATLAB Function block overflow and data range parameters are saved as selected.

No frame-based sampling mode for outputs

In R2015b, you can no longer set the sampling mode of outputs to frame based in MATLAB Function blocks. Models created in previous releases using frame-based sampling mode continue to behave as they did before this release.

Code generation for cell arrays

In R2015b, you can generate code from MATLAB code that uses cell arrays.

The code generation software classifies a cell array as *homogeneous* or *heterogeneous*. This classification determines how a cell array is represented in the generated C/C++ code. It also determines how you can use the cell array in MATLAB code from which you generate C/C++ code. See Homogeneous vs. Heterogeneous Cell Arrays.

As long as you do not specify conflicting requirements, you can control whether a cell array is homogeneous or heterogeneous. See Control Whether a Cell Array is Homogeneous or Heterogeneous.

For information about restrictions when you use cell arrays in a MATLAB Function block, see Cell Array Requirements and Limitations for Code Generation.

LAPACK calls during simulation for algorithms that call linear algebra functions

To improve the simulation speed for MATLAB Function block algorithms that call linear algebra functions, the simulation software can now call LAPACK functions. If the input arrays for the linear algebra functions meet certain criteria, the simulation software generates calls to relevant LAPACK functions.

LAPACK is a software library for numerical linear algebra. MATLAB uses this library in some linear algebra functions such as `eig` and `svd`. The simulation software uses the LAPACK library that is included with MATLAB.

For information about the open source reference version of LAPACK, see LAPACK — Linear Algebra PACKage.

Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions

Image Processing Toolbox

bwareaopen	houghpeaks	immse	integralBoxFilter
grayconnected	imabsdiff	imresize	psnr
hough	imcrop	imrotate	
houghlines	imgaborfilt	imtranslate	

See Image Processing Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Computer Vision System Toolbox

- insertText
- extractLBPFeatures

See Computer Vision System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Statistics and Machine Learning Toolbox functions

- kmeans
- randsample

See Statistics and Machine Learning Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional MATLAB functions

Data Types in MATLAB

- cell
- fieldnames
- struct2cell

See Data Types in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

String Functions in MATLAB

- iscellstr
- strjoin

See String Functions in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Communications System Toolbox, DSP System Toolbox, and Phased Array System Toolbox functions and System objects

Communications System Toolbox

comm.CoarseFrequencyCompensator

See Communications System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

DSP System Toolbox

- dsp.IIRHalfbandDecimator
- dsp.IIRHalfbandInterpolator
- dsp.AllpassFilter

See DSP System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Phased Array System Toolbox

- phased.TwoRayChannel
- phased.GCCEstimator
- phased.WidebandRadiator
- phased.SubbandMVDRBeamformer
- phased.WidebandFreeSpace
- gccphat

See Phased Array System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

R2015a

Version: 8.5

New Features

Bug Fixes

Compatibility Considerations

Simulink Editor

Bus Smart Editing Cue: Automatically create a bus from a set of signals

You can select multiple blocks or signals in an area of a model to create a bus for.
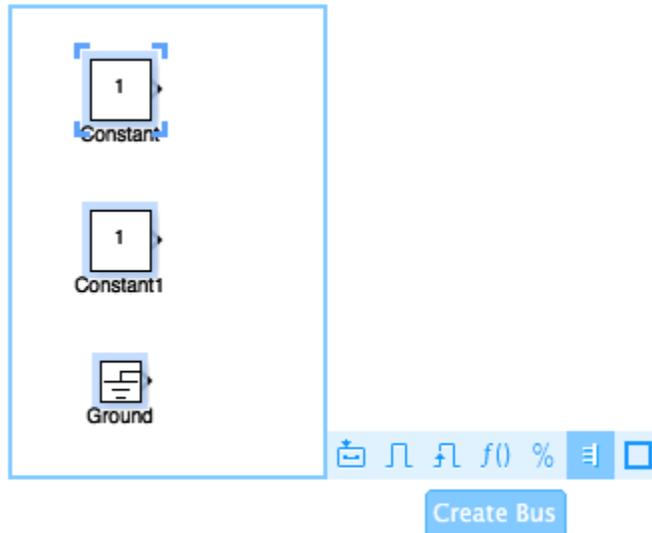
Using the **Create Bus** button to create a bus automatically performs these actions:

- Creates a Bus Creator block with the right number of inputs
- Resizes the Bus Creator block to an appropriate size for the number of input signals
- Orients the Bus Creator block to the direction of the input signals
- Connects the signals to the Bus Creator block

Area Annotations: Call out and separate regions of interest in model

You can select multiple objects in an area of a model to annotate an area of interest in your model.

1   In the Simulink Editor, drag to select the area of the model you want to create an area around.

2   In the action bar, click **Create Area**.
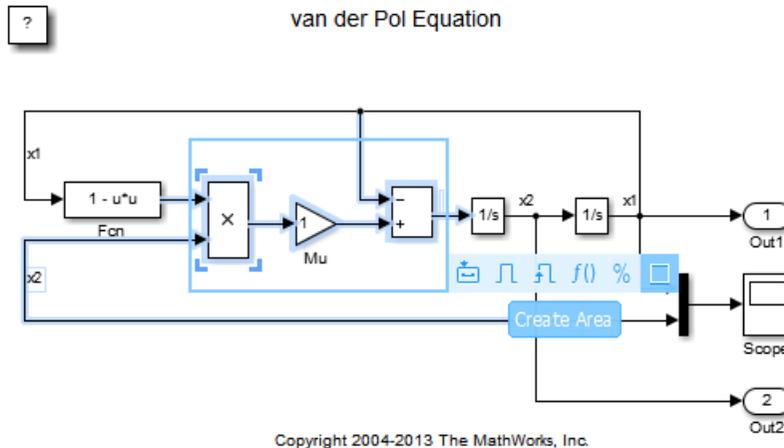
For more information, see Box and Label Areas of a Model.

Perspectives Controls: Access alternative views of your model, such as harness and interface views

You can switch to different views of a model, such as harness view to view test harnesses and interface view to view interfaces of your system.

1   In the Simulink Editor, click the control in the lower-right corner.
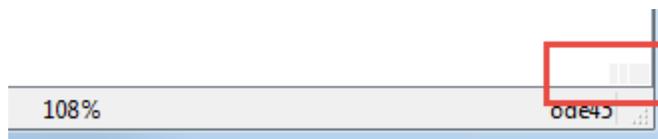
2   Click the view you want to see.

Saving of viewmarks in Simulink models

You can now store viewmarks with your model. For more information, see Use Viewmarks to Save Views of Models.

Highlighting of the subsystem you navigated from

When you navigate out of a subsystem, Simulink now highlights that subsystem temporarily. The highlight helps you to identify the subsystem you were working in most recently.
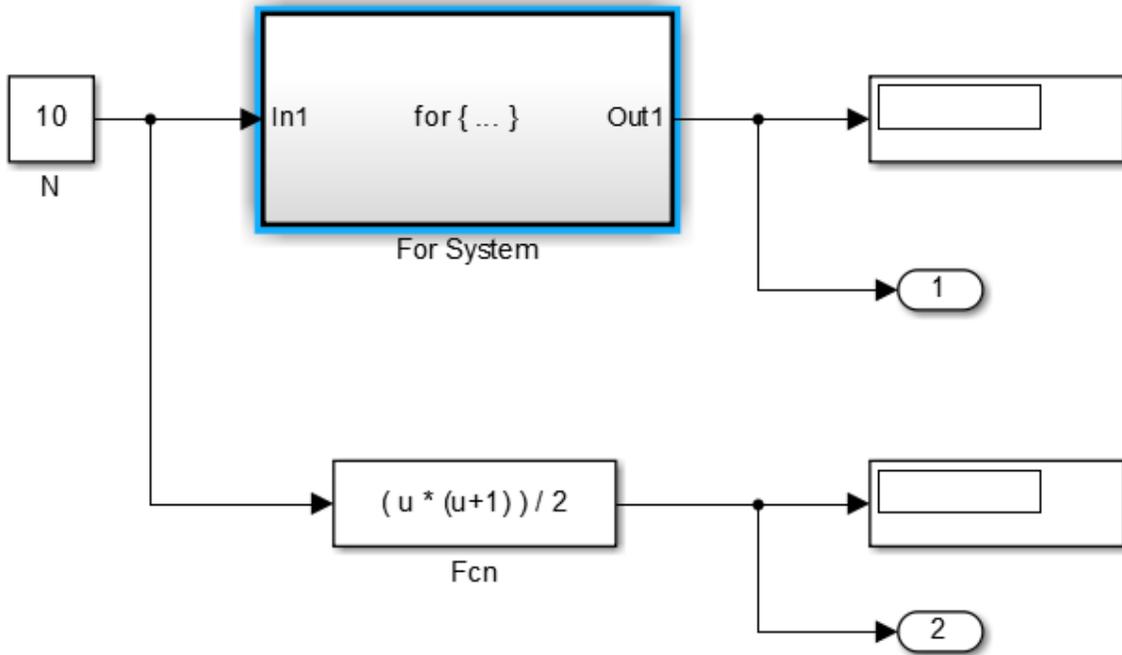
Annotation connector colors and width

You can now specify the color and width of annotation connectors. Right-click the annotation connector and use the **Format** menu.

To create an annotation connector, see Add Lines to Connect Annotations to Blocks

Undo and redo of block parameter value changes

Block parameter changes using block dialog boxes are now undoable. When you use **Undo** after you make changes to block parameter values, the values revert and the affected block briefly appears highlighted. A **Redo** command as the next action restores the changes and also briefly highlights the block. For more information on Undo and Redo, see "Interactive Model Building".

Display of product name in model title bar

The Simulink Editor now displays the word **Simulink** next to the model name in the model title bar.

Simulation Analysis and Performance

Dashboard Block Library: Tune and test simulations with graphical controls and displays

The Dashboard block library contains controls and displays that enable you to tune block parameters or monitor signals. The controls can be active during simulation so you can tune parameters and optimize your model while it is running or paused. The blocks are added directly to your Simulink model canvas and connected to signals or blocks. The Dashboard block library includes:

- Knobs
- Switches
- Gauges
- Dashboard scope
- Lamp

Algebraic Loop Highlighting: Find and remove algebraic loops in the model to boost simulation speed

You can now identify and remove all algebraic loops in your model using the new highlight feature. This option traverses through the hierarchy of a model to highlight real and artificial algebraic loops. A report lists the loops with color codes for easy isolation. This technique allows you to view all the loops in a model simultaneously, so you can remove them quickly and speed up your simulation.

For more information, see Highlight Algebraic Loops in the Model.

Faster Simulations with Accelerated Referenced Models: Run faster consecutive simulations and step back and forth through simulations

Simulating referenced models in Accelerator mode now supports these actions that involve using `SimState`:

- Do a fast restart
- Save complete `SimState` in final state logging
- Restart from saved state

- Step through a simulation

If the referenced model simulated in Accelerator mode contains any of these blocks, then you cannot perform the actions listed above:

- Level 2 MATLAB S-Function
- MATLAB System
- n-D Lookup Table
- S-Function (with custom `SimState` or `PWork` vectors)
- To File
- Simscape blocks

Use SimulationMetadata to retrieve simulation metadata information

You can store and retrieve metadata about a simulation using `SimulationMetadata`. The `SimulationOutput` object of the simulation contains the metadata object. Use `SimulationMetadata` to analyze archived simulations or compare results from multiple simulations. See Simulink.SimulationMetadata for more details.

Simplified conversion of logged data to Dataset format for a common logged data format

You can use the `Simulink.SimulationData.Dataset` constructor to convert data that was logged in one of the following formats to `Dataset` format:

- Array
- Structure
- Structure with time
- MATLAB `timeseries`
- `ModelDataLogs`

Converting data from other Simulink logging formats to `Dataset` format simplifies writing scripts to post-process data logged using different:

- Logging techniques (for example, for models with multiple To Workspace blocks using different data formats or for models that use To Workspace blocks and log signals using `ModelDataLogs` format)

- Simulation modes (for example, Normal and Accelerator)

The conversion to `Dataset` format also makes it easier to take advantage of features that require `Dataset` format. Now you can easily convert data logged in earlier releases that used a format other than `Dataset` to work well with `Dataset` data in a more recent release.

You can use the `Simulink.SimulationData.Dataset.concat` method to combine `Dataset` objects into one concatenated `Dataset` object.

For more information, see Simulink.SimulationData.Dataset.

Default setting for Automatic solver parameter selection

The default setting for **Automatic solver parameter selection** in the **Diagnostics** pane of the Model Configuration Parameters dialog box is now set to `none`. Previously, the default setting for this option was `warning`.

Improved heuristic for step size calculation

Simulink now uses an improved heuristic to calculate the **Max step size** and **Fixed step size** when you set these parameters to `auto` in the **Solver** pane of the Model Configuration Parameters dialog box. The heuristic uses the dynamics of a model in a better way to calculate these parameters. For more information, see Max step size.

Step size details in solver information tooltip

When you set **Max step size** and **Fixed step size** to `auto` in the **Solver** pane of the Model Configuration Parameters dialog box, you can see the numerical values of these settings in the solver information tooltip. Previously, Simulink displayed this information as a warning message. For more information, see Solver Overview.

Solver information in model after simulation

Starting in R2015a, after a model compiles, the right hand corner of the status bar displays the solver used to compile a model. Previously, Simulink reverted to the configuration setting after a simulation completed. For more information, see Solver Overview.

Component-Based Modeling

Consistent Data Support for Testing Components: Load input and log data of a component from buses and all data types

Logging states and root outports now supports full logging capabilities in Normal and Accelerator (including Model Reference Accelerator) simulation modes. State and root outport logging in R2015a:

- Supports all data types, including:

  - Bus data (virtual and nonvirtual buses and arrays of buses)
  - Enumerated data
  - Fixed-point data

- Logs states and outports based on their rates

Simulink supports `Dataset` objects for root inport data import, which eliminates the need to specify multiple inputs using a comma-separated list.

Model Reference Conversion Advisor enhancements

In R2015a, the Model Reference Conversion Advisor make the conversion of a subsystem to a referenced model easier than in previous releases.

- You can have the advisor compare the results of simulating the top model for the referenced model to the results of simulating the baseline model that has the subsystem. In the **Check conversion input parameters** check, select **Check simulation results after conversion**.

  You can specify the following for the comparison of the simulations:

  - Stop time for the simulation
  - Absolute signal tolerance
  - Relative signal tolerance

- In the **Check conversion input parameters** check, you can specify the simulation mode for the Model block that references the referenced model.

- The conversion process stores all of the data it creates (bus objects, signal objects, and tunable parameters) in one file. To specify the file for the advisor to use, in the **Check conversion input parameters** check, use the **Conversion data file name** option.
- The reports for the checks include links that highlight the relevant blocks in the model.
- Restoring the model to the pre-conversion state is faster than in previous releases. Also, the report for the **Complete conversion** check now includes a link to restore the model. That option provides an alternative to selecting **File** > **Load Restore Point** in the advisor.

For details, see Convert a Subsystem to a Referenced Model.

The Simulink.SubSystem.convertToModelReference function supports the Model Reference Conversion Advisor capabilities.

Reduced algebraic loops during model reference simulation

Model reference simulation now addresses some cases that previously resulted in algebraic loops.

Multi-instance support for nonreusable functions in referenced models

Simulation of nonreusable functions in referenced models no longer forces you to set the **Configuration Parameters** > **Model Referencing** > **Total number of instances allowed per top model** parameter to One. You can also set the parameter to Multiple.

Model referencing checks in Model Advisor to reduce warning messages

The Model Advisor includes two new checks to reduce the number of warning messages during simulation and code generation. These checks replace the **Configuration Parameters** > **Diagnostics** > **Model Referencing** > **Model configuration mismatch** diagnostic, which generated many warning messages that you could safely ignore.

- The **Check diagnostic settings ignored during accelerated model reference simulation** check lists models referenced in Accelerator mode for which you set certain runtime diagnostics to a value other than none or Disable all.

  Simulink ignores the diagnostics that the check lists. For more information, see Certain Diagnostic Configuration Parameters Ignored for Models Referenced in Accelerator Mode.

- The **Check code generation identifier formats used for model reference** check lists referenced models with certain parameters whose settings do not contain a $R token (which represents the name of the reference model). Code generation prepends the $R token (if not present) to the identifier format. For more information, see Configuration Parameters Changed During Code Generation.

Compatibility Considerations

The **Model configuration mismatch** diagnostic no longer appears in the **Configuration Parameters** > **Diagnostics** > **Model Referencing** pane. Simulink ignores the setting of the corresponding ModelReferenceCSMismatchMessage parameter.

- When loading a configuration set from a MAT-file or a model, if the parameter setting is either error or warning, the setting is not honored and remains none.
- When you save a configuration set to a MAT-file or save a model to disk, the parameter is not saved.
- When you export a model to a previous version, the parameter is written to disk with a setting of none.
- If you use get_param with this parameter on a configuration set or on a model, the returned value is always the default value (none).
- If you use a set_param command for this parameter on a configuration set or on a model with a setting other than none, Simulink ignores the setting.

Model configuration parameter changes

The following configuration parameters are no longer supported and are not saved in the model configuration:

- CodeGenDirectory (Simulink Coder)
- ConfigAtBuild (Simulink Coder)
- ConfigurationMode (Simulink Coder)
- ConfigurationScript (Simulink Coder)
- CustomRebuildMode (Simulink Coder)
- DataInitializer (Simulink Coder)
- Echo
- EnableOverflowDetection

- `FoldNonRolledExpr`
- `GenerateClassInterface`(Simulink Coder)
- `GenerateCodeInfo` (Simulink Coder)
- `IncludeERTFirstTime` (Simulink Coder)
- `InitialValueSource` (Embedded Coder)
- `MisraCompliance` (Embedded Coder)
- `ModuleName` (Embedded Coder)
- `ModuleNamingRule` (Embedded Coder)
- `ProcessScript` (Simulink Coder)
- `ProcessScriptMode` (Simulink Coder)
- `SimBlas`
- `SimDataInitializer`
- `SimExtrinsic`
- `TargetTypeEmulationWarnSuppressLevel`
- `UseTempVars`

In R2014b, these configuration parameters were command-line only parameters and were not available in the Configuration Parameters dialog box.

Compatibility Considerations

- When you save a configuration set to a MAT-file or save a model to disk, these parameters are NOT saved.
- If you use a `set_param` command with one of these parameters on a configuration set or on a model, the value of the parameter is only changed while the model is loaded in the current MATLAB session.
- If you use a `get_param` command with one of these parameters on a configuration set or on a model, the behavior is unchanged.

Property name change in Simulink.ConfigSetRef

The `WSVarName` property of the `Simulink.ConfigSetRef` object has been renamed to `SourceName`. The `SourceName` property specifies the name of the variable in the workspace or the data dictionary that contains the referenced configuration set. The `WSVarName` will be removed in a future release.

Compatibility Considerations

To avoid future incompatibility, change instances of this property name to the new name.

Flexible structure assignment of buses

When a non-tunable structure is assigned to a bus signal (such as a block which uses a structure for its initial condition parameter), the data types of the fields of the structure no longer need to match the data types of the bus elements. The software now performs an automatic casting of the data type of the structure field so that it matches the data type of the bus signal.

Support for empty subsystems as variant choices

Previously, if you added an empty subsystem with no inputs or outputs as a variant choice inside a Variant Subsystem block, Simulink discarded the empty subsystem. This is because empty variant choices were considered invalid. Moreover, if the Variant Subsystem block contained a valid variant choice and an empty variant choice, no preprocessor conditionals were generated for the valid choice when you built the model.

In R2015a, Simulink considers an empty subsystem as a valid variant choice. If you add an empty variant choice inside a Variant Subsystem block, specify a variant condition for this choice in one of the following ways.

- Specify a variant activation condition for the empty choice. During simulation, if the empty variant choice is active, Simulink ignores the empty choice.
- Comment out the variant activation condition by placing a % symbol before the condition.

Moreover, if the Variant Subsystem block contains a valid variant choice and an empty variant choice, preprocessor conditionals are now generated for the valid choice when you build the model.

For an example, see Define, Configure, and Activate Variant Choices.

Conversion of MATLAB variables used in variant control expressions into `Simulink.Parameter` objects

MATLAB variables allow you to rapidly prototype variant control expressions when you are building your model. Previously, if you wanted to generate preprocessor

conditionals for code generation, you had to manually convert these variables into `Simulink.Parameter` objects.

In R2015a, use the function `Simulink.VariantManager.findVariantControlVars` to find and convert MATLAB variables used in variant control expressions into `Simulink.Parameter` objects. For an example, see Convert Variant Control Variables into Simulink.Parameter Objects.

Project and File Management

Simulink Project Sharing: Share a project using GitHub, email, or a MATLAB toolbox

Release 2015a provides new options for sharing Simulink projects:

- Make your project publicly available on GitHub®.
- Share your project via email.
- Package your project as a MATLAB toolbox.
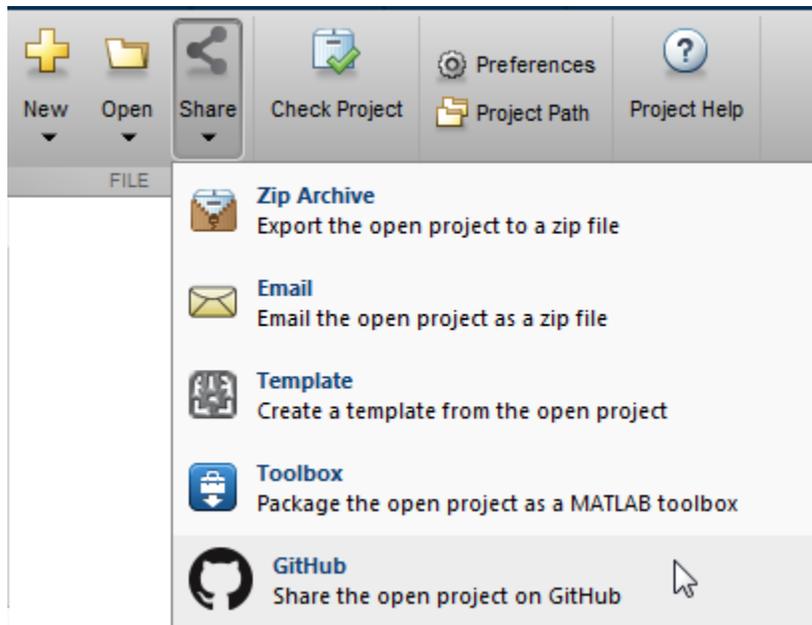
For details, see Sharing Simulink Projects.

Interactively manage the MATLAB search path for your project

You can interactively add or remove folders from the project path. With Simulink Project, opening your project adds the project path to the MATLAB search path. Closing your

project removes the project path from the MATLAB search path. For more information, see Specify Project Path.

Easy viewing and editing of project labels

R2015a provides easier management of label data through the improved file details view.
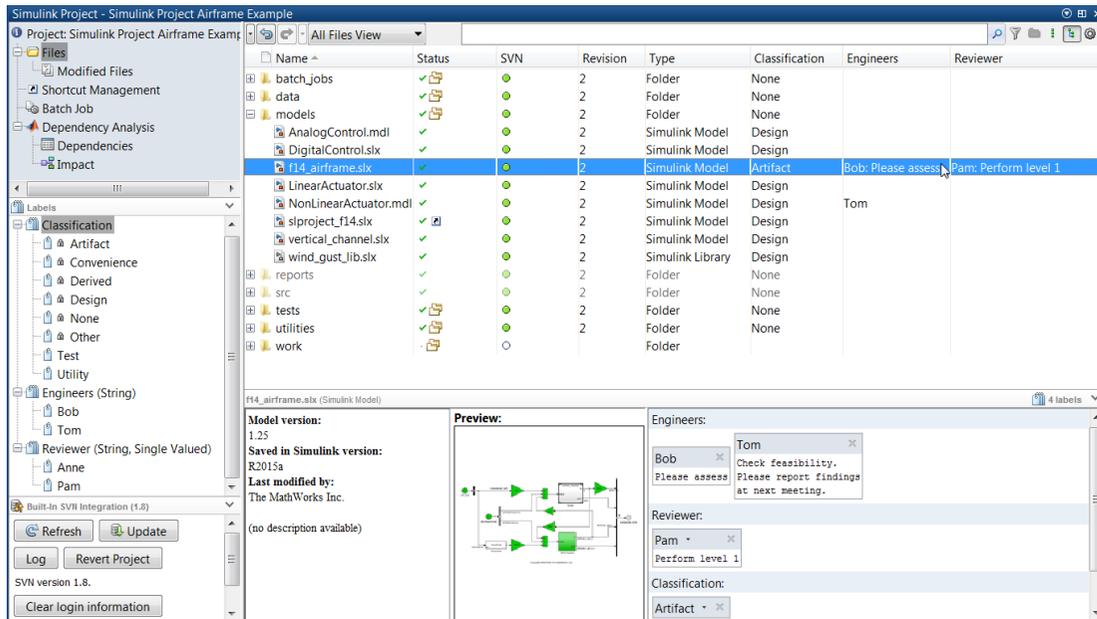
For more information, see:

- Create Labels
- Add Labels to Files
- View and Edit Label Data

Changed file lists and branch deletion in Git Manage Branches dialog box

In R2015a, when using Git source control, you can view branch details and delete branches in the Manage Branches dialog box. You can see which files changed for a particular commit in the branch viewer, and view the author, date, and commit message.
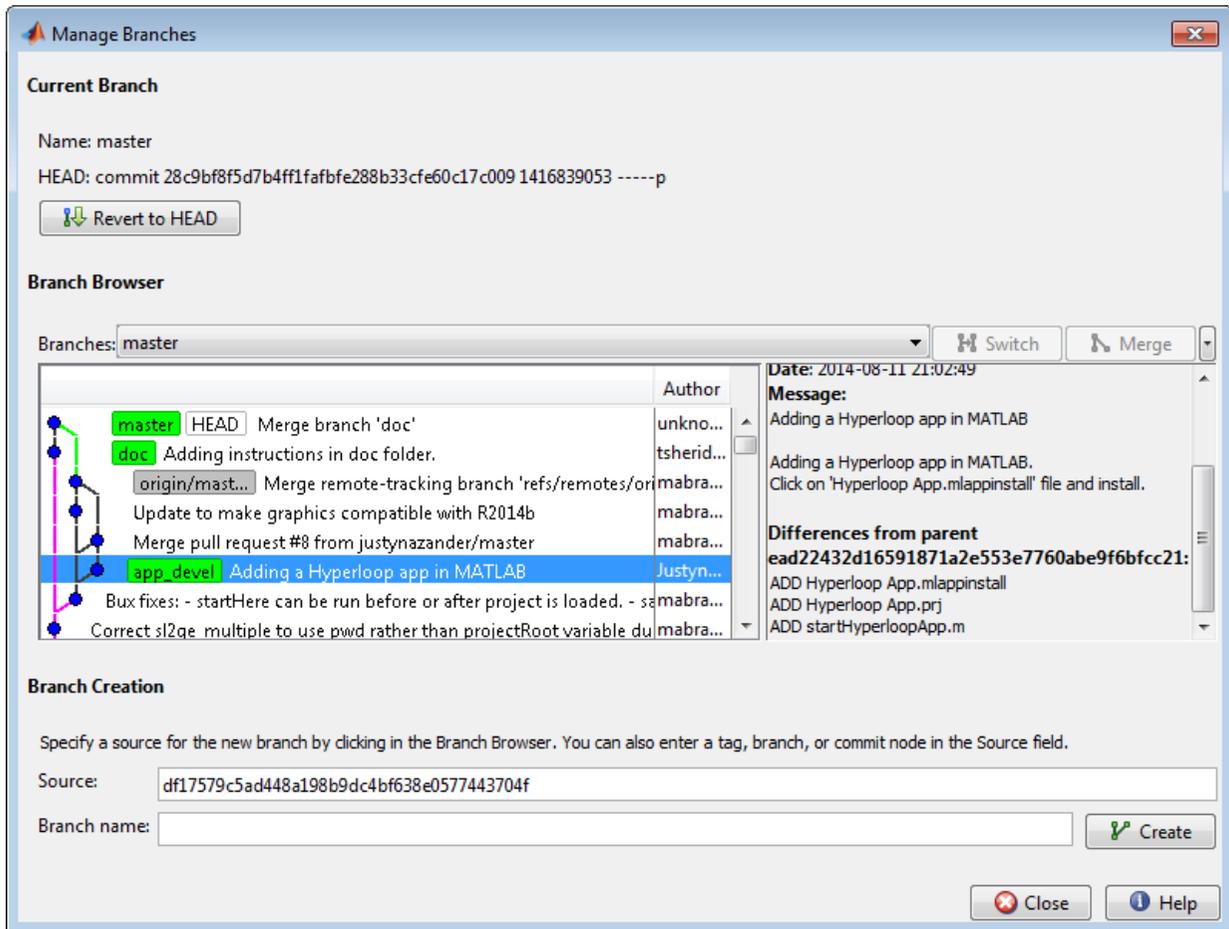
For details, see Branch and Merge Files with Git.

New preferences to control loading and saving models

Simulink has a new preference, **Do not load models that are shadowed on the MATLAB path**. Use the preference to specify whether to load a model that is shadowed by another file of the same name higher on the MATLAB path. If you turn the preference on by selecting the check box, then Simulink displays an error when you try to load a shadowed model. For details, see Do not load models that are shadowed on the MATLAB path.

Another preference, **Do not load models created with a newer version of Simulink**, is now on by default. These preferences help you avoid accidentally loading or editing the wrong model or library.

You can use a new preference, **Save a thumbnail image inside SLX files** to control whether to save a small screen shot image of the model. You can view the screenshot for a selected model in the Current Folder browser preview pane. If your model is very large and you want to reduce the time taken to save the model, then you can turn this preference off to avoid saving thumbnail model images.

Compatibility Considerations

The preference **Do not load models created with a newer version of Simulink** is now on by default. Simulink does not the load the model and displays an error message in the Command Window. In previous releases, the preference was off by default, so Simulink loaded models last saved in a newer version and displayed a warning message.

Improved error reporting from get_param, set_param and save_system

The functions `get_param`, `set_param` and `save_system` have improved error reporting to help you correct problems specifying the correct model or block. The new messages tell you if the model is not loaded, the model name is not valid, or the block is not found in the specified model.

Model dependency analysis option to find enumeration definition files

In R2015a, model dependency analysis provides an option to search for enumeration definition files. Turn on this option to detect enumerated data types used as part of a bus object definition. You can then include the files when exporting the model and required files into a zip file.

The option is not on by default because it requires a model update. You cannot perform this search with project impact analysis, because the search requires model updates. Instead, select **Analysis > Model Dependencies > Generate Manifest**, and turn on the **Find enumeration definition files (performs a Model Update)** option.

For details, see Generate Manifests.

Export to previous version supports seven years

In R2015a, in the Export to Previous Version dialog box, the **Save as type** list includes seven years of previous releases. In future releases, this list will continue to provide seven years of previous releases.

In previous releases, the list included versions back to R14 (2004). If you want to export to older versions than seven years, you can use the `save_system` function instead. For details, see save_system.

Data Management

Data Dictionary API: Automate the creation and editing of data dictionaries with MATLAB scripts

You can now manage data dictionaries and interact with dictionary content at the MATLAB command prompt in addition to the Model Explorer. For example, you can:

- Migrate models to use data dictionaries
- Import data from and export data to external files and the MATLAB base workspace
- Create, delete, and reassign entries
- Save and discard changes to entire dictionaries or discard changes to individual entries
- Search for specific entries
- Compose reference dictionary hierarchies

See Store Data in Dictionary Programmatically for examples and a list of relevant functions and classes.

Rename All: Change the name of a parameter and all its references

You can now use Model Explorer to rename a variable everywhere it is used by blocks in a Simulink model.

See Rename Variables for an example.

MATLAB Editor features for editing model workspace code

When you view the dialog box for a model workspace, for example using the property dialog pane of Model Explorer, and set **Data source** to `MATLAB Code`, the code editing area now behaves like the MATLAB Editor. For example, the editing area applies syntax highlighting to your code.

Management of variables from block dialog box fields

You can now navigate to and edit variables by right-clicking expressions that you specify in block dialog boxes. For each variable in an expression, you can navigate to the workspace that defines the variable or open a separate dialog box to edit the variable.

You can also create variables by right-clicking an expression that contains the names of potential variables. For each name you specify in the expression, you can create a variable in a workspace that is appropriate for the model.

See Manage Variables from Block Parameter for more information.

Other Data section added to data dictionary

To store reference data that are not used by a model for simulation, but that are still relevant to the model, use the Other Data section of a data dictionary. For example, if a model simulates the behavior of mechanical equipment, you can store the manufacturer specifications in the Other Data section.

You can also store objects of any MATLAB or Simulink class, including custom classes, in the Other Data section of a dictionary. By contrast, the Design Data section stores only objects relevant to simulation of a model.

Data dictionaries now have a Design Data section, a Configurations section, and an Other Data section. Prior to R2015a, the Design Data section was named Global Design Data.

Model-wide renaming of data stores

You can now rename a data store everywhere it is used by Data Store Read and Data Store Write blocks in a Simulink model. Previously, you manually updated all Data Store Read and Data Store Write blocks with the new name of the data store.

See Rename Data Stores for more information.

Reporting of enumerated types used by model

You can now use the existing function `Simulink.findVars` to discover the enumerated data types that are needed by a model. `Simulink.findVars` can report the names of enumerated types that are used to define model variables.

You can enable the reporting of enumerated types using the name-value pair `IncludeEnumTypes`. With this name-value pair enabled, `Simulink.findVars` returns a `Simulink.VariableUsage` object for each enumerated type in addition to the `Simulink.VariableUsage` objects returned for model variables.

Root Inport Mapping tool updates

The Root Inport Mapping tool now supports:

- Models configured for Fast Restart.
- Importing and mapping of Simulink Design Verifier test vectors. For more information, see Import Test Vectors from Simulink Design Verifier Environment.

Connection to Educational Hardware

Simulink Support Package for Apple iOS Devices: Create an App that runs Simulink models and algorithms on your Apple iOS device

You can run Simulink models on Apple iOS devices. You can also tune parameter values in the model, and receive data from the model, while it is running on these devices.

Use the **Simulink Support Package for Apple iOS Devices** block library to access the Apple iOS hardware:

- Accelerometer
- Audio Capture
- Audio Playback
- Camera
- Display
- FromApp
- Gyroscope
- Location Sensor
- ToApp
- UDP Receive
- UDP Send

To install or update this support package, perform the steps described in Install Support for Apple iOS Devices.

MathWorks response to the Shellshock vulnerability

The support package for BeagleBoard (v14.2) contains a vulnerable version of the Bash shell. The default configuration of this support package does not expose this vulnerability to a network connection. We recommend running this device inside of a trusted network or behind a firewall. We will update this package as updated software is available.

For more information, see http://mathworks.com/matlabcentral/answers/158586-what-is-mathworks-response-to-the-shellshock-vulnerability.

Removed support for Gumstix Overo and PandaBoard hardware

The following support packages have been removed and are no longer available:

- Simulink Support Package for Gumstix® Overo® Hardware
- Simulink Support Package for PandaBoard Hardware

Signal Management

Array of buses with Unit Delay block

You can use an array of buses as an input signal to a Unit Delay block.

Block Enhancements

Resettable Subsystem block to reset the subsystem states

The Resettable Subsystem block is a new block in the Ports & Subsystems library. Use this subsystem to reset the states of all blocks inside the subsystem on triggering. For more information, see Resettable Subsystem.

Conditional display of the Sample Time parameter

The Sample Time parameter in the dialog box of certain Simulink blocks is now hidden by default. If you set the sample time to a value other the default (at the command line or if it is set that way in an existing model), then the parameter is visible. For more information, see Blocks for Which Sample Time Is Not Recommended.

Inheritance of frame-based input returns error

Note: This release note applies only if you have installed DSP System Toolbox.

As part of general product-wide changes pertaining to frame-based processing, certain block options that use the frame attribute of the input signal now cause an error in Simulink. For more information on changes in the DSP System Toolbox, see Frame-based processing.

The following sections provide more detailed information about the specific R2015a Simulink software changes for frame-based processing:

Input Processing Parameter Set to Inherited

Setting `Input processing` parameter to `Inherited` now errors for these blocks:

- Relay
- Backlash
- Difference
- Delay
- Unit Delay
- Variable Integer Delay
- Discrete Derivative
- Tapped Delay
- Transfer Fcn Real Zero
- Detect Increase
- Detect Decrease
- Detect Change
- Detect Rise Positive
- Detect Rise Nonnegative
- Detect Fall Negative
- Detect Fall Nonpositive

Compatibility Considerations

To ensure consistent results for models created in previous releases, set **Input processing** to:

- `Columns as channels (frame based)`, for frame-based input signals (double-line)
- `Elements as channels (sample based)`, for sample-based signal input signals (single-line)

If you are not sure of which option to choose, select and run the Simulink Upgrade Advisor checks:

- `Check model for block upgrade issues requiring compile time information`

- `Check model for custom library blocks that rely on frame status of the signal`, for blocks in a custom library

Inherited Setting on Save 2-D Signals

In the To Workspace block, setting the **Save Format** parameter to `Structure` or `Array` and the **Save 2-D signals as** parameter to `Inherit from input` now causes an error.

Compatibility Considerations

To ensure consistent results for models created in previous releases, set **Save 2-D signals as** to

- **3-D array (concatenate along third dimension)**, for sample-based input signals
- **2-D array (concatenate along first dimension)**, for frame-based input signals

For models created in R2015a:

- For frame-based processing, set **Save 2-D signals as** to **2-D array (concatenate along first dimension)**.
- For sample-based processing, set **Save 2-D signals as** to **3-D array (concatenate along third dimension)**.

If you are not sure of which option to choose, run these Simulink Upgrade Advisor checks:

- `Check model for block upgrade issues requiring compile time information`
- `Check model for custom library blocks that rely on frame status of the signal`, for blocks in a custom library

Frame-based Inputs Removed for Bias Block

Frame-based input support is removed from the Bias block.

Compatibility Considerations

To ensure consistent results for models created in older releases,

1   Change the block input to sample based by inserting a Frame Conversion block with **Sampling mode of output signal** set to `Sample-based`.

2   Insert a Frame Conversion block at the output of the block with **Sampling mode** set to `Frame based`.

3   Set the bias parameter of the block to `repmat(b, N, 1)`, where `b` is the value of the bias, and `N` is the input frame length.

Frame-based Inputs Removed for Tapped Delay Block

Frame-based input support is removed from the Tapped Delay block.

Compatibility Considerations

To ensure consistent results for models created in older releases, replace the Tapped Delay block by a Unit Delay block with **Input Processing** set to `Columns as channels (frame based)`.

Frame-based Input Removed for Transfer Fcn First Order Block

Frame-based signal support is removed from the Transfer Fcn First Order block.

Compatibility Considerations

To ensure consistent results for models created in older releases:

1   Replace the Transfer Fcn First Order block with a Discrete Transfer Fcn block.

2   In the Discrete Transfer Fcn block, set **Input Processing** to `Columns as channels (frame based)`.

3   Set **Numerator**, **Denominator** and **Initial Condition** of the new block to `[1-p 0]`, `[1 -p]` and `ic/(1-p)`, respectively, where `p` is the value of the pole and `ic` is the value of the initial condition on the Transfer Fcn First Order block.

Frame-based Input Removed for Transfer Fcn Lead or Lag Block

Frame-based input support is removed from the Transfer Fcn Lead or Lag block.

Compatibility Considerations

To ensure consistent results for models created in older releases:

1 Replace the Transfer Fcn Lead or Lag block by a Discrete Filter block.

2 In the Discrete Filter block, set **Input Processing** to `Columns as channels (frame based)` and **Filter Structure** to `Direct form I`.

3 Set **Numerator** and **Denominator** of the new block to `[1 -z]` and `[1 -p]`, respectively, where `p` is the value of the pole and `z` is the value of the zero on the Transfer Fcn Lead or Lag block.

Sampling Mode Set to Frame-based

Setting **Sampling Mode** parameter to `Frame based` now errors for these blocks:

- Inport
- Signal Specification
- Outport

Compatibility Considerations

To ensure consistent results for models created in older releases, set **Sampling Mode** to `Sample based` or `Auto` instead.

If you are not sure of which option to choose, select and run the Simulink Upgrade Advisor checks:

- `Check model for block upgrade issues requiring compile time information`
- `Check model for custom library blocks that rely on frame status of the signal`, for blocks in a custom library

Scope block to Time Scope Block conversion

You can convert a Scope block to a Time Scope block to try the new block. On the Scope toolbar, click the Try Time Scope button . The Scope block converts to a Time Scope block. For Floating Scopes, this button is disabled (grayed out).

The Time Scope block includes simulation controls (run, forward, backward), additional support for signals (sample-based, frame-based), and debugging tools such as Cursors and Triggers. See Migrate Scope To Time Scope.

Option to provide PID gains as external inputs to PID Controller and PID controller (2DOF) blocks

A new option in the PID Controller and PID Controller (2DOF) blocks adds signal inputs for the PID gains and filter coefficients. Previously the PID parameters had to be entered in the block dialog box as numerical values or MATLAB expressions. Enabling external inputs for the parameters allows you to compute PID gains and filter coefficients externally to the block and provide them to the block as signal inputs. External gain input is useful, for example, to implement gain-scheduled PID control, in which controller gains are determined by logic or other calculation in the Simulink model and passed to the block. To enable external inputs for the PID coefficients, in the block dialog box, in the Controller parameters section, in the **Source** menu, select **external**.
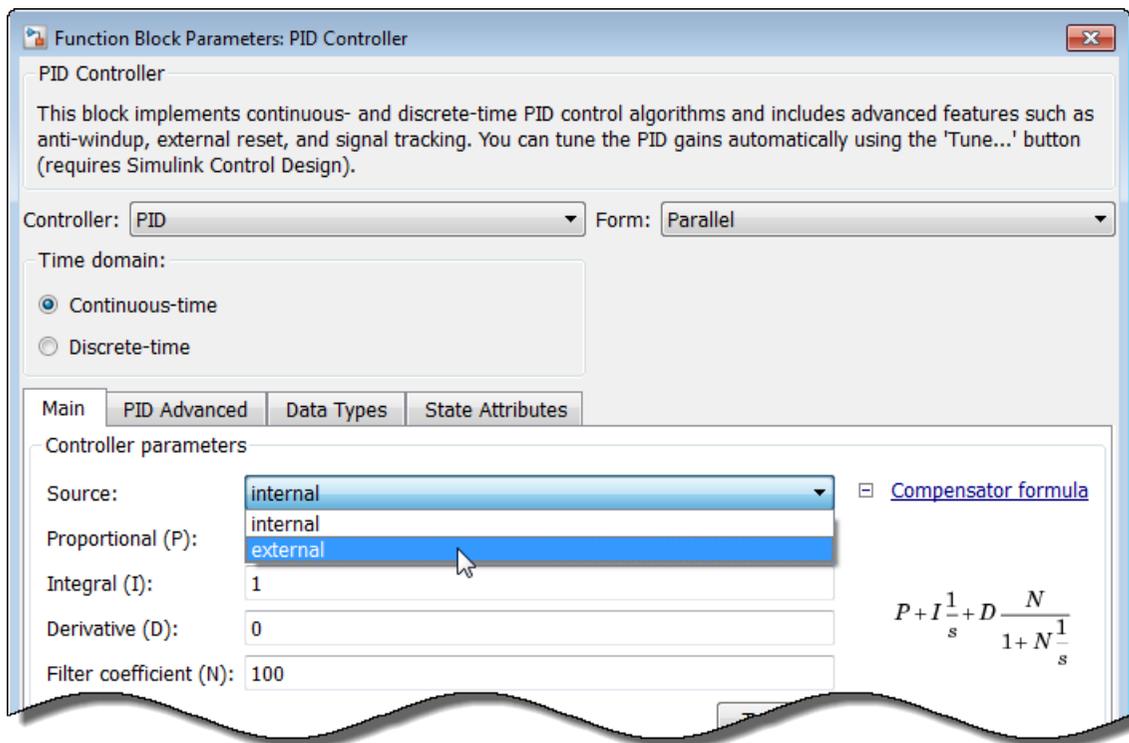
When you click **OK** or **Apply**, the new inputs appear on the block in the Simulink model.
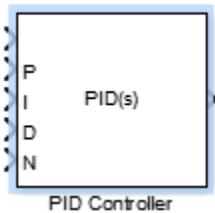
PID Controller

For more information about using the PID controller blocks, see the PID Controller and PID Controller (2 DOF) block reference pages.

Improvements for creating System objects

The following improvements have been made to creating your own System objects:

- Number of allowable code generation inputs increased to 32
- `isInputSizeLockedImpl` method for specifying whether the input port dimensions are locked
- `matlab.system.display.Action` class, used in the `getPropertyGroupsImpl` method, to define a MATLAB System block button that can call a System object method
- `getSimulateUsingImpl` and `showSimulateUsingImpl` methods to set the value of the `SimulateUsing` parameter and specify whether to show the `SimulateUsing` parameter in the MATLAB System block

MATLAB System block support for model coverage analysis

The Simulink Verification and Validation software now supports model coverage analysis for the MATLAB System block with the **Simulate using** parameter set to `Code generation`. For more information, see MATLAB System Block Limitations.

Enable port on the Delay block

The Delay block now provides an enable port to control its execution at every time step. You can show this port using the **Show enable port** parameter in the block dialog box. For more information, see the Delay block reference page.

MATLAB Function Blocks

More efficient generated code for logical indexing

Code generated for logical array indexing is faster and uses less memory than in previous releases. For example, the generated code for the following function is more efficient than in previous releases.

```
function x = foo(x,N)
assert(all(size(x) == [1 100]))
x(x>N) = N;
```

In R2015a, you do not have to replace $x(x>N) = N$ with a `for`-loop to improve performance.

Faster compile time for large functions and models due to decreased constant folding limit

When possible, the code generation software replaces an expression with the result of the expression evaluation. This practice is called constant folding. In R2015a, the maximum number of instructions that the code generation software constant folds is lower than in previous releases. The lower constant folding limit can reduce compile times for large functions and models.

Compatibility Considerations

For large functions or models, it is possible that some expressions that were constant-folded in previous releases are not constant-folded in R2015a. In these cases, the generated code contains the expressions rather than the results of the evaluated expressions.

JIT compilation technology to reduce model update time

MATLAB Function block uses just-in-time (JIT) compilation technology to improve model update of many MATLAB Function blocks. For these blocks, Simulink does not generate C code or a MEX-file to simulate the block. Simulink applies JIT mode to MATLAB Function blocks that qualify. You do not have to enable it.

When a MATLAB Function block uses JIT mode, debugging is disabled. To debug, set a breakpoint in the MATLAB Function block before simulation. Simulink enables debugging, and does not use JIT mode.

Compatibility Considerations

By default, the software uses JIT mode on MATLAB Function blocks to speed up compilation time. When a block uses JIT mode, debugging is disabled. During simulation, you cannot set a breakpoint. If you set a breakpoint before simulation begins, the software enables debugging. You no longer directly enable or disable debugging with **Enable debugging/animation** on the **Simulation Target** pane of the Configuration Parameters dialog box or menu option.

In previous releases, if you set the command-line parameter `SFSIMEnableDebug`, the software enabled debugging for the model. Now, setting this parameter prevents the block from using JIT mode. Do not set this parameter if you want to improve model update performance using JIT mode.

Some MATLAB Function blocks do not qualify for JIT mode, such as blocks that integrate custom C code. In these cases, the software defaults to MEX-file generation with debugging enabled. For optimal simulation performance for these blocks, turn off debugging by using this command.

```
sfc('coder_options', 'forceDebugOff', 1);
```

After you run this command, these blocks do not have debugging or run-time error checking.

Code generation for casts to and from types of variables declared using `coder.opaque`

For code generation, you can use the MATLAB `cast` function to cast a variable to or from a variable that is declared using `coder.opaque`. Use `cast` with `coder.opaque` only for numeric types.

To cast a variable declared by `coder.opaque` to a MATLAB type, you can use the `B = cast(A,type)` syntax. For example:

```
x = coder.opaque('size_t','0');
x1 = cast(x, 'int32');
```

You can also use the `B = cast(A,'like',p)` syntax. For example:

```
x = coder.opaque('size_t','0');
x1 = cast(x, 'like', int32(0));
```

To cast a MATLAB variable to the type of a variable declared by `coder.opaque`, you must use the `B = cast(A,'like',p)` syntax. For example:

```
x = int32(12);
x1 = coder.opaque('size_t', '0');
x2 = cast(x, 'like', x1));
```

Use `cast` with `coder.opaque` to generate the correct data types for:

- Inputs to C/C++ functions that you call using `coder.ceval`.
- Variables that you assign to outputs from C/C++ functions that you call using `coder.ceval`.

Without this casting, it is possible to receive compiler warnings during code generation.

Consider this MATLAB code:

```
yt = coder.opaque('size_t', '42');
yt = coder.ceval('foo');
y = cast(yt, 'int32');
```

- `coder.opaque` declares that `yt` has C type `size_t`.
- `y = cast(yt, 'int32')` converts `yt` to `int32` and assigns the result to `y`.

Because `y` is a MATLAB numeric type, you can use `y` as you would normally use a variable in your MATLAB code.

The generated code looks like:

```
size_t yt= 42;
int32_T y;
y = (int32_T)yt;
```

It is possible that the explicit cast in the generated code prevents a compiler warning.

Improved recognition of compile-time constants

In previous releases, the code generation software recognized that structure fields or array elements were constant only when all fields or elements were constant. In R2015a,

in some cases, the software can recognize constant fields or constant elements even when some structure fields or array elements are not constant.

For example, consider the following code. Field `s.a` is constant and field `s.b` is not constant:

```
function y = create_array(x)
s.a = 10;
s.b = x;
y = zeros(1, s.a);
```

In previous releases, the software did not recognize that field `s.a` was constant. In the generated code, if variable-sizing was enabled, `y` was a variable-size array. If variable-sizing was disabled, the code generation software reported an error. In R2015a, the software recognizes that `s.a` is a constant. `y` is a static row vector with 10 elements.

As a result of this improvement, you can use individual assignments to assign constant values to structure fields. For example:

```
function y = mystruct(x)
s.a = 3;
s.b = 4;
y = zeros(s.a,s.b);
```

In previous releases, the software recognized the constants only if you defined the complete structure using the `struct` function: For example:

```
function y = mystruct(x)
s = struct('a', 3, 'b', 4);
y = zeros(s.a,s.b);
```

In some cases, the code generation software cannot recognize constant structure fields or array elements. See Code Generation for Constants in Structures and Arrays.

Compatibility Considerations

The improved recognition of constant fields and elements can cause the following differences between code generated in R2015a and code generated in previous releases:

- A function output can be more specific in R2015a than it was in previous releases. An output that was complex in previous releases can be real in R2015a. An array output that was variable-size in previous releases can be fixed-size in R2015a.

- Some branches of code that are present in code generated using previous releases are eliminated from the generated code in R2015a.

Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions

Image Processing Toolbox

- bweuler
- bwlabel
- bwperim
- regionprops
- watershed

See Image Processing Toolbox.

Computer Vision System Toolbox

- opticalFlow
- vision.DeployableVideoPlayer on Mac platform.

  In previous releases, `vision.DeployableVideoPlayer` supported code generation on Linux® and Windows platforms. In R2015a, `vision.DeployableVideoPlayer` also supports code generation on a Mac platform.

See Computer Vision System Toolbox.

Code generation for additional Communications System Toolbox, DSP System Toolbox, and Phased Array System Toolbox System objects

Communications System Toolbox

- comm.CarrierSynchronizer
- comm.FMBroadcastDemodulator
- comm.FMBroadcastModulator
- comm.FMDemodulator
- comm.FMModulator

- comm.SymbolSynchronizer

See Communications System Toolbox.

DSP System Toolbox

- iirparameq
- dsp.LowpassFilter
- dsp.HighpassFilter

See DSP System Toolbox.

Phased Array System Toolbox

- pilotcalib
- phased.UCA
- phased.MFSKWaveform

See Phased Array System Toolbox.

Code generation for additional Statistics and Machine Learning Toolbox functions

- betafit
- betalike
- pca
- pearsrnd

See Statistics and Machine Learning Toolbox.

Code generation for additional MATLAB functions

Linear Algebra

- bandwidth
- isbanded
- isdiag

- istril
- istriu
- lsqnonneg

See Linear Algebra in MATLAB.

Statistics in MATLAB

- cummin
- cummax

See Statistics in MATLAB.

Code generation for additional MATLAB function options

- `dimension` option for cumsum and cumprod

See Functions and Objects Supported for C and C++ Code Generation — Alphabetical List.

Model Advisor

Multiple instances of advisors

Model Advisor infrastructure changes allow you to save analysis time. To prepare for advisor analysis, you can open and configure:

- Different advisors on the same model
- The same advisor on different models

However, you can run the analysis for only one advisor at time.

Advisors	More information
Model Advisor	Consulting the Model Advisor
Performance Advisor	How Performance Advisor Improves Simulation Performance
Upgrade Advisor	Consult the Upgrade Advisor
Model Reference Conversion Advisor	Convert a Subsystem to a Referenced Model
Code Generation Advisor – Available with Simulink Coder	High-Level Code Generation Objectives
Fixed-Point Advisor – Available with Fixed-Point Designer	Fixed-Point Advisor
HDL Workflow Advisor – Available with HDL Coder	HDL Workflow Advisor
Simulink Code Inspector™ Compatibility Checker – Available with Simulink Code Inspector	Model Compatibility

Improved advisor startup performance

Model Advisor infrastructure changes have improved secondary startup performance of these advisors.

Advisors	More information
Model Advisor	Consulting the Model Advisor

Advisors	More information
Performance Advisor	How Performance Advisor Improves Simulation Performance
Upgrade Advisor	Consult the Upgrade Advisor
Model Reference Conversion Advisor	Convert a Subsystem to a Referenced Model
Code Generation Advisor – Available with Simulink Coder	High-Level Code Generation Objectives
Fixed-Point Advisor – Available with Fixed-Point Designer	Fixed-Point Advisor
HDL Workflow Advisor – Available with HDL Coder	HDL Workflow Advisor
Simulink Code Inspector Compatibility Checker – Available with Simulink Code Inspector	Model Compatibility

Model Advisor check input parameters retained for each instance of check

For Model Advisor checks with input parameters, instances of the check retain the input parameters that you enter. When you run the check, the results reflect the input parameter for that instance of the check. Previously, when you entered an input parameter for a check, all instances of the check were updated with the value you entered.

Model referencing checks in Model Advisor to reduce warning messages

The Model Advisor includes two new checks to reduce the number of warning messages during simulation and code generation. These checks replace the **Configuration Parameters** > **Diagnostics** > **Model Referencing** > **Model configuration mismatch** diagnostic, which generated many warning messages that you could safely ignore. For details, see "Model referencing checks in Model Advisor to reduce warning messages" on page 6-10 in the "Component-Based Modeling" section of these release notes.

Compatibility Considerations

The **Model configuration mismatch** diagnostic no longer appears in the **Configuration Parameters** > **Diagnostics** > **Model Referencing** pane. Simulink ignores the setting of the corresponding `ModelReferenceCSMismatchMessage` parameter. For details, see "Model referencing checks in Model Advisor to reduce warning messages" on page 6-10 in the "Component-Based Modeling" section of these release notes.

R2014b

Version: 8.4

New Features

Bug Fixes

Compatibility Considerations

Simulink Editor

Smart Editing Cues: Accelerate model building with just-in-time contextual prompts

The Simulink Editor provides several new model editing options that:

- Allow you to edit a model from within the diagram, without opening separate dialog boxes from a menu
- Provide editing options based on the context of your most recent editing operations

Add and Configure a Block without Leaving a Diagram

You can add a block to a model by typing a block name in a new Simulink Editor quick insert interface. You can perform a quick insert in several ways. For example, to add a Gain block and set the **Gain** parameter to 3, here is one approach:

1   Left-click in an empty space in the diagram, near where you want to add the Gain block.

2   Type "ga".

    A list of blocks beginning with "ga" appears.

3   The Gain block appears first in the list, so just press **Enter**.

    A Gain block appears in the diagram.

4   In the hot parameter edit box, enter a gain of 3.

The hot parameter edit box allows you to enter a value for one parameter per block that you add using the quick insert feature.

For more information, see Add Blocks Using Quick Insert.

Insert a Complementary Block

You can insert a complementary block for the following blocks from within the canvas.

- GoTo and From
- Data Store Read and Data Store Write

For example, to add a From block associated with GoTo block in a model, hover over the GoTo block

1 Hover over the GoTo block.
2 Click and drag the blue tear-off guide.

Perform Actions on a Marquee Selection

You can perform the following actions on a marquee selection (multiple selected objects in an area of a model), without leaving the canvas.

- Create a subsystem
- Create a triggered, enabled, or function-call subsystem
- Comment or uncomment the selected blocks

To select an action, hover over the blue action button. For example, to create a subsystem from the selected blocks, click the **Create Subsystem** button.
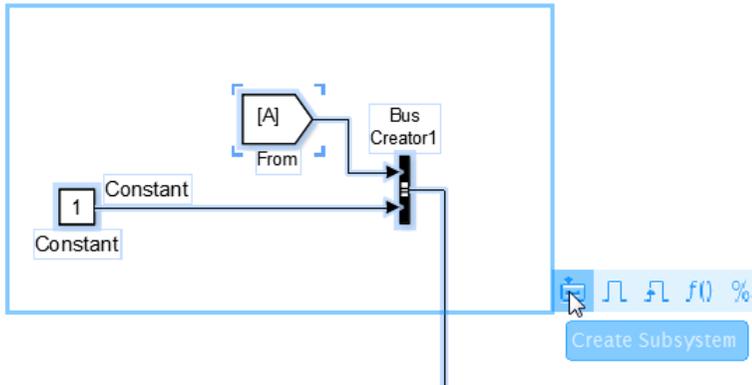
Connect Aligned Blocks Using a Guide

When you align the ports of two blocks, blue guide lines appear. Release the mouse button. Some blue guide lines remain. You can click a remaining blue guide line to connect blocks.

Viewmarks: Save graphical views of a model for quick access to areas of interest

You can create and manage viewmarks, which are bookmarks to parts of a model. Use viewmarks to capture graphical views of a model or parts of a model. You can capture viewmarks for specific levels in a model hierarchy. You can also pan and zoom in a displayed system, to capture the specific portion of interest.

Some examples of ways you can use viewmarks include:

- Navigate to different levels of complex models without opening multiple Simulink Editor tabs or windows.
- Review model designs
- Visually compare different versions of a model.

You can manage your viewmarks for all models in a single gallery of viewmarks, organized by model.

For more information, see Use Viewmarks to Save Views of Models.

Annotation Connectors: Associate annotations with blocks in models

You can add blue connector lines between an annotation and a block. The connector is similar to a callout, identifying the block that an annotation applies to. As you move the annotation or block to which the connector attaches, Simulink redraws the connector. For more information, see Add Lines to Connect Annotations to Blocks.

When you create an annotation, by default the annotation appears in the model. You can now configure an annotation so that you can choose to hide that annotation. This allows you to include annotations that provide additional information about a model without cluttering the model.

To configure an annotation so that you can hide it:

1   Right-click the annotation.
2   In the context menu, select **Convert to Markup**.

By default, all annotations appear in the model. To hide annotations that are converted to markup, select **Display** > **Hide Markup**.

R2014b includes two new `set_param` and `get_param` parameters relating to showing and hiding annotations:

- `MarkupType` — specifies whether an annotation can be hidden (`'markup'`) or always appears (`'model'`)
- `ShowMarkupType` — Specifies whether to display markup annotations (`'on'`) or hide them (`'off'`).

For more information, see Show or Hide Annotations.

Edit bar for quick annotation formatting

When you click in an annotation, the Simulink Editor now displays an annotation edit bar. Click an edit bar button to format an annotation without having to open a menu. Types of formatting you can do include:

- Make text bold or italic
- Enlarge or shrink the font size
- Center text

Annotation table column and row resizing

In an annotation, you can interactively resize table columns or rows by dragging a table column or row border.

Reuse of annotation text formatting

You can copy the formatting from one piece of annotation text to other text in the same annotation, using **Format Painter** button in the annotation edit bar. For details, see Copy Formatting.

Annotation layering

You can specify whether an annotation appears in front of or behind other annotations. (Annotations always appear behind blocks.)

To display an annotation in front of another annotation, use the following steps.

1 Do a marquee selection (bounding box) that includes the annotation that you want to appear in front.
2 Select the **Diagram** > **Arrange** > **Front** menu option.

Access Diagnostic Viewer from status bar

When you update, simulate, or build a model, the status bar at the bottom of the Simulink Editor displays the total number of diagnostics generated. If these diagnostics are in the form of warnings or information, Simulink does not bring the Diagnostic Viewer into focus so that you can continue developing your model. When you are ready to diagnose warnings or view information, click the link displayed in the status bar to bring the Diagnostic Viewer into focus.

Printing to file

When you use the Print Model dialog box and select **Print To File**, the default file format is now PDF instead of the previous default of Postscript. Exporting to PDF generally makes it easier to share models with other people.

The default file name is now the name of the system you exported, which makes it easier to tell which system is in a file. Previously, Simulink generated a file name that did not reflect the system name.

Simulation Analysis and Performance

Fast Restart: Run consecutive simulations more quickly

Fast Restart is a new simulation workflow that eliminates the need for compiling a model repeatedly in iterative simulations. In this mode, the model compiles only in the first iteration. Simulink uses this compile information for successive runs, so there is no need to recompile. You can tune parameters or root-level inputs between runs as long as there are no structural changes to the model. This saves you time spent on recompiling and improves overall simulation efficiency.

Use Fast Restart to tune parameters in a model iteratively, calibrate a system for a desired response, or run multiple simulations in which the compile time is comparable to simulation time. For more information, see How Fast Restart Improves Iterative Simulations.

New Simulation Data Inspector: View live signal data and access visualization options such as data cursors

You can now view a signal in the Simulation Data Inspector during model simulation. This is especially helpful for workflows that involve debugging and optimizing a model. For more information, see Stream Data to the Simulation Data Inspector.

The Simulation Data Inspector now supports Simscape simulation output and includes new visualization options, such as data cursors, which help you inspect signal values in the Simulation Data Inspector plot.

Fixed Point Support for Conditional Breakpoints

Signals of fixed point data type now support conditional breakpoints based on the converted double value.

Quick Scan simulation in Performance Advisor for faster diagnosis

The Quick Scan feature in Performance Advisor helps you analyze a model quickly to deliver an approximate analysis of suboptimal conditions or settings in the model. Quick Scan does not require any preset conditions, and you can run it anytime. Using this

feature, you can get a preview of potential performance improvement changes in a model without performing baseline measurement, multiple compilations, or simulations. See Perform a Quick Scan Diagnosis for more information.

Removal of warning when variable-step solver is selected for discrete models

When you simulate a discrete model with a variable-step solver, Simulink automatically switches the solver selection from `Variable-step continuous` (default) to `Variable-step discrete`. When doing so, Simulink no longer displays a warning about the switch, even if you have set **Diagnostics** > **Automatic solver parameter selection** to `warning` in the **Model Configuration Parameters** dialog box.

Block callbacks not evaluated in Rapid Accelerator mode with up-to-date check off

Previously, when you simulated a model in Rapid Accelerator mode with the `RapidAcceleratorUpToDateCheck` parameter set to `off`, Simulink evaluated the start and stop callbacks of the model and the blocks. Starting in R2014b, when the `RapidAcceleratorUpToDateCheck` parameter is set to `off`, Simulink evaluates only the model start and stop callbacks.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
slupdate	Warns	Use the Upgrade Advisor instead.	slupdate will be removed in a future release. The slupdate command can only upgrade some parts of your model. Use the Upgrade Advisor instead. See Model Upgrades.

Improvements to Scope blocks and Scope viewers

Enhancements to viewing signals.

Scope blocks:

- Allow scrolling of signals to the left
- Supports fast restart

Floating Scope blocks:

- New **Log/Unlog Viewed Signals to Workspace** button on the **Scope Parameters > History** pane to set the **Log signal data** check boxes for attached signals
- Supports simulation stepping
- Supports fast restart
- Support for legends
- Ability to change line properties, axes, and figure colors.

Scope Viewers:

- Includes all the enhancements for Floating Scope blocks
- Supports the `Dataset` format for the **Signal Logging format** parameter in the **Configuration Parameters > Data Import/Export** pane.
- The functionality for the removed Floating Scope viewer is now included with Scope viewers.

See Scope and Floating Scope, Scope Viewer Tasks, Signal Selector.

Component-Based Modeling

Model Templates: Build models using design patterns that serve as starting points to solve common problems

Model templates enable reuse of settings and sharing of knowledge. Create models from templates to encourage best practices and take advantage of previous solutions to common problems. Instead of the blank canvas of a new model, select a template to help you get started.

Use built-in templates or create templates from models that you already configured for your environment or application.

For details, see Create a New Model.

Simulink Functions: Create and call functions across Simulink and Stateflow

The Simulink Function block serves as a starting point for implementing functions using Simulink blocks. Simulink Function responds to the Function Caller block or to a call from Stateflow using a provided function prototype.

The Function Caller block, using a provided function prototype, invokes a function implementation, such as the Simulink Function block, or a Stateflow exported chart function.

Interface Display: View and trace the input and output signals of a model or subsystem

For Simulink models, the Interface feature helps you to understand and manage your model interfaces. You can start with the input and output connections at the edge of the model diagram and examine signal paths going in and coming out of a component. You can trace individual signals, buses, and elements of a bus to learn their uses in the model. The feature also displays characteristics of the inputs and outputs, such as dimensions, port data types, and sample times. Select **Display** > **Interface** in your Simulink model to turn on this display.

Model reference conversion enhancements

In R2014b, the Model Reference Conversion Advisor and the
Simulink.SubSystem.convertToModelReference function provide new automatic fix
options for converting a subsystem in a model that meets either of these conditions.

- The model contains a Data Store Memory block that Data Store Read or Data Store
  Write blocks access across the subsystem boundaries.
- The top model uses tunable parameters.

The `Simulink.SubSystem.convertToModelReference` function has a new
`PropagateSignalStorageClass` argument. Set that argument to `true` to have the
conversion propagate the signal storage class. For example:

```
open_system('sldemo_mdlref_conversion');
Simulink.SubSystem.convertToModelReference(gcb,'test_model',...
'PropagateSignalStorageClass',true)
```

Compatibility Considerations

The automatic fix for Data Store Memory blocks creates `Simulink.Signal` objects and
redirects Data Store Read and Data Store Write blocks to access those signal objects. The
automatic fix for tunable parameters creates `Simulink.Parameter` objects for tunable
parameters.

The conversion process creates data. Conversion data that the top model and the
new referenced model share is saved in the data dictionary, if the top model uses one.
Otherwise, the data is saved in a MAT-file.

- If the top model does not use a data dictionary, the conversion stores the data in a
  MAT-file whose name uses this format: `<model_name>_conversion_data.mat`.

  - If the model has callbacks or if you have scripts that rely on the previous variable
    names and locations, load the `<model_name>_conversion_data.mat` file before
    running the callbacks or scripts.

- If the top model uses a data dictionary, the conversion action depends on the state of
  the data dictionary.

  - If the data dictionary does not need to be saved, the conversion stores the data in
    the data dictionary.

7-11

- If the data dictionary needs to be saved, the conversion does not save the data. You need to save the data dictionary when the conversion is complete.

Include Simulink Models as Variant Choices

Previously, you could only include subsystems as variant choices inside a Variant Subsystem block. If you wanted to include a model as a variant choice, you had to first wrap the model inside a Subsystem block and then include the subsystem block as the variant choice.

In R2014b, you can include a Simulink model as a variant choice inside a Variant Subsystem block without wrapping the model inside a Subsystem block.

You cannot include a Model block that contains variants inside a Variant Subsystem block. When you attempt this inclusion, Simulink suggests converting the Model block into a Subsystem block.

For example, consider a model block called **IntelligentController** that has two variants: `FuzzyLogicController` and `KalmanFilterController`. If you add this block inside a Variant Subsystem block, Simulink converts the model into a subsystem containing two model blocks: one representing `FuzzyLogicController` and the other, `KalmanFilterController`.

For more information, see Define, Configure, and Activate Variant Choices.

Arithmetic and Bit-Wise Operators in Variant Condition Expressions

In R2014b, you can use arithmetic and bit-wise operators in variant condition expressions, provided the expressions evaluate to a Boolean value.

For a list of supported operators, see Operators and Operands in Variant Condition Expressions.

Export of chart-level functions in export-function models

In R2014b, you can export graphical functions in Stateflow charts residing in export-function models. For more information, see Export Stateflow Functions for Reuse.

Project and File Management

Block Dependencies in Impact Graph: Highlight the blocks affected by changes made to project files

Perform fine-grained dependency analysis using the Impact graph in Simulink Project. The Impact graph now displays which blocks have dependencies. For example, to find the impact of modifying a library, you can find design and test files dependencies. You can then expand the dependent files to see which subsystems have dependencies. You can view dependent blocks, models and libraries, and double-click to highlight the blocks in the models.

Other improvements in dependency analysis tools:

- Impact graph items now have popups when you mouse over a file, so that you can read the text and expand files without needing to change the zoom.
- Line routing in the graph is improved to reduce line crossings and make it easier to interpret large projects.
- Dependency and Impact analysis views have new toolstrips with reorganized tools to fit common workflows. The toolstrips enable discovery of tools previously hidden in context menus.
- You can now exclude external toolboxes from dependency analysis, which can avoid time-consuming analysis.

For details, see Perform Impact Analysis.

Identify modified or conflicted folder contents using source control summary status

In Simulink Project, folders now display rolled-up source control status. This makes it easier to locate changes in files, particularly conflicted files. You can hover over the source control status for a folder to view a tooltip displaying how many files inside are modified, conflicted, added or deleted.

For details, see View Modified Files and Resolve Conflicts.

Simplified file views in Simulink Project

In Simulink Project, the file views are simplified to combine the All files and Project Files nodes. You can find all files in the Files view, and use a filter if you only want to display files in the project. You can use drag and drop to add, move, and remove files from the project.

The source control and project information nodes are also combined to simplify the project tree.

For details of all these views, see Try Simulink Project Tools with the Airframe Project.

Simplified browsing and sharing of project templates

Simulink Project now discovers templates using the MATLAB path. This enables simpler browsing and sharing of templates. You no longer need to import templates or manage a separate template path. If a colleague emails you a template, you can use it by placing it in the current folder or elsewhere on the MATLAB path. The template manager has been removed. You can still access all your existing templates using the Add Template dialog when creating a project.

For details, see Create a New Project Using Templates.

SVN and Git example Simulink Projects

Try out source control features using new SVN and Git example Simulink Projects:

```
sldemo_slproject_airframe_svn
sldemo_slproject_airframe_git
```
For details, see Try Simulink Project Tools with the Airframe Project.

Data Management

Root Import Mapping tool

The Root Import Mapping tool has been updated. In addition to a new user interface, the tool lets you:

- Import signals from base workspace, MAT-files, and Microsoft Excel (Windows systems only) files
- Export signals
- Visualize signals
- Save and import scenarios
- Create new scenarios

For more information, see Import and Map Root-Level Inport Data.

Compatibility Considerations

The Root Import Mapping tool no longer accepts data in the time expression format.

Minimize and maximize buttons for the Configuration Parameters dialog box

You can minimize and maximize the Configuration Parameters dialog box using buttons on the title bar.

Overflow diagnostics to distinguish between wrap and saturation

You can now separately control the diagnostics for overflows that wrap and overflows that saturate by setting each diagnostic to error, warning, or none. These controls simplify debugging models in which only one type overflow is of interest. For example, if you need to detect only overflows that wrap, in the **Data Validity** pane of the Configuration Parameters dialog box you can set **Wrap on overflow** to error or warning, and set **Saturate on overflow** to none.

Change in behavior of isequaln

Previously, when you used function `isequaln` to compare two Simulink data objects, the function compared only the handles of the two objects. This behavior was incorrect and did not conform to the intended behavior of isequaln in MATLAB. Consider the following example:

```
a = Simulink.Parameter;
b = Simulink.Parameter;
isequaln(a,b);
ans = false
```

Now, the behavior of `isequaln` has changed to conform to the behavior of isequaln in MATLAB. Now, `isequaln` compares two Simulink data objects by comparing their individual property values. Based on the above example, provided objects `a` and `b` have the same property values, the new result will be as follows:

```
a = Simulink.Parameter;
b = Simulink.Parameter;
isequaln(a,b);
ans = true
```

Compatibility Considerations

If you are using the `isequaln` function in your MATLAB code to compare Simulink data objects, check your code to ensure that it still works correctly.

Change in Simulink check for types derived from Simulink.IntEnumType

Previously, Simulink generated an error if any of the underlying values of a type derived from `Simulink.IntEnumType` did not fit on the emulation target. This check is defined by the configuration parameter `TargetBitPerInt`.

Now, Simulink generates an error if any of the underlying values of a type derived from `Simulink.IntEnumType` do not fit on the production target. This check is defined by the configuration parameter `ProdBitPerInt`.

Compatibility Considerations

Simulation errors can occur if any of the underlying values of types derived from `Simulink.IntEnumType` do not fit on your production target, even if those values previously fit on your emulation target.

Methods no longer inherited by Simulink enumerations

The base class `Simulink.IntEnumType` no longer defines these methods:

- `getDescription`
- `getHeaderFile`
- `getDataScope`
- `addClassNameToEnumNames`

You can define these methods in your enumeration classes, but classes derived from `Simulink.IntEnumType` no longer inherit these methods.

Previously, you called these inherited methods at the command prompt to query attributes of an enumerated type derived from `Simulink.IntEnumType`. For example, you called the method `getDefaultValue` to query the default enumeration member. Now, to query attributes of an enumerated type, you use the function `Simulink.data.getEnumTypeInfo`.

The table shows how to use `Simulink.data.getEnumTypeInfo`, instead of the methods, to query the attributes of an enumerated type `BasicColors`.

Previous method call	New function call
`BasicColors.getDescription`	`Simulink.data.getEnumTypeInfo(...` `'BasicColors','Description')`
`BasicColors.getHeaderFile`	`Simulink.data.getEnumTypeInfo(...` `'BasicColors','HeaderFile')`
`BasicColors.getDataScope`	`Simulink.data.getEnumTypeInfo(...` `'BasicColors','DataScope')`
`BasicColors.addClassNameToEnumNames`	`Simulink.data.getEnumTypeInfo(...` `'BasicColors','AddClassNameToEnumNames')`

Compatibility Considerations

If you run MATLAB code that uses these previously inherited methods without overriding them, MATLAB returns errors.

However, code that overrides these methods, as described in Customize Simulink Enumeration, is unaffected.

Connection to Educational Hardware

More Arduino Support: Run your model on Arduino Leonardo, Mega ADK, Mini, Fio, Pro, Micro and Esplora boards

Updates to Simulink Support Package for Arduino Hardware: You can run your model on Arduino Leonardo, Mega ADK, Mini, Fio, Pro, Micro, and Esplora boards.

Documentation installation with hardware support package

Starting in R2014b, each hardware support package installs with its own documentation. See Simulink Supported Hardware for a list of support packages available for Simulink, with links to documentation.

Signal Management

Signal name inheritance from bus object elements

For a Bus Creator block that specifies a bus object, you can now have bus signal names inherit signal names from the corresponding element names in the bus object. To inherit signal names from bus element names, clear the new **Override bus signal names from inputs** check box. This approach:

- Enforces strong data typing.
- Avoids your having to enter a signal name multiple times. Without this option, you need to enter the signal names in the bus object and in the model, which can lead to accidentally creating signal name mismatches.
- Supports the array of buses requirement to have consistent signal names across array elements.

The Bus Creator block parameters dialog box has been reorganized to group related parameters. For details, see the Bus Creator block reference page.

Compatibility Considerations

In R2014b, if you open a model created before R2014b that uses a bus object for the output data type, clearing the **Override bus signal names from inputs** parameter might require you to change the model because:

- A signal name in a downstream Bus Selector or Bus Assignment block might no longer be the same.

  Change any selected signal names in the Bus Selector or Bus Assignment block dialog boxes that do not match the corresponding bus object element names.
- Signal names in signal logging data might change.

  Update scripts to reflect signal logging signal names that match the corresponding bus object names.

Faster and more flexible Simulink.Bus.createMATLABStruct function

If you use the Simulink.Bus.createMATLABStruct function repeatedly for the same model (for example, in a loop in a script), you can now improve performance by avoiding

multiple model compilations. For improved speed, put the model in compile before using the function multiple times.

Also, you can now use a cell array of bus object names as an input argument for the function.

Block Enhancements

Nearest interpolation method available for n-D Lookup Table Block

The 1-D Lookup Table, 2-D Lookup Table, and n-D Lookup Table blocks have the option to select `Nearest` for interpolation methods.

MATLAB System block updates

The MATLAB System block:

- Can now be contained in a For Each Subsystem block. The new supportsMultipleInstanceImpl method enables using System objects in Simulink For Each subsystems. Include this method in your System object class definition file when you define a new kind of Simulink. For more information, see System Objects in For Each Subsystems.

- The **New** > **Block Extension** option has been renamed to **New** > **Simulink Extension**.

Level-1 MATLAB S-Functions

Support for Level-1 MATLAB S-Functions will be removed in a future release.

Compatibility Considerations

For information on alternatives to creating custom blocks, see Comparison of Custom Block Functionality.

Unfiltered-derivative option in discrete-time PID Controller blocks

You can now specify an unfiltered derivative term in the discrete-time PID Controller and PID Controller (2DOF) blocks. Previously, these blocks required a finite derivative filter constant on the derivative term.

To specify an unfiltered derivative, in the **Main** pain of the block dialog box, uncheck **Use filtered derivative**. Unchecking this option replaces the derivative filter with a Discrete Derivative block. The option is checked by default for compatibility with previous versions.

MATLAB Function Blocks

Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions

Image Processing Toolbox

bwdist	imadjust	intlut	ordfilt2
bwtraceboundary	imclearborder	iptcheckmap	rgb2ycbcr
fitgeotrans	imlincomb	medfilt2	stretchlim
histeq	imquantize	multithresh	ycbcr2rgb

For the list of Image Processing Toolbox functions supported for code generation, see Image Processing Toolbox.

Computer Vision System Toolbox

vision.DeployableVideoPlayer on Linux.

For the list of Computer Vision System Toolbox functions supported for code generation, see Computer Vision System Toolbox.

Code generation for additional Communications System Toolbox and DSP System Toolbox functions and System objects

Communications System Toolbox

- iqcoef2imbal
- iqimbal2coef
- comm.IQImbalanceCompensator

For the list of Communications System Toolbox™ functions supported for code generation, see Communications System Toolbox.

DSP System Toolbox

- dsp.CICCompensationDecimator

- dsp.CICCompensationInterpolator
- dsp.FarrowRateConverter
- dsp.FilterCascade

  You cannot generate code directly from this System object. You can use the `generateFilteringCode` method to generate a MATLAB function. You can generate C/C++ code from this MATLAB function.
- dsp.FIRDecimator for transposed structure
- dsp.FIRHalfbandDecimator
- dsp.FIRHalfbandInterpolator
- dsp.PeakToPeak
- dsp.PeakToRMS
- dsp.PhaseExtractor
- dsp.SampleRateConverter
- dsp.StateLevels

For the list of DSP System Toolbox functions and System objects supported for code generation, see DSP System Toolbox.

Code generation for `ode23` and `ode45` ordinary differential equation solvers in MATLAB

- ode23
- ode45
- odeget
- odeset

See Numerical Integration and Differentiation in MATLAB.

Code generation for additional MATLAB functions

Data and File Management in MATLAB

- feof
- frewind

See Data and File Management in MATLAB.

Linear Algebra in MATLAB

- ishermitian
- issymmetric

See Linear Algebra in MATLAB.

String Functions in MATLAB

str2double

See String Functions in MATLAB.

Code generation for additional MATLAB function options

- `'vector'` and `'matrix'` eigenvalue options for `eig`
- All output class options for `sum` and `prod`
- All output class options for `mean` except `'native'` for integer types
- Multidimensional array support for `flipud`, `fliplr`, and `rot90`
- Dimension to operate along option for `circshift`

See Functions and Objects Supported for C and C++ Code Generation — Alphabetical List.

Code generation for enumerated types based on built-in MATLAB integer types

In previous releases, enumerated types used in MATLAB Function blocks were based on the `Simulink.IntEnumType` class. In R2014b, you can also base an enumerated type on one of the following built-in MATLAB integer data types:

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`

You can use the base type to control the size of the enumerated type in generated C and C++ code. You can choose a base type to:

- Represent an enumerated type as a fixed-size integer that is portable to different targets.
- Reduce memory usage.
- Interface to legacy code.
- Match company standards.

The base type determines the representation of the enumerated types in generated C and C++ code. For the base type `Simulink.IntEnumType`, the code generation software generates a C enumeration type. For example:

```
typedef enum {
  GREEN = 1,
  RED
} LEDcolor;
```
For the built-in integer base types, the code generation software generates a `typedef` statement for the enumerated type and `#define` statements for the enumerated values. For example:

```
typedef int16_T LEDcolor;

#define GREEN                           ((LEDcolor)1)
#define RED                             ((LEDcolor)2)
```

See Enumerated Types Supported in MATLAB Function Blocks.

Code generation for function handles in structures

You can now generate code for structures containing fields that are function handles. See Function Handle Definition for Code Generation.

Collapsed list for inherited properties in code generation report

The code generation report displays inherited object properties on the **Variables** tab. In R2014b, the list of inherited properties is collapsed by default.

Model Advisor

New check for Unit Delay and Zero-Order Hold blocks that perform rate transition

A new check in Model Advisor and Upgrade Advisor identifies Unit Delay and Zero-Order Hold blocks that are used for rate transition between input and output signals. The check prompts you to replace these blocks with actual Rate Transition blocks. The replacement provides accurate information about block transfer rates and enables traceability. For more information, see Check Unit Delay and Zero-Order Hold blocks for rate transition.

Highlighted configuration parameters from Model Advisor reports

When you click a link to a configuration parameter from a Model Advisor report, the parameter is highlighted in the Configuration Parameters dialog box.

R2014a

Version: 8.3

New Features

Bug Fixes

Compatibility Considerations

Simulink Editor

Annotations with rich text, graphics, and hyperlinks

In addition to plain text and text formatted with TeX, annotations can now include:

- Rich text, which gives you the ability to format text and to add tables and lists, as you would using Microsoft Word
- Images, either by copying and pasting or by importing a graphics file
- Hyperlinks to Web pages or other documents

For details, see Create an Annotation with a Link, Lists, and an Image.

Diagnostic Viewer to collect information, warnings, and error messages

In addition to displaying errors and warnings generated during simulation, the Diagnostic Viewer now displays information at the time of update diagram and build. The messages are displayed in a hierarchical structure within tabs for each model. For details, see Manage Errors and Warnings

Compatibility Considerations

The diary function does not intercept messages, errors, warning, and information transmitted to the Diagnostic Viewer.

Therefore, if you use the diary function to log messages, errors, warnings, and information generated during model build and simulation, replace instances of diary with `sldiagviewer.diary` in one of these ways.

- `sldiagviewer.diary('filename','encoding')`, where both *filename* and *encoding* are optional arguments.
  - The default value for *filename* is `diary.txt` in the current folder.
  - A valid value for encoding is `UTF-8`. If you do not specify a value, the default encoding value is set.
  - The command toggles the logging state of the specified file.
  - You can keep multiple log files active simultaneously.

- `sldiagviewer.diary('on')` and `sldiagviewer.diary('off')` toggle the logging state of the file specified in the last executed `sldiagviewer.diary` command.

  If no file was specified in the last command, the logging state of the default file is toggled.

Option to bring contents of a hierarchical subsystem into the parent subsystem with one click

You can now expand subsystem contents to flatten the model hierarchy. Expanding a subsystem is useful when refactoring a model. Flattening a model hierarchy can be the end result, or just one step in refactoring. For example, you could pull a set of blocks up to the parent system by expanding the subsystem, deselect the blocks that you want to leave in the parent, and then create a subsystem from the remaining selected blocks.

For details, see Expand Subsystem Contents.

Support for native OS touch gestures, such as pinch-to-zoom and panning

MathWorks® supports the use of multitouch gestures for panning and zooming on the Microsoft Windows with a Windows 7 certified or Windows 8 certified touch display.

- Zoom by spreading two fingers.
- Zoom in by pinching two fingers together.
- Pan by dragging two fingers.

Other supported Simulink platforms that also support multitouch gestures might also support pan and zoom gestures, but MathWorks has not fully tested those platforms.

Operating system print options for models

The Print Model dialog box includes a **Print using system dialog** button that opens the print dialog box for your operating system. The operating system print dialog box provides printing options for models in addition to those that the Print Model dialog box provides. For example, you can use the operating system print dialog box for double-sided printing, color printing (if your print driver supports color printing), and nonstandard paper sizes. For details, see Specify the Page Layout and Print Job.

Preference for line crossing style

By default, straight signal lines that cross each other but are not connected display a slight gap before and after the vertical line where it intersects the horizontal line. You can change the line crossing style to line hops or solid lines. Use **Simulink Preferences > Editor Defaults > Line crossing style**. For details, see Line crossing style.

Scalable graphics output to clipboard for Macintosh

On Macintosh platforms, when you copy a model to the clipboard, Simulink now saves the model in a scalable format, in addition to a bitmap format. When you paste from the clipboard to an application, that application selects the format that best meets its requirements. For details, see Export Models to Third-Party Applications.

Sliders, dials, and spinboxes available as parameter controls in masks

In this release, Simulink provides the capability to control mask parameters through three additional widgets: sliders, dials, and spinboxes. For details, see Parameters & Dialog Pane.

Component-Based Modeling

Option to choose default variants

In R2014a, you can specify a default variant choice. If no other variant is active, Simulink selects and uses the default choice.

For more information, see Set Default Variant.

Option to choose variants that differ in number of input and output ports

In this release, variant subsystems can have different numbers of inports and outports, provided that they satisfy the following conditions:

- The inport names are a subset of the parent variant subsystem's inport names.
- The outport names are a subset of the parent variant subsystem's outport names.

During simulation, Simulink disables the inactive ports in a variant subsystem block.

Advisor-based workflow for converting subsystems to Model blocks

The new Model Reference Conversion Advisor simplifies the process of converting a subsystem to a referenced model. The advisor includes **Fix** buttons to automatically update the model for a successful conversion. After converting the subsystem, by default the advisor automatically updates the model, removing the Subsystem block and adding a Model block that references the newly created referenced model. For details, see Convert a Subsystem to a Referenced Model.

The Simulink.SubSystem.convertToModelReference command now has a `UseConversionAdvisor` argument, which opens the Model Reference Conversion Advisor, and an `AutoFix` argument, which automatically fixes several kinds of conversion issues.

Single-model workflow for algorithm partitioning and targeting of multicore processors and FPGAs

You can use the concurrent execution dialog box to configure a model for concurrent execution on heterogeneous targets.

A new example has been added. Navigate to **Simulink** > **Examples** > **Modeling Features** > **Modeling Concurrency** > **Modeling Concurrent Execution on Multicore Targets**.

For a list of supported heterogeneous targets, see Supported Heterogeneous Targets.

Easier MATLAB System block creation via autocompletion and browsing for System object names

The MATLAB System block dialog box has new browse and autocompletion capabilities to specify the System object name. It also allows creation of new System objects from templates.

Improved algebraic loop handling and reduced data copies with the Bus Selector block

The Bus Selector block processing now reduces:

- Artificial algebraic loops involving nonvirtual buses
- Data copies during bus element selection
- The number of lines of generated code

Compatibility Considerations

The Bus Selector block performance enhancements result in these compatibility considerations.

- Attaching a non-auto storage class at output of Bus Selector causes an error.

  Workaround: Insert a Signal Conversion block after the Bus Selector block, and specify the storage class on the output of the Signal Conversion block.

- In model reference Accelerator mode, Bus Selector output signal logging using the ModelDataLogs format causes an error.

  To work around this issue, change the signal logging format to Dataset or insert a Signal Conversion block after the Bus Selector block and log the Signal Conversion block output (instead of the Bus Selector block output).

- The priority specified on a Bus Selector block is ignored.

- A Bus Selector block connected to root Outport block in referenced model honors the **Invalid root Inport/Outport block connection** diagnostic. This is an issue only if the diagnostic is set to `Error` or `Warning`. There is no impact on generated code.

  Workaround: Insert a Signal Conversion block after the Bus Selector block.

Faster response time when opening bus routing block dialog boxes and propagating signal labels

Opening the Bus Creator, Bus Selector, or Bus Assignment block dialog box is faster. Propagation of signal labels is also faster.

Usability enhancements to configure a model for concurrent execution on a target

Modeling for concurrent execution has the following enhancements:

- You can use MATLAB System blocks at the top level of a model to model parallel computations. For more information, see Model Parallel Computations.
- You can clear the **Solver** > **Allow tasks to execute concurrently on target** check box in the Configuration Parameters dialog for a referenced model. In this case, Simulink will not report a parameter mismatch for the model hierarchy. When this option is cleared, any Rate Transition blocks in the referenced model may have the `Ensure deterministic data transfer (maximum delay)` option selected.
- Simulink now handles the data transfer for the rate transitions occurring at the top level of a model. For more information, see Configure Your Model.
- There are new function to work with models for concurrent execution. Use these functions instead of the command-line interface from previous releases. For more information, see Command-Line Interface for Concurrent Execution.

Compatibility Considerations

Use the new functions instead of the following:

- `Simulink.SoftwareTarget.concurrentExecution`
- `Simulink.SoftwareTarget.AperiodicTrigger`
- `Simulink.SoftwareTarget.PeriodicTrigger`

- `Simulink.SoftwareTarget.PosixSignalHandler`
- `Simulink.SoftwareTarget.Task`
- `Simulink.SoftwareTarget.TaskConfiguration`
- `Simulink.SoftwareTarget.WindowsEventHandler`
- `Simulink.SoftwareTarget.Trigger`

Default setting of Underspecified initialization detection diagnostic is `Simplified`

Simplified initialization mode is the new default mode of initialization for all models created in R2014a. Simplified initialization mode has improved output consistency over classic initialization mode. For more information, see Conditional Subsystem Output Initialization.

However, any new configuration sets created using Simulink.ConfigSet still use classic initialization mode.

Discrete-Time Integrator block has dialog box changes for initialization

The **Use Initial value as initial and reset value for** parameter of the Discrete-Time Integrator block has been replaced by the **Initial Condition Setting** parameter. The new parameter provides options to carry out either output or state initialization. An additional option of `Compatibility` does exist for this parameter, but use it only for compatibility purposes.

System objects Propagates mixin methods

Four new methods have been added to the Propagates mixin class. You use this mixin when creating a new kind of System object for use in the MATLAB System block in Simulink. You use these methods to query the input and specify the output of a System object.

- propagatedInputComplexity
- propagatedInputDataType
- propagatedInputFixedSize
- propagatedInputSize

Simulation Analysis and Performance

Reduced setup and build time for Model blocks when using Rapid Accelerator mode

Building a model that uses model referencing in Rapid Accelerator mode is faster than in previous releases, because Simulink reuses the simulation target.

If you have Parallel Computing Toolbox, you can enhance Rapid Accelerator build speed further by using parallel builds.

In R2014a, Simulink stores Rapid Accelerator build artifacts in the simulation cache folder instead of the code generation folder. This change avoids cluttering the code generation folder with simulation artifacts.

Compatibility Considerations

- If you have a script that relies on Rapid Accelerator build artifacts being stored in the code generation folder, the R2014a change to the artifact storage location requires you to update that script only if you specify two different folders for the Simulink preferences Simulation cache folder and Code generation folder.
- In R2014a, the Upgrade Advisor checks that S-functions work properly in top model Rapid Accelerator simulation. The advisor identifies and assists you with updating S-functions that meet all of the following conditions:
  - The S-function was created in R2013b or earlier, using either the S-Function Builder block or the Legacy Code Tool.
  - The S-function uses a bus signal as an input or output.
  - Simulink has added padding to that bus signal.

Before you simulate such S-functions in a top model in Rapid Accelerator mode, regenerate the S-functions with the tool used for creating them. The Model Advisor automatically regenerates as many of these S-functions as it can and identifies any other S-functions that you must regenerate.

Performance Advisor checks that validate overall performance improvement for all suggested changes and set code generation option for MATLAB System block

- Performance Advisor validates the overall performance improvement to your model using a final validation check. If performance is worse than baseline, Performance Advisor discards all changes and loads the original model. For more information, see Final Validation.
- Performance Advisor uses the **Check MATLAB System block simulation mode** check to identify which MATLAB System blocks can generate code and changes the **Simulate using** parameter value to `Code generation` where possible. For more information, see Check MATLAB System block simulation mode.

Improved navigation of the Performance Advisor report

For improved navigation and readability, in the Performance Advisor HTML report, you can:

- Filter the report to display results for checks that pass, warn, or fail.
- Display check results based on a keyword search.
- Quickly navigate to sections of the report using a navigation pane of the contents.
- Expand and collapse content in the check results.

For more information, see Use Performance Advisor Reports.

Block behavior for asynchronous initiator with constant sample time

In the simulation target workflow, Simulink displays a warning for an asynchronous initiator with constant sample time. To avoid the warning, set the sample time parameter of the asynchronous initiator to `inherited`. This change in sample time parameter does not affect the code generation workflow.

Global setting for validation of checks in Performance Advisor

You can enable validation for all selected checks in Performance Advisor using a global setting. Previously, you could only enable validation for checks in Performance Advisor individually. For more information, see Select Validation Actions for the Advice.

Guided setup in Performance Advisor

The user interface in Performance Advisor has been enhanced with a guided setup and workflow. The interface helps you to follow the workflow in Performance Advisor and also select settings required for performance optimization runs. For more information, see Prepare a Model for Performance Advisor.

Project and File Management

Branching support through Git source control

Git integration with Simulink Project provides distributed source control with support for creating and merging branches and working offline. You can manage your models and source code using Git within Simulink Project.

For details, see Set Up Git Source Control.

Tip: If you want to add version control to your project files without sharing with another user, you can create a local Git repository in your sandbox with four clicks. For details, see Add a Project to Git Source Control.

Comparison of project dependency analysis results

You can compare a Simulink Project impact analysis graph with a previously saved result of dependency analysis. This creates an interactive report you can use to investigate how the structure of project dependencies has changed.

For details, see Save, Reload, and Compare Dependency Analysis Results.

Impact graph layout algorithm improved for easier identification of top models and their dependencies

Improved hierarchical layout algorithm for the Simulink Project Impact graph makes it easier to identify top models, now always on the left, and their dependencies, on the right. Graph layout is repeatable so the top model is always in the same place if you run dependency analysis multiple times. Graph performance is also faster. Dependencies are layered to vertically line up all files referenced by the same file and minimize layer crossings. These layers make it easier to identify dependencies at the same level. This makes it easier to see connections between a top model and its dependencies.

For details, see Perform Impact Analysis.

Impact analysis example for finding and running impacted tests

The `sldemo_slproject_impact` example shows how to perform impact analysis in Simulink Project to find and run impacted tests to validate a change. You can search for dependencies of modified files to identify the tests you need to run. You can run a batch processing function on the files found by impact analysis and examine the results in Simulink Project.

For details, see Perform Impact Analysis with a Simulink® Project.

Performance improvements for common scripting operations such as adding and removing files and labels

Common scripting operations for programmatically adding and removing files and file labels in a Simulink Project are now faster. For example, adding a label to 100 files is up to 40 times faster. Performance improvement depends on the project size.

For details, see Automate Project Management Tasks.

Conflict resolution tools to extract conflict markers

Source control tools can insert conflict markers in files. Simulink Project can identify conflict markers and offer to extract them and compare the files causing the conflict. You can then decide how to resolve the conflict.

For details, see Resolve Conflicts.

Updated Power Window Example

The Power Window Control Project example has been updated to take advantage of design concepts such as:

- Simulink Projects
- Referenced models
- Variant subsystems

For more information, see Power Window Case Study.

Data Management

Data dictionary for defining and managing design data associated with models

In R2014a, Simulink provides the ability to store, edit, and access design data using a data dictionary, which functions as a persistent repository of design data that your model uses.

For more information, see the following.

- What Is a Data Dictionary?
- Considerations before Migrating to Data Dictionary
- Migrate Single Model to Use Dictionary
- View and Revert Changes to Dictionary Entries

Rapid Accelerator mode signal logging enhanced to avoid rebuilds and to support buses and referenced models

In Rapid Accelerator mode, you can now log:

- Bus signals (including virtual, nonvirtual, and array of buses signals)
- Signals in referenced models

Also, in Rapid Accelerator mode, no rebuild occurs when you change:

- The **Configuration Parameters** > **Data Import/Export** > **Signal logging** parameter
- Any settings using the Signal Logging Selector

Simplified tuning of all parameters in referenced models

This release simplifies the way Simulink considers the `InlineParameters` option when it is set to `Off`. You can perform the following operations:

- Tune all block parameters in your model during simulation, either through the parameters themselves or through the tunable variables that they reference.

- Preserve the mapping between a block parameter and a variable in generated code even when the block parameter does not reference any tunable variables.
- Retain the mapping between tunable workspace variables and variables in generated code, irrespective of the `InlineParameters` setting.
- Set the value of `InlineParameters` to `Off` for model references.

These behaviors are consistent across models containing reusable subsystems and reference models.

The simplified behavior enhances the generated code and provides improved mapping between a block parameter and a variable in generated code.

Block parameter expression	Code generated previously	Code generated in R2014a

Expressions referencing global variables (e.g., `K+1`)	Variable name is not preserved. Block parameter name is preserved.
	
	`struct Parameters_model_ {`
	`  real_T Gain_Gain; // Expression:`
	`}`
	`y = model_P.Gain_Gain*u;`
		Expression is considered tunable. Variable name is preserved in code and is tunable.
		
		`real_T K = 2.0;`
		`y = (K+1)*u;`

Expressions referencing mask parameters for nonreusable subsystems (e.g., `MP*3`), the value of `MP` being a nontunable expression.	Variable name is not preserved. Block parameter name is preserved.
	
	`struct Parameters_model_ {`
	`  real_T Gain_Gain; // Expression:`
	`}`
	`y = model_P.Gain_Gain*u;`
		Expression is considered tunable. Variable name is substituted by parameter value.
		
		`struct Parameters_model_ {`
		`  real_T Subsystem_MP;`
		`}`
		`y = (model_P.Subsystem_MP * 3) * u;`

Expressions referencing model arguments (resp. mask parameters) for referenced models (resp. reusable subsystems) (e.g., `Arg+1`)	Variable name is not preserved. Block parameter name is preserved.
	
	`struct Subsystem {`
	`  Gain_Gain; // Expression: Arg+1`
	`}`
	`y = model_P.Subsystem1.Gain_Gain*u;`
		Variable name is preserved as an argument name.
		
		`subsystem(y, u, rtp_Arg) {`
		`  y = (rtp_Arg+1)*u;`
		`}`

`Simulink.findVars` supported in referenced models

In this release, Simulink provides the ability to search model reference hierarchies for variables that are used or not used.

See Simulink.findVars for information on how to run these searches from the command-line.

Saving workspace variables and their values to a MATLAB script

In a future release, Simulink will not support Simulink.saveVars, which provides the ability to save workspace variables to a MATLAB script.

Instead, you can use the MATLAB function matlab.io.saveVariablesToScript to perform this operation from the command line.

Compatibility Considerations

If your scripts contain references to the function Simulink.saveVars, replace these references with matlab.io.saveVariablesToScript.

Frame-based signals in the To Workspace block

In the To Workspace block parameters dialog box, if you set **Save format** to either `Array` or `Structure`, you can set the new **Save 2-D signals as** parameter. By default, this new parameter is set for sample-based signals. To have the To Workspace block treat the input signal as a frame-based signal, change the setting to `2-D array (concatenate along first dimension)`.

Use this new To Workspace block parameter to treat an input signal to the block as a frame-based signal. This method of having Simulink treat an input signal as frame-based configures the model to work in future releases, when Simulink will no longer support using a signal property to specify that a signal is frame-based.

For information about frame-based signals, in the DSP System Toolbox documentation, see Frame-Based Processing.

Compatibility Considerations

Before R2014a, if you added a To Workspace block to a model, and you set **Save format** to either `Array` or `Structure`, you did not need to change any block parameter settings to handle frame-based signals.

In R2014a, in addition to setting **Save format** to either `Array` or `Structure`, you need to set the new **Save 2-D signals as** parameter to `2-D array (concatenate along first dimension)`.

If you open a model created before R2014a that inputs a frame-based signal to a To Workspace block, Simulink sets the **Save 2-D signals as** parameter to `Inherit from input (this choice will be removed - see release notes)`. When you simulate the model, Simulink displays a warning, with a link to the Upgrade Advisor, which you can use to update your model to use the new frame-based signal handling approach (automatically setting **Save 2-D signals as** to `2-D array (concatenate along first dimension)`.

Simulation mode consistency for Data Import/Export pane output options parameter

The behavior for the **Configuration Parameters** > **Data Import/Export** > **Output options** parameter value of `Produce specified output only` is now consistent for Rapid Accelerator, Accelerator, and Normal mode. In all three simulation modes, this setting results in automatically generating output signal data for variable-step solvers for the start and stop times, as well as for the times that the user specifies.

Compatibility Considerations

Prior to R2014a, in Rapid Accelerator mode the `Produce specified output only` setting only generated data for the specified times, and did not automatically generate data for the start and stop times.

You need to update scripts that rely on:

- Indexing that assumes that the first generated value is the first data value that you specify using the `Produce specified output only` setting
- The number of the data points

Dimension mismatch handling for root Inport blocks improved

For the input and output of a root Inport block, Simulink no longer distinguishes between a dimension of 1 and a dimension that is a vector of ones.

For example, Simulink no longer reports an error when both of these conditions apply:

- Input data for a root Inport block includes an element whose dimension is [1x1].
- The root Inport block has as its data type for output a bus object whose corresponding element has a dimension of [1].

Compatibility Considerations

Before R2014a, Simulink reported an error for this kind of dimension mismatches. If you have tests that rely on Simulink reporting an error, you need to modify the tests to identify such mismatches explicitly.

`Simulink.Bus.createObject` support for structures of timeseries objects

You can use the Simulink.Bus.createObject function to create bus objects from a structure of MATLAB timeseries objects. This facilitates importing logged signal data.

Signal logging override for model reference variants

You can override programmatically a subset of signals for model reference variant systems, including:

- Model reference variants
- Model blocks that contain a Subsystem Variant block or model reference variant

To log a subset of signals for these model reference variant systems, set the SignalLoggingSaveFormat parameter to Dataset. For details, see Override Signal Logging Settings from MATLAB.

Improved To Workspace block default for fixed-point data

The block parameter **Log fixed-point data as fi object** is now enabled by default for the To Workspace block in the Simulink library. This causes Simulink to log data to the To Workspace block in a data format designed for fixed-point data.

Compatibility Considerations

You need to update any scripts that:

- Insert from the Simulink library a To Workspace block that logs fixed-point data
- Rely on the **Log fixed-point data as fi object** parameter not being enabled (which results in logging fixed-point data as doubles)

Legacy Code Tool support for 2–D row-major matrix

When you use the legacy_code function, you can now specify the automatic conversion of a matrix between a 2-D column-major format and a row-major format. Use the `convert2DMatrixToRowMajor` S-function option. The 2-D column-major format is used by MATLAB, Simulink, and the generated code. The row-major format is used by C. By default, the value is false (`0`).

Model Explorer property name filtering refined

In the Model Explorer **Filter Contents** edit box, when you enter a property name without specifying a value (for example, `BlockType:`), the **Contents** pane displays only those objects that have that property.

For details, see Filter Objects in the Model Explorer.

Connection to Educational Hardware

Support for Arduino Due hardware

You can use the Arduino Due support package independently of the Arduino support package for Arduino Uno, Arduino Mega 2560, and Arduino Nano hardware. The Arduino Due support package shares the common block library with the Arduino Support package including the Ethernet and the WiFi blocks. The target hardware for this support package is a 32–bit ARM architecture-based microcontroller.

To use this support package, install Arduino Due support package as follows:

1 In the Command Window, enter `supportPackageInstaller`.
2 Using `supportPackageInstaller`, install the Arduino Due support package.

Support for Arduino WiFi Shield hardware

You can use the Arduino WiFi Shield with the *Simulink Support Package for Arduino Hardware* to connect to wireless networks. The block library for the Arduino WiFi shield hardware includes WiFi TCP/IP and WiFi UDP blocks that enable you to design wireless communication in embedded systems. The WiFi shield supports wireless external mode simulation via TCP/IP.

The block library for the Arduino WiFi Shield hardware includes:

· Arduino WiFi TCP/IP Send and Arduino WiFi TCP/IP Receive blocks that enable TCP/IP based wireless communication.
· Arduino WiFi UDP Send and Arduino WiFi UDP Receive blocks that enable UDP based wireless communication.

To install or update this support package, perform the steps described in Install Support for Arduino Hardware.

For more information, see Arduino Hardware

Support for LEGO MINDSTORMS EV3 hardware

You can run Simulink models on LEGO MINDSTORMS EV3 hardware. You can also tune parameter values in the model, and receive data from the model, while it is running on LEGO MINDSTORMS EV3 hardware.

Use the **Simulink Support Package for LEGO MINDSTORMS EV3 Hardware** block library to access LEGO MINDSTORMS EV3 peripherals:

- Button
- Color Sensor
- Encoder
- Gyro Sensor
- Infrared Sensor
- Display
- Motor
- Speaker
- Touch Sensor
- Ultrasonic Sensor

To install or update this support package, perform the steps described in Install Support for LEGO MINDSTORMS EV3 Hardware.

For more information, see LEGO MINDSTORMS EV3 Hardware.

Updates to support for LEGO MINDSTORMS NXT hardware

You can use the dGPS Sensor from Dexter Industries with the *Simulink Support Package for LEGO MINDSTORMS NXT Hardware*. Use the GPS Sensor block to measure the latitude and longitude of your current position, or get the distance and angle from your current position to a destination latitude and longitude.

To install or update this support package, perform the steps described in Install Support for LEGO MINDSTORMS NXT Hardware.

For more information, see LEGO MINDSTORMS NXT Hardware.

Support for Samsung GALAXY Android devices

You can run Simulink models on the Samsung GALAXY S4 and Tab 2 10.1 devices. You can also tune parameter values in the model, and receive data from the model, while it is running on Samsung GALAXY S4 and Tab 2 10.1 devices.

Use the **Simulink Support Package for Samsung GALAXY Android Devices** block library to access the Samsung GALAXY Android hardware:

- Accelerometer
- Ambient Temperature Sensor
- Audio Capture
- Audio Playback
- Camera
- Display
- FromApp
- Gyroscope
- Humidity Sensor
- Light Sensor
- Location Sensor
- Pressure Sensor
- ToApp
- UDP Receive
- UDP Send

To install or update this support package, perform the steps described in Install Support for Samsung GALAXY Android Devices.

Block Enhancements

Enumerated data types in the Direct Lookup Table (n-D) block

The Direct Lookup Table (n-D) block now supports enumerated data types for its index and table data values.

Improved performance and code readability in linear search algorithm for Prelookup and n-D Lookup Table blocks

The Prelookup and n-D Lookup Table blocks have an improved linear search algorithm which improves performance and code readability.

System object file templates

The MATLAB System block has new templates to help create System objects.

Relay block output of fixed-in-minor-step continuous signal for continuous input

The Relay block now delivers fixed-in-minor-step continuous signals for continuous input. Previously, the Relay block output signal was a continuous output for a continuous input. The new sample time represents the behavior of the block more accurately.

MATLAB Function Blocks

Generating Simulation Target typedefs for imported bus and enumerated data types

You can configure Simulink to generate typedefs for imported bus and enumerated data types or to use typedefs you provide in a header file. In the **Simulation Target** pane of the **Configuration Parameters** dialog box, use the **Generate typedefs for imported bus and enumeration types** check box. This setting applies to model simulation for MATLAB Function blocks. For more information see Simulation Target Pane: General.

Complex data types in data stores

You can now access complex data types in Data Store Memory blocks and `Simulink.Signal` objects using MATLAB Function blocks.

Unicode character support for MATLAB Function block names

You can use international characters for MATLAB Function block names in the Simulink Editor.

Support for int64 and uint64 data types in MATLAB Function blocks

You can now use int64 and uint64 data types in MATLAB code inside MATLAB Function blocks. You cannot use int64 or uint64 types for input or output signals to MATLAB Function blocks.

Streamlined MEX compiler setup and improved troubleshooting

You no longer have to choose a compiler using `mex -setup`. `mex` automatically locates and uses a supported installed compiler. You can use `mex -setup` to change the default compiler. See Changing Default Compiler.

Code generation for additional Image Processing Toolbox functions

Image Processing Toolbox

| affine2d | im2uint16 | imhist |

bwpack	im2uint8	imopen
bwselect	imbothat	imref2d
bwunpack	imclose	imref3d
edge	imdilate	imtophat
getrangefromclass	imerode	imwarp
im2double	imextendedmax	mean2
im2int16	imextendedmin	projective2d
im2single	imfilter	strel

See Image Processing Toolbox.

Code generation for additional Signal Processing Toolbox, Communications System Toolbox, and DSP System Toolbox functions and System objects

Signal Processing Toolbox

- findpeaks
- db2pow
- pow2db

See Signal Processing Toolbox.

Communications System Toolbox

- comm.OFDMModulator
- comm.OFDMDemodulator

See Communications System Toolbox.

DSP System Toolbox

ca2tf	firhalfband	ifir	iirnotch
cl2tf	firlpnorm	iircomb	iirpeak
firceqrip	firminphase	iirgrpdelay	tf2ca

fireqint	firnyquist	iirlpnorm	tf2cl
firgr	firpr2chfb	iirlpnormc	dsp.DCBlocker

See DSP System Toolbox.

Code generation for MATLAB `fminsearch` optimization function, additional interpolation functions, and additional `interp1` and `interp2` interpolation methods

You can generate code for the interp1 function `'spline'` and `'v5cubic'` interpolation methods.

You can generate code for the interp2 function `'spline'` and `'cubic'` methods.

You can generate code for these interpolation functions:

- interp3
- mkpp
- pchip
- ppval
- spline
- unmkpp

See Interpolation and Computational Geometry in MATLAB.

You can generate code for these optimization functions:

- fminsearch
- optimget
- optimset

See Optimization Functions in MATLAB.

Code generation for `fread` function

You can now generate code for the `fread` function.

Enhanced code generation for `switch` statements

Code generation now supports:

- Switch expressions and case expressions that are noninteger numbers, nonconstant strings, variable-size strings, or empty matrices
- Case expressions with mixed types and sizes

If all case expressions are scalar integer values, the code generation software generates a C `switch` statement. At run time, if the switch value is not an integer, the code generation software generates an error.

When the case expressions contain noninteger or nonscalar values, the code generation software generates C `if` statements in place of a C `switch` statement.

Code generation for value classes with `set.prop` methods

You can now generate code for value classes that have `set.prop` methods.

Code generation error for properties that use `AbortSet` attribute

Previously, when the current and new property values were equal, the generated code set the property value and called the set property method regardless of the value of the `AbortSet` attribute. When the `AbortSet` attribute was true, the generated-code behavior differed from the MATLAB behavior.

In R2014a, the code generation software generates an error if your code has properties that use the `AbortSet` attribute.

Toolbox functions for code generation

See Functions and Objects Supported for C and C++ Code Generation — Alphabetical List and Functions and Objects Supported for C and C++ Code Generation — Categorical List.

Communications System Toolbox

- comm.OFDMModulator
- comm.OFDMDemodulator

Data and File Management Functions

fread

DSP System Toolbox Classes and Functions

ca2tf	firhalfband	ifir	iirnotch
cl2tf	firlpnorm	iircomb	iirpeak
firceqrip	firminphase	iirgrpdelay	tf2ca
fireqint	firnyquist	iirlpnorm	tf2cl
firgr	firpr2chfb	iirlpnormc	dsp.DCBlocker

Image Processing Toolbox

affine2d	im2uint16	imhist
bwpack	im2uint8	imopen
bwselect	imbothat	imref2d
bwunpack	imclose	imref3d
edge	imdilate	imtophat
getrangefromclass	imerode	imwarp
im2double	imextendedmax	mean2
im2int16	imextendedmin	projective2d
im2single	imfilter	strel

Interpolation and Computational Geometry Functions

- interp2
- interp3
- mkpp
- pchip
- ppval
- polyarea
- rectint
- spline

- unmkpp

Matrix and Array Functions

flip

Optimization Functions

- fminsearch
- optimget
- optimset

Polynomial Functions

- polyder
- polyint
- polyvalm

Signal Processing Toolbox

- findpeaks
- db2pow
- pow2db

Modeling Guidelines

Modeling guidelines for high-integrity systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0029: Usage of Assignment blocks
- himl_0004: MATLAB Code Analyzer recommendations for code generation
- himl_0005: Usage of global variables in MATLAB functions
- himl_0006: MATLAB code if / elseif / else patterns
- himl_0007: MATLAB code switch / case / otherwise patterns
- himl_0008: MATLAB code relational operator data types
- himl_0009: MATLAB code with equal / not equal relational operators
- himl_0010: MATLAB code with logical operators and functions

Model Advisor

Improved navigation of the Model Advisor report, including a navigation pane, collapsible content, and filters based on check status

For improved navigation and readability, the Model Advisor HTML report allows you to:

- Filter the report to display results for checks that pass, warn, or fail.
- Display check results based on a keyword search.
- Quickly navigate to sections of the report using a table-of-contents navigation pane.
- Expand and collapse content in the check results.

For more information, see View and Save Model Advisor Reports.

Option to run Model Advisor checks in the background

If you have a Parallel Computing Toolbox license, you can run the Model Advisor in the background, allowing you to continue working on your model during Model Advisor analysis. When you start a Model Advisor analysis run in the background, Model Advisor takes a snapshot of your model. The Model Advisor analysis does not reflect changes that you make to your model while Model Advisor is running. For more information, see Run Checks in the Background.

Upgrade Advisor check for get_param calls for block CompiledSampleTime

Upgrade Advisor has a new check to scan MATLAB files in your working environment for `get_param` function calls that return the block `CompiledSampleTime` parameter. For multirate blocks, Simulink returns this parameter as a cell array of pairs of doubles. Run this check to identify MATLAB code that accepts the block `CompiledSampleTime` parameter as a pair of doubles. You can edit these instances of code to accept a cell array of pairs of doubles.

For more information, see Check get_param calls for block CompiledSampleTime.

Upgrade Advisor check for signal logging in Rapid Accelerator mode

When simulating your model in Rapid Accelerator mode, you can run a check in Upgrade Advisor to see if signals logged in your model are globally disabled. Using this check, you can enable signal logging globally.

For more information, see Check Rapid Accelerator signal logging.

R2013b

Version: 8.2

New Features

Bug Fixes

Compatibility Considerations

New Simulink Editor

Ability to add rich controls, links, and images to customized block interfaces using the Mask Editor

The mask editor is enhanced to support designing mask dialogs using controls such as images, links, and buttons. For details, see Masking.

Content preview for subsystems and Stateflow charts

Without opening the blocks, in the Simulink Editor you can preview the content of the following hierarchical items:

- Subsystems
- Model blocks
- Stateflow charts and subcharts

For example, the Throttle & Manifold subsystem uses content preview, and the Compression subsystem does not.
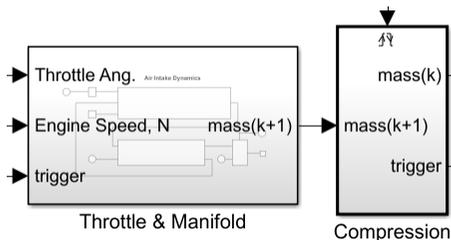
Content preview can help to make a model self-documenting.

For details, see Preview Content of Hierarchical Items.

Comment-through capability to temporarily delete blocks and connect input signals to output signals

You can comment through blocks such that they appear transparent during simulation. For more information, see Comment Blocks.

New options added to `find_system` command

You can exclude or include blocks from search using the `IncludeCommented` option of the find_system command. For more information, see Modeling Basics.

Visual cues for signal lines that cross

Unconnected signal lines that cross have a small break on either side of the intersection, to show that the signals are not connected. For example:
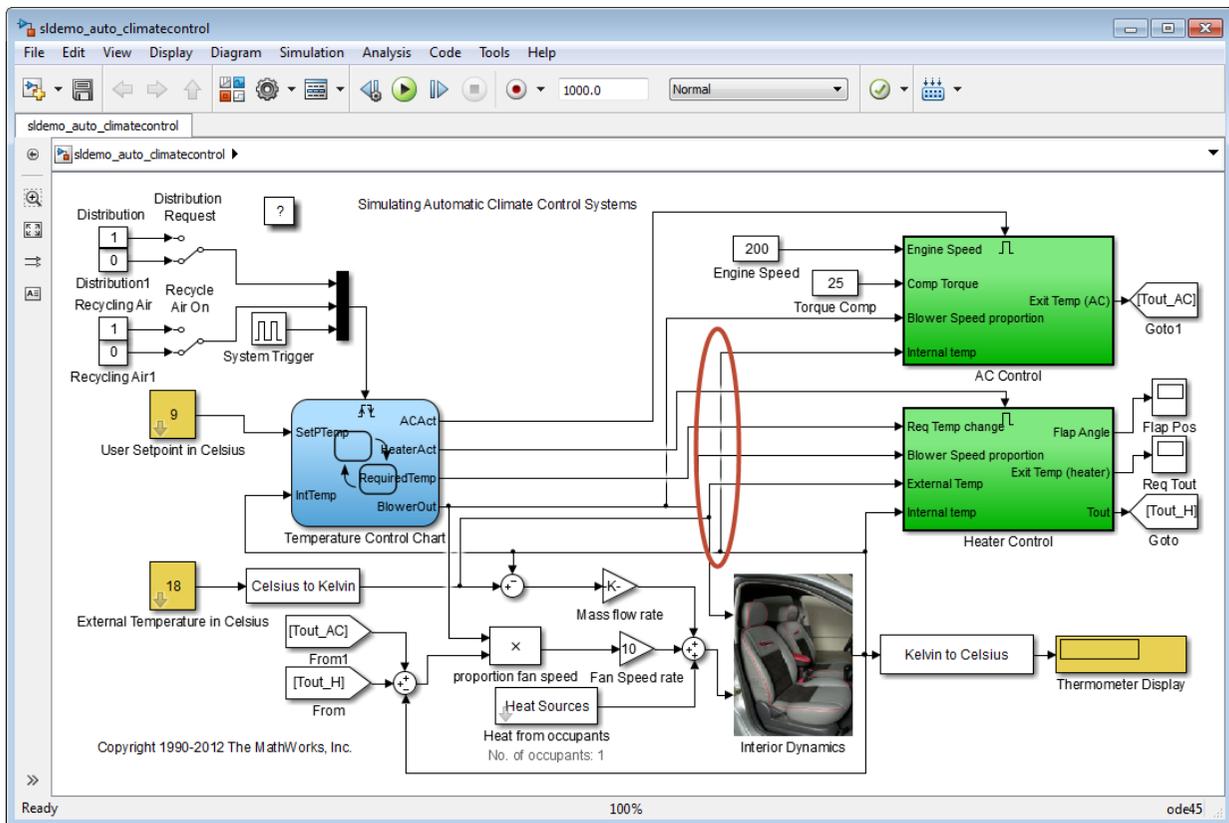
For details, see Signal Line Crossings.

UTF-16 character support for block names, signal labels, and annotations in local languages

You can use international characters when editing text in the Simulink Editor for the following:

- Block names (the text below the block)
- Block annotations (text in the Block Properties dialog box **Annotations** tab, for block property tokens such as `%<BlockDescription>`)
- Signal names (in the signal label text box below a signal)
- Model annotations

You can use international text for block names and signal names when using MATLAB commands such as `find_system`.

Unified Print Model dialog box for printing

The reorganized Print Model dialog box provides the same printing interface on Microsoft Windows, Macintosh, and Linux platforms.

To consolidate model printing options, the Print Model dialog now includes:

- Paper type
- Page orientation

For details, see Printing Options.

Compatibility Considerations

The reorganized Print Model dialog box no longer has these options:

- `Selection` for **Print Range**

  You can use tiled printing and options such as **Current system and below** to specify the part of a model to print.

- **Properties**

  Instead, use the **File** > **Print** > **Printer Setup** > **Properties** dialog box.

Block Parameters dialog box access from Block Properties dialog box

In the Block Properties dialog box, you can open the Block Parameters dialog box by using the **Open Block** link. For details, see Block Properties Dialog Box.

Notification bar icon indicator for multiple notifications

After you simulate a model, if there are multiple notifications, the Simulink Editor notification bar icon becomes an arrow. For details, see Record Simulation Data.

Component-Based Modeling

MATLAB System block for including System objects in Simulink models

The MATLAB System block is a new block in the Simulink User-Defined Functions library. Use this block to create a Simulink block that includes a System object in your model. This capability is useful for including algorithms. For more information, see System Object Integration.

Variant Manager that manages all the variants in a model in one place

Variant Manager is a graphical tool that allows you to define and manage multiple variant configurations for Simulink models. For details, see Variant Management.

Improved componentization capabilities for modeling scheduling diagrams with root-level function-call inports

Models can now export functions by using root-level function-call Inport blocks. Such models can simulate in Normal, Accelerator, Rapid Accelerator, SIL, and PIL simulation modes.

Array of buses signal logging in model reference accelerator mode

You can log an array of buses signal in models using model reference accelerator mode or Normal mode. In R2013a, you could only log array of buses signals in Normal mode.

Ability to add, delete, and move input signals within Bus Creator block

The Bus Creator block includes **Up**, **Down**, **Add**, and **Remove** buttons to simplify organizing the input signals. When you add or remove signals, Simulink automatically updates the **Number of inputs** parameter of the Bus Creator block and maintains port connectivity in the model.

For details, see Reorder, Add, or Remove Signals in the Bus Creator block reference page.

Streamlined approach to migrating from Classic to Simplified initialization mode

Simplified initialization mode has increased flexibility, such as accepting an empty matrix ([ ]) for **Initial output**. These abilities make it easier to migrate custom libraries and models from classic to simplified initialization mode. For more information, see Conditional Subsystem Output Initialization.

Simplified display of sorted execution order

The display of sorted execution order is enhanced for subsystems and blocks such as function-call blocks, making it easier to understand the execution order in a model.

Enhanced model reference rebuild algorithm for MATLAB Function blocks

In conjunction with the **Model Configuration** > **Model Referencing** > **Rebuild** parameter, Simulink examines a set of known target dependencies when determining whether they have changed. R2013b adds another known target dependency: MATLAB files used by MATLAB function blocks.

For details, see Rebuild.

Simulation Analysis and Performance

LCC compiler included on Windows 64-bit platform for running simulations

The Windows 64-bit platform now includes `LCC-win64` as the default compiler for running simulations. You no longer have to install a separate compiler for simulation in Stateflow and Simulink. You can run simulations in Accelerator and Rapid Accelerator modes using this compiler. You can also build model reference simulation targets and accelerate MATLAB System block simulations.

Note: The `LCC-win64` compiler is not available as a general compiler for use with the command-line `MEX` in MATLAB. It is a C compiler only. You cannot use it for `SIL/PIL` modes.

`LCC-win64` is used only when another compiler is not configured in MATLAB. To build `MEX` files, you must install a compiler. See `http://www.mathworks.com/support/compilers/current_release/`.

Signal logging in Rapid Accelerator mode

You can log signals in Rapid Accelerator mode. Signal logging in Rapid Accelerator mode is similar to signal logging in Normal and Accelerator mode. During model development and testing, using Rapid Accelerator mode usually speeds up simulation with signal logging. For details, see Signal Logging in Rapid Accelerator Mode.

Performance Advisor checks for Rapid Accelerator mode and data store memory diagnostics

Performance Advisor checks have been enhanced to improve simulation and performance optimization.

A single check compares all four simulation modes before recommending the optimal choice:

- Normal
- Accelerator

- Rapid Accelerator
- Rapid Accelerator with up-to-date check off

There is another check that compares compiler optimizations if the selected simulation mode is Accelerator or Rapid Accelerator.

Also, you can now disable runtime diagnostics for data store memory to reduce runtime overhead and improve simulation speed. For more information, see Diagnostics.

Long long integers in simulation targets for faster simulation on Win64 machines

If your target hardware and your compiler support the C99 long long integer data type, you can select this data type for code generation and simulation. Using long long results in more efficient generated code that contains fewer cumbersome operations and multiword helper functions. This data type also provides more accurate simulation results for fixed-point and integer simulations. If you are using Microsoft Windows (64-bit), using long long improves performance for many workflows including using Accelerator mode and working with Stateflow software.

For more information, see the **Enable long long** and **Number of bits: long long** configuration parameters on the Hardware Implementation Pane.

At the command line, you can use the following new model parameters:

- `ProdLongLongMode`: Specifies that your C compiler supports the long long data type. Set this parameter to `'on'` to enable `ProdBitPerLongLong`.
- `ProdBitPerLongLong`: Describes the length in bits of the C long long data type supported by the production hardware.
- `TargetLongLongMode`: Specifies whether your C compiler supports the long long data type. Set this parameter to `'on'` to enable `TargetBitPerLongLong`.
- `TargetBitPerLongLong`: Describes the length in bits of the C long long data type supported by the hardware used to test generated code.

For more information, see Model Parameters.

Auto-insertion of rate transition block

Simulink can now automatically handle rate transitions for periodic tasks even if the Greatest Common Divisor (GCD) rate is not in the model. Previously, Simulink required

that a block with the GCD sample rate be present in the model to allow automatic insertion of the rate transition block. For more information, see Automatically handle rate transition for data transfer.

Compiled sample time for multi-rate blocks returns cell array of all sample times

For multi-rate blocks (including subsystems), Simulink now returns the compiled sample time for the block as a cell array of all the sample rates present in the block. Therefore, to query the `CompiledSampleTime` and determine if a subsystem is multi-rate, you no longer need to loop over all the blocks inside a subsystem or build up a list of sample times for the subsystem. The subsystem block `CompiledSampleTime` parameter now contains that information.

Previously, Simulink returned only the greatest common divisor (GCD) of all sample rates present in the block. This might cause a problem if the GCD rate did not exist in the model, thereby causing Simulink Coder to generate empty functions.

To obtain the complete list of sample times in a block, use the following command:

```
theBlockSampleTimes =
get_param(SubsystemBlock,'CompiledSampleTime');
```

This is more efficient than querying the sample time for each block and sorting the list.

For more information, see Sample Times in Subsystems.

Compatibility Considerations

Because Simulink now returns the `CompiledSampleTime` parameter as a cell array of pairs of doubles (instead of a pair of doubles), some compatibility issues can occur.

Consider a variable blkTs, which has been assigned the compiled sample time of a multi-rate block.

```
blkTs = get_param(block,'CompiledSampleTime');
```

Here are some examples in which the original code works only if blkTs is a pair of doubles and the block is a single-rate block:

- Example 1

  ```
  if isinf(blkTs(1))
  ```

```
    disp('found constant sample time')
end
```

Since blkTs is now a cell array, Simulink gives the following error message:

```
Undefined function 'isinf' for input arguments of type 'cell'
```
Instead, use this code, for which blkTs can be a cell array or a pair of doubles.

```
if isequal(blkTs, [inf,0])
    disp('found constant sample time')
end
```

- Example 2

```
if all(blkTs == [-1,-1])
    disp('found triggered sample time')
end
```

For the above example, since blkTs is now a cell array, Simulink gives the following error:

```
Undefined function 'eq' for input arguments of type 'cell'
```

Instead, use this code, for which blkTs can be a cell array or a pair of doubles.

```
if isequal(blkTs, [-1,-1])
    disp('found triggered sample time')
end
```

- Example 3

```
if (blkTs(1) == -1)
    disp('found a triggered context')
end
```

Again, since blkTs is now a cell array, Simulink gives the following error:

```
Undefined function 'eq' for input arguments of type 'cell'
```

Instead, use this code.

```
if ~iscell(blkTs)
    blkTs = {blkTs};
end
for idx = 1:length(blkTs)
    thisTs = blkTs{idx};
```

```
    if (thisTs(1) == -1)
        disp('found a triggered context')
    end
end
```

The above code checks for a triggered type sample time (triggered or async). In cases in which a block has constant sample time ([inf,0]) in addition to triggered or async or when a block has multiple async rates, this alternative detects sample times in all such cases.

Improvement to model reference parallel build check in Performance Advisor

The model reference parallel build check has been improved to take into account parallel build overhead time when estimating overall overhead time. For more information, see Check model reference parallel build.

Improved readability in Performance Advisor reports

Tables in the HTML report generated by Performance Advisor now appear in a new format for improved clarity and readability.

Simulation Data Inspector launch using simplot command

The `simplot` command now launches the Simulation Data Inspector. The `simplot` command is no longer supported and redirects to the Simulation Data Inspector.

Use the Simulation Data Inspector to inspect and compare signal data from simulations. For more information on using the Simulation Data Inspector, see Validate System Behavior.

Compatibility Considerations

In R2013b, the `simplot` command launches the Simulation Data Inspector, and the return arguments to the function are empty handles. In previous releases, the `simplot` command returned handles to the graphics figure.

Project and File Management

Impact analysis by exploring modified or selected files to find dependencies

In Simulink Project, use impact analysis to find the impact of changing particular files. You can investigate dependencies visually to explore the structure of your project. You can analyze selected or modified files to find their required and impacted files. Visualize changes and dependencies in the Impact graph.

Impact analysis can show you how a change will impact other files before making the change. For example:

- Investigate the potential impact of a change in requirements by finding the design files linked to the requirements document.
- Investigate change set impact by finding upstream and downstream dependencies of modified files before committing the changes. This can help you identify design and test files that might need modifications and find the tests you need to run.

You can label, open, or export the files found by impact analysis. You can use the impact analysis results to create reports or artifacts describing the impact of a change.

For details, see Perform Impact Analysis.

Option to export impact analysis results to the workspace, batch processing, or image files

In Simulink Project, after performing impact analysis, you can export the results to a workspace variable, to batch processing, or to image files. This enables further processing or archiving of impact analysis results.

For details, see Export Impact Results.

Identification of requirements documents during project dependency analysis

In Simulink Project, Dependency Analysis searches for requirements documents linked using the Requirements Management Interface. You can view linked requirements

documents in Simulink Project and navigate to and from the linked documents. You can only create or edit Requirements Management links if you have Simulink Verification and Validation.

Previously, you could find requirements documents only for a single model by generating a manifest. Now you can find requirements documents linked anywhere in your project.

For details, see Choose Files and Run Dependency Analysis.

Simplified label creation by dragging a label onto files in any view

In Simulink Project, add labels to files by dragging a label onto files. Create labels and categories of labels from any view. These features simplify label creation and application without any need to switch view. You can view project labels at the same time as project files or dependency analysis results.

Previously you could create new labels only at the Labels node, and you could apply a label to a single file only by opening a dialog box.

For details, see Create Labels and Add Labels to Files.

Shortcut renaming, grouping, and execution from any view using the Toolstrip

Simulink Project tools for managing shortcuts enable you to:

- Create shortcut groups to organize shortcuts by type, for example, to separate shortcuts for loading data, opening models, generating code, and running tests.
- Annotate shortcuts to make their purpose visible, without changing the file name or location of the script or model the shortcut points to, for example, to change a cryptic file name to a useful string for the shortcut name.
- Execute project shortcuts from any view in Simulink Project using the toolstrip.

These features simplify shortcut management, allowing you to use shortcuts while viewing your project files or dependency analysis results.

Previously you could view and execute shortcuts only at the Shortcuts node. Now you can find and execute shortcuts whenever they are needed in the projects workflow without switching the view.

For details, see Create Shortcuts to Frequent Tasks, Annotate Shortcuts to Use Meaningful Names, and Use Shortcuts to Find and Run Frequent Tasks.

Data Management

Streamlined selection of one or more signals for signal logging

In the Simulink Editor, you no longer need to open the Signal Properties dialog box to enable signal logging for a signal. Instead:

1   Select one or more signals.

2   Click the **Record** button arrow ▾ and click **Log/Unlog Selected Signals**.

Simplified modeling of single-precision designs

In R2013b, you can model single-precision designs more easily.

- There is now a model-wide setting that you can use to specify that Simulink use singles as the default type during data type propagation. See "New option to set default for underspecified data types" on page 9-16.

- Simulink avoids the use of double data types to achieve strict single design for operations between singles and integers. In previous releases, Fixed-Point Designer used double data types in intermediate calculations for higher precision. You might see a difference in numerical behavior of an operation between earlier releases and R2013b. See "Operations between singles and integer or fixed-point data types avoid use of doubles" on page 9-17.

- There is a new Model Advisor check that detects the presence of double data types in a model. See Identify questionable operations for strict single-precision design.

New option to set default for underspecified data types

There is now a model-wide setting to specify the data type to use if Simulink cannot infer the type of a signal during data type propagation. You can now choose to set the default value for underspecified data types to double or single for simulation and code generation. For embedded designs that target single-precision processors, set the **Default for underspecified data type** configuration parameter to `single` to avoid introducing double data types. For more information, see Default for underspecified data type.

Operations between singles and integer or fixed-point data types avoid use of doubles

Simulink now supports strict single-precision algorithms for mixed single and integer data types for cast and math operations. Operations, such as cast, multiplication and division, use single-precision math instead of introducing higher precision doubles for intermediate calculations in simulation and code generation. You no longer have to explicitly cast integers or fixed-point inputs of these operations to single precision. To detect the presence of double data types in your model to help you refine your mixed single and integer design, use the Model Advisor **Identify questionable operations for strict single-precision design** check.

Compatibility Considerations

In R2013b, for both simulation and code generation, Simulink avoids the use of double data types to achieve strict single design for operations between singles and integers. In previous releases, Simulink used double data types in intermediate calculations for higher precision. You might see a difference in the numerical behavior of an operation between earlier releases and R2013b.

For example, when the cast is from a fixed-point or integer data type to single or vice versa, the type for intermediate calculations can have a big impact on the numerical results. Consider:

* Input type: `ufix128_En127`
* Input value: 1.999999999254942 — Stored integer value is `(2^128 -2^100)`
* Output type: single

Release	Calculation performed by Fixed-Point Designer	Output Result	Design Goal
R2013b	`Y = single(2^-127) * single(2^128-2^100)` `= single(2^-127) * Inf`	Inf	Strict singles
Previous releases	`Y = single( double(2^-127) * double(2^128 - 2^100))` `= single(2^-127 * 3.402823656532e+38)`	2	Higher precision intermediate calculation

There is also a difference in the generated code. Previously, Fixed-Point Designer allowed the use of doubles in the generated code for a mixed multiplication that used single and integer types.

```
m_Y.Out1 = (real32_T)((real_T)m_U.In1*(real_T)m_U.In2);
```

In R2013b, it uses strict singles.

```
m_Y.Out1=(real32_T)m_U.In1*m_U.In2;
```

To revert to the numerical behavior of previous releases, insert explicit casting from integers to doubles for the inputs of these operations.

Connection status visualization and connection method customization for root inport mapping

The Root Inport Mapping dialog box has the following updates:

- The Root Inport Mapping dialog box now has connection status visualization. If the Root Inport Mapping status area lists warnings or failures, the software highlights the Inport block associated with the data. Warnings display as yellow Inport blocks outlined in orange, failures display as yellow Inport blocks outlined with bold red, and successes display as normal Inport blocks outlined with blue. Selecting the line item highlights the associated Inport block. For more information, see Understanding Mapping Results.

- The root inport mapping capability has a new function, getSlRootInportMap. This function provides a new connection method for custom mappings. Use this function when you have a mapping method that is similar to, but not exactly the same as, one of the existing Simulink root inport mapping methods.

Conversion of numeric variables into `Simulink.Parameter` objects

You can now convert a numeric variable into a `Simulink.Parameter` object using a single step.

```
myVar = 5; % Define numerical variable in base workspace
% Create data object and assign variable value to data object value
myObject = Simulink.Parameter(myVar);
```

Previously, you did this conversion using two steps.

```
myVar = 5; % Define numerical variable in base workspace
myObject = Simulink.Parameter; % Create data object
myObject.Value = myVar; % Assign variable value to data object value
```

Model Explorer search options summary hidden by default

To provide more space for displaying search results, the Model Explorer **Search Results** pane hides the summary of the search options (such as search criteria) that you used.

To view the search options summary, at the top of the **Search Results** pane, click the **Expand Search Results** button ☺.

Simulink.DualScaledParameter class

The new `Simulink.DualScaledParameter` class extends the capabilities of the `Simulink.Parameter` class. You can define a parameter object that stores two scaled values of the same physical value. Suppose you want to store temperature measurements as Fahrenheit or Celsius values. You can define a parameter that stores the temperature in either measurement scale with a computational method to convert between the dual-scaled values.

You can use `Simulink.DualScaledParameter` objects in your model for both simulation and code generation. The parameter computes the internal value before simulation or code generation via the computational method, which can be a first-order rational function. This offline computation results in leaner generated code.

For more information, see Simulink.DualScaledParameter.

Legacy data type specification functions return numeric objects

In previous releases, the following functions returned a MATLAB structure describing a fixed-point data type:

- float
- sfix
- sfrac
- sint
- ufix
- ufrac
- uint

Effective R2013b, they return a Simulink.NumericType object. If you have existing models that use these functions as parameters to dialog boxes, the models continue to run as before, and there is no need to change any model settings.

These functions do not offer full Data Type Assistant support. To benefit from this support, use fixdt instead.

Function	Return Value in Previous Releases — MATLAB structure	Return Value Effective R2013b — `NumericType`
`float('double')`	Class: 'DOUBLE'	DataTypeMode: 'Double'
`float('single')`	Class: 'SINGLE'	DataTypeMode: 'Single'
`sfix(16)`	Class: 'FIX' IsSigned: 1 MantBits: 16	DataTypeMode: 'Fixed-point: unspecified scaling' Signedness: 'Signed' WordLength: 16
`ufix(7)`	Class: 'FIX' IsSigned: 0 MantBits: 7	DataTypeMode: 'Fixed-point: unspecified scaling' Signedness: 'Unsigned' WordLength: 7
`sfrac(33,5)`	Class: 'FRAC' IsSigned: 1 MantBits: 33 GuardBits: 5	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Signed' WordLength: 33 FractionLength: 27
`ufrac(44)`	Class: 'FRAC' IsSigned: 0 MantBits: 44 GuardBits: 0	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Unsigned' WordLength: 44 FractionLength: 44
`sint(55)`	Class: 'INT' IsSigned: 1 MantBits: 55	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Signed' WordLength: 55 FractionLength: 0
`uint(77)`	Class: 'INT' IsSigned: 0 MantBits: 77	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Unsigned' WordLength: 77 FractionLength: 0

Compatibility Considerations

MATLAB Code

MATLAB code that depends on the return arguments of these functions being a structure with fields named `Class`, `MantBits` or `GuardBits`, no longer works correctly. Change the code to access the appropriate properties of a `NumericType` object, for example, `DataTypeMode`, `Signedness`, `WordLength`, `FractionLength`, `Slope`, and `Bias`.

C Code

Update C code that expects the data type of parameters to be a legacy structure to handle `NumericType` objects instead. For example, if you have S-functions that take legacy structures as parameters, update these S-functions to accept `NumericType` objects.

MAT-files

Effective in R2013b, if you open a Simulink model that uses a MAT-file that contains a data type specification created using the legacy functions, the model uses the same data types and behaves in the same way as in previous releases but Simulink generates a warning. To eliminate the warning, recreate the data type specifications using `NumericType` objects, and save the MAT-file.

You can use the `fixdtupdate` function to update a data type specified using the legacy structure to use a `NumericType`. For example, suppose you saved a data type specification in a MAT-file in a previous release as follows:

```
oldDataType = sfrac(16);
save myDataTypeSpecification oldDataType
```
Use `fixdtUpdate` to recreate the data type specification to use `NumericType`:

```
load DataTypeSpecification
fixdtUpdate(oldDataType)

ans =

  NumericType with properties:

      DataTypeMode: 'Fixed-point: binary point scaling'
        Signedness: 'Signed'
        WordLength: 16
    FractionLength: 15
            IsAlias: 0
          DataScope: 'Auto'
         HeaderFile: ''
        Description: ''
```
For more information, at the MATLAB command line, enter:

```
fixdtUpdate
```

Root Inport Mapping Error Messages

The error message handling has been improved for root import mapping at the consistency check phase. The Simulation Diagnostics Viewer now displays these error messages.

Root inport mapping example

The Using Simulink Mapping Modes in Custom Mapping Functions example shows how to use the getSlRootInportMap function to create a mapping object. This example uses a mapping mode similar to the Simulink block name mapping mode.

Connection to Educational Hardware

Ability to run models on target hardware from the Simulink toolbar

You can use the **Deploy to Hardware** or **Run** button on the Simulink toolbar to run models on target hardware.

To enable these buttons, first configure your model to use for Run on Target Hardware.

1 If you have not done so already, use supportPackageInstaller and install a Simulink support package.

2 In a model, select **Tools** > **Run on Target Hardware** > **Prepare to Run**.

    The Configuration Parameters dialog box opens and displays the Run on Target Hardware pane.

3 Set the **Target hardware** parameter to match your target hardware.

After configuring the model, you can use either button:

- To deploy the model, click **Deploy to Hardware**. The model runs as a standalone application on the target hardware.
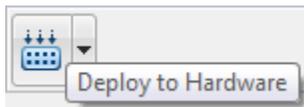
- To use External mode, set **Simulation mode** to `External`, and then click **Run**. In External mode, you can tune parameters and monitor a model running on your target hardware.
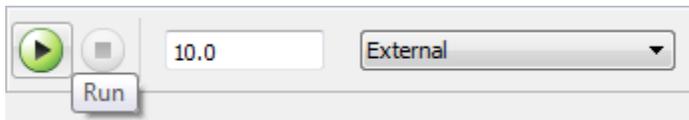
Note: This feature works only with Simulink support packages. The names of these support packages begin with "Simulink Support Package for..."

For more information, see What is "Run on Target Hardware"?.

Note: Some target hardware does not support External mode. For more information, consult the documentation for the specific target hardware.

Support for Arduino hardware available on Mac OS X

You can use *Simulink Support Package for Arduino Hardware* on Apple Mac OS X platform. This includes support for Arduino Mega 2560, Arduino Uno, Arduino Nano, and Arduino Ethernet Shield hardware.

To use this feature, install the *Simulink Support Package for Arduino Hardware*, as described in Install Support for Arduino Hardware.

Support for Arduino Ethernet Shield and Arduino Nano 3.0 hardware

You can use *Simulink Support Package for Arduino Hardware* with the Arduino Ethernet Shield hardware and Arduino Nano 3.0 hardware. The block library for Arduino Ethernet Shield hardware includes TCP/IP and UDP blocks that enable you to design network-enabled embedded systems.

To use this feature, install the *Simulink Support Package for Arduino Hardware*, as described in Install Support for Arduino Hardware.

The block library for the Arduino Ethernet Shield hardware includes the following blocks:

- Arduino TCP/IP Send and Arduino TCP/IP Receive enable TCP/IP communications with networked devices using an Ethernet port.
- Arduino UDP Send and Arduino UDP Receive enable UDP communications with networked devices using an Ethernet port.

For more information about this feature, see the Arduino Hardware topic.

Signal Management

Port number display to help resolve error messages

You can display the port numbers in a block that an error message highlights by hovering over the block input or output ports. For details, see Display Port Numbers When Addressing Errors.

Enforced bus diagnostic behavior

For models that use bus primitives, set the **Configuration Parameters** > **Diagnostics** > **Connectivity** > **Mux blocks used to create bus signals** diagnostic to `error`. Bus primitives include the Bus Creator, Bus Selector, and Bus Assignment blocks, as well as bus objects.

R2013b enforces setting the diagnostic to `error`. Benefits of setting this diagnostic to `error` include:

- Prevents introducing Mux/bus mixtures into your model. For information about the problems with that mixture, see Prevent Bus and Mux Mixtures.
- Improves handling of feedback loops.
- Supports important signal features, including:

    - Nonzero initialization of bus signals
    - Bus support for blocks such as Constant, Data Store Memory, From File, From Workspace, To File, and To Workspace
    - Signal label propagation enhancements
    - Arrays of buses

Compatibility Considerations

When you compile (update or simulate) a model in R2013b, Simulink analyzes the model to determine the extent to which the model uses Mux blocks to create bus signals. The table describes how Simulink handles different kinds of models at compile time, in relationship to the diagnostic.

Model Configuration	Simulink Actions at Compile Time	Your Required Actions
No bus primitives or muxes that involve different data types	Displays no warning, error, or upgrade message related to this diagnostic Sets diagnostic to `error` when you save the model	None
Bus primitives and diagnostic is set to `error`	None	None
No bus primitives, but one or more mux with different data types	Displays an error message prompting you to launch the Upgrade Advisor	Run the Upgrade Advisor and compile your model.
Bus primitives and diagnostic set to `Warning` or `None`	Displays an error message prompting you to launch the Upgrade Advisor	Run the Upgrade Advisor and set the diagnostic to `error`.

Block Enhancements

Improved performance of LUT block intermediate calculations

Blocks in the Lookup Tables library have a new internal rule for fixed-point data types to enable faster hardware instructions for intermediate calculations (with the exception of the Direct Lookup Table (n-D), Prelookup and Lookup Table Dynamic blocks). To use this new rule, select `Speed` for the **Internal Rule Priority** parameter in the dialog box. Select `Precision` to use the internal rule in R2013a.

Name changes that unify storage class parameters

The following parameters have been renamed to unify the names of storage class parameters.

Old Name	New Name
RTWStateStorageClass	StateStorageClass
CodeGenStateStorageClass	StateStorageClass

The blocks affected by this name change are Delay, Unit Delay, Memory, Discrete State-Space, Discrete Zero-Pole, Discrete Filter, Discrete Transfer Function, Data Store Memory, and Discrete-Time Integrator.

Compatibility Considerations

Simulink will not support the old parameter names in a future release. If you use the old parameter names in your code to programmatically set parameter values, replace them with the new names.

Warnings when using old parameter names with spaces

Parameter names that include spaces (including block type names) from Simulink Version 1.3 (1994) and earlier now warn. Do not use parameter names with spaces.

Compatibility Considerations

Old parameter names that include spaces (including block type names) from Simulink Version 1.3 (1994) and earlier now cause a warning if you use them with get_param,

`set_param`, `find_system`, or `add_block`. Messages direct you to the new parameter name if it exists.

Strictly monotonically increasing time values on Repeating Sequence block

The **Time values** parameter in the Repeating Sequence block can take only strictly monotonically increasing values. In R2013a, this parameter could take duplicate time values. Run the Upgrade Advisor check "Check model for known block upgrade issues" to identify this issue in your model.

pow function in Math function block that supports Code Replacement Library (CRL)

The Math Function block supports the Code Replacement Library for the `pow` function, which is useful if you use code generation for models that use this function. For more information, see the Math Function block.

Continuous Linear Block improvements, such as diagnostics, readability, and stricter checks

The Continuous library blocks State-Space, Transfer Fcn, and Zero-Pole have been enhanced as follows:

- Improved error diagnostics.
- More readable generated code for large matrices.
- Stricter checks for changes made to tunable parameters at run time.
- Unused tunable parameters removed from generated code.
- Ability for the State-Space block to act as a source block. This change enables the modeling of autonomous linear state-space systems.
- The following changes in the Transfer Fcn block:
  - Computation of time-domain realization has been enhanced for better performance and error diagnostics
  - Support of sparse Jacobian computation for reduced memory usage, better implicit solver support, and improved structural analysis

MATLAB Function Blocks

Code generation for Statistics Toolbox and Phased Array System Toolbox

Code generation now supports more than 100 Statistics Toolbox™ functions. For implementation details, see Statistics Toolbox Functions.

Code generation now supports most of the Phased Array System Toolbox™ functions and System objects. For implementation details, see Phased Array System Toolbox Functions and Phased Array System Toolbox System Objects.

Toolbox functions for code generation

For implementation details, see Functions Supported for C/C++ Code Generation — Alphabetical List.

Data Type Functions

- narginchk

Programming Utilities

- mfilename

Specialized Math

- psi

Computer Vision System Toolbox Classes and Functions

- extractFeatures
- detectSURFFeatures
- disparity
- detectMSERFeatures
- detectFASTFeatures
- vision.CascadeObjectDetector
- vision.PointTracker
- vision.PeopleDetector

- cornerPoints
- MSERRegions
- SURFPoints

External C library integration using `coder.ExternalDependency`

You can define the interface to external code using the new `coder.ExternalDependency` class. Methods of this class update the compile and build information required to integrate the external code with MATLAB code. In your MATLAB code, you can call the external code without needing to update build information. See coder.ExternalDependency.

Updating build information using `coder.updateBuildInfo`

You can use the new function `coder.updateBuildInfo` to update build information. For example:

```
coder.updateBuildInfo('addLinkFlags','/STACK:1000000');
```
adds a stack size option to the linker command line. See coder.updateBuildInfo.

Conversion of MATLAB expressions into C constants using `coder.const`

You can use the new function `coder.const` to convert expressions and function calls to constants at compile time. See coder.const.

Highlighting of constant function arguments in the compilation report

The compilation report now highlights constant function arguments and displays them in a distinct color. You can display the constant argument data type and value by placing the cursor over the highlighted argument. You can export the constant argument value to the base workspace where you can display detailed information about the argument.

For more information, see Viewing Variables in Your MATLAB Code.

`coder.target` syntax change

The new syntax for `coder.target` is:

```
tf = coder.target('target')
```

For example, `coder.target('MATLAB')` returns true when code is running in MATLAB. See coder.target.

You can use the old syntax, but consider changing to the new syntax. The old syntax will be removed in a future release.

LCC compiler included on Windows 64-bit platform for running simulations

The Windows 64-bit platform now includes `LCC-win64` as the default compiler for running simulations. You no longer have to install a separate compiler for simulation of MATLAB Function blocks.

`LCC-win64` is used only when another compiler is not configured in MATLAB.

Modeling Guidelines

Modeling guidelines for high-integrity systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0024: Inport interface definition
- hisl_0025: Design min/max specification of input interfaces
- hisl_0026: Design min/max specification of output interfaces
- hisl_0027: Usage of Signed Square Root blocks
- hisl_0028: Usage of Reciprocal Square Root blocks

Model Advisor

Collapsible content within Model Advisor reports

The Model Advisor report now collapses the results, making it easier to navigate through the report.

Reorganization of Model Advisor checks

Checks previously provided with Simulink in the Model Advisor Embedded Coder folder are now available with either Simulink Coder or Embedded Coder. For a list of checks available with each product, see:

- Simulink Coder Checks
- Embedded Coder Checks

Check for strict single precision usage

The new Identify questionable operations for strict single-precision design check identifies blocks that introduce double-precision operations. Use this check to help you refine your strict single design.

R2013a

Version: 8.1

New Features

Bug Fixes

Compatibility Considerations

New Simulink Editor

Reordering of tabs in tabbed windows

You can rearrange the order of tabs in a Simulink Editor window. For example, if you have opened several subsystems, to make it easier to access the tab for the last subsystem that you opened, you could select that tab and drag it to make it the first tab on the left.

Scalable vector graphics for mask icons

Images in `.svg` format can be used with the `image` command for creating block mask images in Simulink.

Simulation Stepper Default Value Change

The default value of the **Interval between stored back steps** stepping option is now 10. In the previous release, this value was 1.

Component-Based Modeling

Direct active variant control via logical expressions

Variant control can accept a variant object or any condition expression. Defining variant objects for use in modeling variants requires saving variant objects external to a model file. Using direct expressions in variant control reduces complexity. For more information, see Variant Systems.

Live update for variant systems and commented-out blocks

You can control active variants by setting variant controls. When you modify active variants, the display refreshes automatically; you do not need to use Update Diagram. You can also comment out blocks in your model to exclude them from simulation. For more information, see Variant Systems.

Masking of linked library blocks

You can now create masks on linked blocks. Masking a linked block (one that already has a mask) creates a stack of masks. Simulink libraries can contain blocks that become library links when copied to a model. Masking such blocks previously involved wrapping them inside a subsystem and creating a mask on that subsystem. Masking linked blocks instead uses less memory and management overhead. For more information, see Masking Linked Blocks.

Target profiling for concurrent execution to visualize task execution times and task-to-core assignment

A new pane, Profile Report on the Concurrent Execution dialog box, enables the visualization of task execution times and task-to-core affinitization. You can profile using Simulink Coder (GRT) and Embedded Coder (ERT) targets. For more information, see Profile and Evaluate.

The `sldemo_concurrent_execution` example has been updated to reflect the use of this capability.

Incremental block-to-task mapping workflow support enabled by automatic block-to-task assignment for multicore execution on embedded targets

This support allows for the partial mapping of blocks to tasks, allowing you to specify task assignments only for the blocks you are interested in. For more information, see Analyze Baseline.

With this change, the **Map blocks to tasks** pane no longer has a **Get Default Configuration** button.

PIL and SIL modes for concurrent execution

The following simulation modes are now supported for concurrent execution:

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

Parameterized task periods for concurrent execution

You can now parameterize task periods for concurrent execution using variables from the MATLAB base workspace.

Relaxed configuration parameter setting requirements

Configuration requirements for model referencing have been removed. The following parameters no longer need to be the same for top and referenced models:

- **Templates** > **Target operating system**
- **Solver** > **Allow tasks to execute concurrently on target**

Connection to Educational Hardware

Support for Gumstix Overo hardware

Run Simulink models on Gumstix Overo hardware. Tune parameter values in the model, and receive data from the model, while it is running on Gumstix Overo hardware.

Use the **Simulink Support Package for Gumstix Overo Hardware** block library to access Gumstix Overo peripherals:

· Overo UDP Receive and Overo UDP Send — Communicate with networked devices using an Ethernet port.

· Overo ALSA Audio Capture — Capture audio from sound card using ALSA

· Overo ALSA Audio Playback — Send audio to sound card for playback using ALSA

· Overo V4L2 Video Capture — Capture video from USB camera using V4L2

· Overo SDL Video Display — Display video using SDL

· Overo GPIO Read and Overo GPIO Write — Communicate with external devices using GPIO pins. The blocks provide diagrams that help you locate specific GPIO pins.

· Overo LED — Illuminate built-in LEDs on your target hardware. The block provides a diagram that helps you locate specific LEDs.

· Overo eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these capabilities and the block library, enter targetinstaller in a MATLAB Command Window. Then, use Support Package Installer to install the support package for Gumstix Overo hardware. For more information, see the Gumstix Overo topic.

After installing the support package, you can open the block library by entering `overolib` in the MATLAB Command Window. The Simulink Support Package for Gumstix Overo Hardware block library is also available in the Simulink Library Browser.

Support for Raspberry Pi hardware

Run Simulink models on Raspberry Pi hardware. Tune parameter values in the model, and receive data from the model, while it is running on Raspberry Pi hardware.

Use the **Simulink Support Package for Raspberry Pi Hardware** block library to access Raspberry Pi peripherals:

- Raspberry Pi UDP Receive and Raspberry Pi UDP Send — Communicate with networked devices using an Ethernet port.
- Raspberry Pi ALSA Audio Capture — Capture audio from sound card using ALSA
- Raspberry Pi ALSA Audio Playback — Send audio to sound card for playback using ALSA
- Raspberry Pi V4L2 Video Capture — Capture video from USB camera using V4L2
- Raspberry Pi SDL Video Display — Display video using SDL
- Raspberry Pi GPIO Read and Raspberry Pi GPIO Write — Communicate with external devices using GPIO pins. The blocks provide diagrams that help you locate specific GPIO pins.
- Raspberry Pi LED — Illuminate built-in LEDs on your target hardware. The block provides a diagram that helps you locate specific LEDs.
- Raspberry Pi eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these capabilities and the block library, enter targetinstaller in a MATLAB Command Window. Then, use Support Package Installer to install the support package for Raspberry Pi hardware. For more information, see the Raspberry Pi topic.

After installing the support package, you can open the block library by entering `pilib` in the MATLAB Command Window. The Simulink Support Package for Raspberry Pi Hardware block library is also available in the Simulink Library Browser.

Blocks for GPIO, LED, and eSpeak Text to Speech on BeagleBoard

The block libraries for BeagleBoard hardware include four new blocks:

- BeagleBoard GPIO Read and BeagleBoard GPIO Write — Communicate with external devices using GPIO pins.
- BeagleBoard LED — Illuminate built-in LEDs on your target hardware.
- BeagleBoard eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these blocks, enter targetinstaller in a MATLAB Command Window. Then, use Support Package Installer to install the support package for BeagleBoard hardware. For more information, see the BeagleBoard topic.

After installing the support package, you can open the updated block library. In a MATLAB Command Window, enter `beagleboardlib`. You can also access these block libraries through the Simulink Library Browser.

Blocks for GPIO, LED, and eSpeak Text to Speech on PandaBoard

The block libraries for PandaBoard hardware include four new blocks:

- PandaBoard GPIO Read and PandaBoard GPIO Write — Communicate with external devices using GPIO pins.
- PandaBoard LED — Illuminate built-in LEDs on your target hardware.
- PandaBoard eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these blocks, enter `supportPackageInstaller` in a MATLAB Command Window. Then, use Support Package Installer to install the support package for PandaBoard hardware. For more information, see the PandaBoard topic.

After installing the support package, you can open the updated block library. In a MATLAB Command Window, enter `pandaboardlib`. You can also access these block libraries through the Simulink Library Browser.

Blocks for Compass and IR Receiver sensors on LEGO MINDSTORMS NXT

The block library for LEGO MINDSTORMS NXT hardware includes two new blocks:

- LEGO MINDSTORMS NXT Compass Sensor — Read the magnetic heading of the compass sensor.
- LEGO MINDSTORMS NXT IR Receiver Sensor — Receive IR signals from of a LEGO Power Functions IR Speed Remote Control.

To get these blocks, in a MATLAB Command Window, enter targetinstaller. Then, use Support Package Installer to install the support package for LEGO MINDSTORMS NXT hardware. For more information, see the LEGO MINDSTORMS NXT topic.

After installing the support package, you can open the updated block library. In a MATLAB Command Window, enter `legonxtlib`. The block library is also available in the Simulink Library Browser.

Project and File Management

Simplified scripting interface for automating Simulink Project tasks

Simulink Projects provide a new API with shorter commands for automating project tasks with file and label management.

See Simulink Projects for links to project functions.

Compatibility Considerations

The new Simulink Projects API replaces the class `Simulink.ModelManagement.Project.CurrentProject` and its methods. `Simulink.ModelManagement.Project.CurrentProject` will be removed in a future release. Instead, use simulinkproject and related functions for project manipulation.

Option to use elements from multiple templates when creating a new project

When you create a new project, you can select multiple templates to apply. You can select templates directly on the New Project menu.

See Use Templates to Create Standard Project Settings.

Saving and reloading of dependency analysis results

The Simulink Project Tool remembers the results of previous dependency analysis and saves the results with your project. You can view your previous results without having to run time-consuming analysis again. You can also save and reload previous results.

See Analyze Project Dependencies.

Robust loading of projects with conflicted metadata project definition files

The Simulink Project Tool now has tolerance for loading projects with conflicts in metadata project definition files. You can load the conflicted project and resolve the conflicts. In previous releases you could not load a project with conflicts in the metadata.

New project preferences to control logging and warnings

Simulink Project Tool has a new Preferences dialog where you can control options for logging and warnings.

Data Management

Fixed-Point Advisor support for model reference

The Fixed-Point Advisor now performs checks on referenced models. It checks the entire model reference hierarchy against fixed-point guidelines. The Advisor also provides guidance about model configuration settings and unsupported blocks to help you prepare your model for conversion to fixed point.

Arrays of buses loading and logging

You can log array of buses signal data using signal logging. For details, see step 5 in Set Up a Model to Use Arrays of Buses.

You can load array of buses data to a root Inport block. For details, see Import Array of Buses Data.

Root Inport Mapping tool changes

The following are changes to the Root Inport Mapping tool:

- The Root Inport Mapping tool can now import the following additional data formats from a MAT-file:

  - Array of buses
  - Asynchronous function-call signals

- A new button, **Map Signals**, has been added to the **Mapping Mode** section of the tool. Use this button to map the data to the root-level ports. Then, use the **Apply** or **OK** buttons to commit the changes to the model.

- The Status area of the tool has been visually updated to better display the mapping status.

- A new function, getInputString, enables you to create a comma-separated list of variables to be mapped.

For more information, see Import and Map Data to Root-Level Inports.

New Root Inport Mapping Examples

The following examples show how to use the Root Inport Mapping dialog box:

- Converting Test Harness Model to Harness-Free Model

  The `slexAutotransRootInportsExample` example shows how to convert a harness model using Signal Builder block as an input to a harness-free model with root inports.

- Custom Mapping for External Inputs of a Model

  This example, available from **Simulink** > **Simulink Examples** > **Modeling Features**, shows how to create a custom mapping function for mapping data to root-level input ports.

Level-1 data classes not supported

Simulink no longer supports level-1 data classes. Extend Simulink data classes using MATLAB class syntax instead.

For more information, see Define Data Classes.

To upgrade your level-1 data classes, see Upgrade Level-1 Data Classes.

Compatibility Considerations

When you upgrade your level-1 data classes, the way that MATLAB code is generated and model files are loaded remains the same. However, you may encounter errors if your code includes the following capabilities specific to level-1 data classes:

- Inexact property names such as `a.datatype` instead of the stricter `a.DataType`.

- Vector matrix containing `Simulink.Parameter` and `Simulink.Signal` data objects. Previously, using level-1 data classes, you could define a vector matrix `v` as follows:

  ```
  a = Simulink.Signal;
  b = Simulink.Parameter;
  v = [a b];
  ```
  Such mixed vector matrices are no longer supported.

In these cases, modify your code to replace these capabilities with those supported by MATLAB class syntax. For more information on how to make these replacements, see Begin Using Object-Oriented Programming in MATLAB documentation.

Simulink data type classes do not support inexact enumerated property value matching

Previously, Simulink data type classes permitted partial enumerated property value matching and did not enforce case sensitivity. For example, after creating a `Simulink.NumericType` data type object

```
a = Simulink.NumericType;
```
you could set the value of property `DataTypeMode` of the object by using one of the following commands:

- Partial matching of enumerated property value:

  ```
  a.DataTypeMode = 's';
  ```
  Here, `'s'` is equivalent to `'Single'`.

- Case-insensitive matching of enumerated property value:

  ```
  a.DataTypeMode = 'Fixed-Point: binary point scaling';
  ```
  Here, `'Fixed-Point: binary point scaling'` is equivalent to `'Fixed-point: binary point scaling'`.

Now, Simulink data type classes do not permit partial or case-insensitive matches of enumerated property values.

Compatibility Considerations

You may encounter errors or warnings if your code relies on setting enumerated property values of data type objects using inexact matches. In these case, replace your code so that these property values are set using exact matches. For example, after creating a `Simulink.NumericType` data type object

```
b = Simulink.NumericType;
```
set the value of property `DataTypeMode` using the following command:

```
b.DataTypeMode = 'Single';
```

Tip: Tab completion works with enumerated properties. For example, if you enter a property name followed by an equal sign, MATLAB pops up a selection box with a list of values for that property.

Simulation Analysis and Performance

Simulation Performance Advisor report that shows both check results and actions taken

Performance Advisor HTML report now include actions in addition to checks. For more information, see View Performance Advisor Reports.

Improved simulation performance when stepping back is enabled

The performance of stepping back using the Simulation Stepper has been improved. For more information on the Simulation Stepper, see Simulation Stepping.

Simulation Data Inspector run-configuration options for names and placement in run list

When managing many runs in the Simulation Data Inspector, you can specify whether to add new runs at the top or bottom of the Signal Browser table. In addition, you can customize automatic naming of new runs added to the Simulation Data Inspector. For more information, see Run Management Configuration.

Arrays of buses displayed in Simulation Data Inspector

The Simulation Data Inspector records logged arrays of buses. After recording an array of buses, the Simulation Data Inspector can display this data in a hierarchical format.

Simulation Data Inspector overwrite run specification

In the Simulation Data Inspector, you can overwrite a previously recorded run with a new run using the **Overwrite Run** button. Overwriting a run eliminates a large accumulation of runs when you are establishing a baseline run for comparing simulation runs. When you overwrite a run, signal selection and color are retained.

Signal Management

Referenced models sample times

You can use one or more variable sample times in a referenced model. You can include blocks with variable sample times, such as the Pulse Generator block, in a referenced model in Normal or Accelerator mode. The Sample Time Legend reflects this new capability with the following:

· Text label `Variable` in the **Description** column.
· Display of hierarchical structure of the value in the reference model hierarchy in the **Value** column.

For more information, see Designate Sample Times.

Opening a model that contains referenced models with variable sample times generates errors in releases prior to R2013a.

Compatibility Considerations

If you want to use an S-function block that contains one variable sample time in an R2013a referenced model, recompile the S-function code in the R2013a environment first. Otherwise, compiling this model reference hierarchy in the R2013a environment generates errors.

Triggered subsystem sample times

The Sample Time Legend now displays the source of triggered subsystem sample times. The block area of a model now has annotations that show the triggered sample time and the sample time index number. For more information, see Designate Sample Times

Simulation of variable-size scalar signals

Previously, a model that used a variable-size scalar signal (width equals 1) would cause an error during a model update. You can now simulate a model with a variable-size scalar signal.

Block Enhancements

CORDIC approximation method for atan2 function of Trigonometric Function block

The Trigonometric Function block now supports the CORDIC approximation method for computing the output of the `atan2` function. For more information, see Trigonometric Function.

Product and Gain blocks support Basic Linear Algebra Subprogram (BLAS) library

The Product and Gain blocks now support the Basic Linear Algebra Subprogram (BLAS) library. The BLAS library is a library of external linear algebra routines optimized for fast computation of large matrix operations. Whenever possible, the blocks use BLAS library routines to increase simulation speed.

Performance Advisor check for Delay block circular buffer setting

To improve simulation, the Performance Advisor checks that each Delay block in the model uses the appropriate buffer type. By default, the block uses an array buffer (the **Use circular buffer for state** option is not selected). However, when the delay length is large, a circular buffer can improve execution speed by keeping the number of copy operations constant. For more information, see Check Delay block circular buffer setting.

MATLAB Function Blocks

Masking of MATLAB Function blocks to customize appearance, parameters, and documentation

In R2013a, you can mask a MATLAB Function block directly. In previous releases, you had to place the MATLAB Function Block in a subsystem, and then mask that subsystem.

Compatibility Considerations

In R2013a, MATLAB scripts or functions that rely on the `MaskType` property of MATLAB Function blocks need to be updated. For example, `get_param(handle_to_block, 'MaskType')` or `get_param(handle_to_block, 'MaskDescription')` now returns an empty value. Using `find_system(block_diagram_root,'SFBlockType','MATLAB Function')` returns all MATLAB Function blocks. Using `get_param(handle_to_block,'SFBlockType')` returns `MATLAB Function`. Do not create masks with Mask Type `Stateflow`, because the behavior is unpredictable.

File I/O function support

The following file I/O functions are now supported for code generation:

- fclose
- fopen
- fprintf

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

Support for nonpersistent handle objects

You can now generate code for local variables that contain references to handle objects or System objects. In previous releases, generating code for these objects was limited to objects assigned to persistent variables.

Include custom C header files from MATLAB code

The `coder.cinclude` function allows you to specify in your MATLAB code which custom C header files to include in the generated C code. Each header file that you specify using `coder.cinclude` is included in every C/C++ file generated from your MATLAB code. You can specify whether the `#include` statement uses double quotes for application header files or angle brackets for system header files in the generated code.

For example, the following code for function `foo` specifies to include the application header file `mystruct.h` in the generated code using double quotes.

```
function y = foo(x1, x2)
%#codegen
coder.cinclude('mystruct.h');

...
```

For more information, see coder.cinclude.

Load from MAT-files

MATLAB Coder now supports a subset of the load function for loading run-time values from a MAT-file while running a MEX function. It also provides a new function, `coder.load`, for loading compile-time constants when generating MEX or standalone code. This support facilitates code generation from MATLAB code that uses `load` to load constants into a function. You no longer have to manually type in constants that were stored in a MAT-file.

To view implementation details for the `load` function, see Functions Supported for Code Generation — Alphabetical List.

For more information, see coder.load.

`coder.opaque` function enhancements

When you use `coder.opaque` to declare a variable in the generated C code, you can now also specify the header file that defines the type of the variable. Specifying the location of the header file helps to avoid compilation errors because the code generation software can find the type definition more easily.

You can now compare `coder.opaque` variables of the same type. This capability helps you verify, for example, whether an `fopen` command succeeded.

```
null = coder.opaque('FILE*','NULL','HeaderFile','stdio.h');
ftmp = null;
ftmp = coder.ceval('fopen',fname,permission);
if ftmp == null
  % Error - file open failed
end
```

For more information, see coder.opaque.

Complex trigonometric functions

You can now use complex acosD, acotD, acscD, asecD, asinD, atanD, cosD, cscD, cotD, secD, sinD, and tanD functions with the MATLAB Function block.

Support for integers in number theory functions

Code generation supports integer inputs for the following number theory functions:

- cumprod
- cumsum
- factor
- factorial
- gcd
- isprime
- lcm
- median
- mode
- nchoosek
- nextpow2
- primes
- prod

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

Enhanced support for class property initial values

If you initialize a class property, you can now assign a different type to the property when you use the class. For example, class `foo` has a property `prop1` of type `double`.

```
classdef foo %#codegen
  properties
    prop1= 0;
  end
  methods
    ...
  end
end
```
Function `bar` assigns a different type to `prop1`.

```
fumction bar %#codegen
  f=foo;
  f.prop1=single(0);
  ...
```
In R2013a, code generation ignores the initial property definition and uses the reassigned type. In previous releases, code generation did not support this reassignment and failed.

Compatibility Considerations

In previous releases, if the reassigned property had the same type as the initial value but a different size, the property became variable-size in the generated code. In R2013a, code generation uses the size of the reassigned property, and the size is fixed. If you have existing MATLAB code that relies on the property being variable-size, you cannot generate code for this code in R2013a. To fix this issue, do not initialize the property in the property definition block.

For example, you can no longer generate code for the following function `bar`.

Class `foo` has a property `prop1` which is a scalar `double`.

```
classdef foo %#codegen
  properties
    prop1= 0;
  end
  methods
    ...
  end
```

```
end
```
Function `bar` changes the size of `prop1`.

```
fumction bar %#codegen
  f=foo;
  f.prop1=[1 2 3];
  % Use f
  disp(f.prop1);
  f.prop1=[1 2 3 4 5 6 ];
```

Default use of Basic Linear Algebra Subprograms (BLAS) Libraries

Code generated for MATLAB Function blocks now uses BLAS libraries whenever they are available. There is no longer an option to turn off the use of these libraries.

Compatibility Considerations

If existing configuration settings disable BLAS, code generation now ignores these settings.

New toolbox functions supported for code generation

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

Bitwise Operation Functions

- flintmax

Computer Vision System Toolbox Classes and Functions

- binaryFeatures
- insertMarker
- insertShape

Data File and Management Functions

- computer
- fclose
- fopen

- fprintf
- load

Image Processing Toolbox Functions

- conndef
- imcomplement
- imfill
- imhmax
- imhmin
- imreconstruct
- imregionalmax
- imregionalmin
- iptcheckconn
- padarray

Interpolation and Computational Geometry

- interp2

MATLAB Desktop Environment

- ismac
- ispc
- isunix

String Functions

- strfind
- strrep

Function being removed

The emlmex function has been removed.

Compatibility Considerations

The emlmex function generates an error in R2013a.

Modeling Guidelines

Modeling Guidelines for High-Integrity Systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- himl_0001: Usage of standardized function headers
- himl_0002: Strong data typing (MATLAB Function block boundary)
- himl_0003: Limitation of MATLAB Function complexity

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow

The MathWorks Automotive Advisory Board (MAAB) working group created Version 3.0 of the MAAB Guidelines Using MATLAB, Simulink, and Stateflow. See MAAB Control Algorithm Modeling for the MathWorks presentation of the guidelines.

Model Advisor

Model Advisor checks reorganized in a future release

In a future release, the Model Advisor checks will be reorganized.

Compatibility Considerations

MathWorks will review the products and licenses required to run the Model Advisor checks.

Model Advisor navigation between Upgrade Advisor, Performance Advisor, and Code Generation Advisor

In the Model Advisor window, you can select:

- **Code Generation Advisor** to help configure your model to meet code generation objectives.
- **Upgrade Advisor** to help upgrade models.
- **Performance Advisor** to help improve the simulation performance of your model.

Report

Single file HTML

Images in the Model Advisor `html` report are now embedded within the file, making it easier to export and maintain the report. Previously, the icon images were stored in separate files.

Format

The Model Advisor report format now uses indenting and *italics* to make it easier to assess the results of an analysis.

Preferences dialog box

From the Model Editor, select **Analysis** > **Model Advisor** > **Preferences** to open the Model Advisor Preferences dialog box. Alternately, in the Model Advisor window, select **Settings** > **Preferences**.

- From **Default Mode**, select `Model Advisor` or `Model Advisor Dashboard` to specify the interface that you want to use.
- Select **Show Accordian** for entry points to other Advisors from the Model Advisor window.

To display additional information about the checks in the Model Advisor window, select:

- **Show By Product Folder** for checks available for each product.
- **Show By Task Folder** for checks related to specific tasks.
- **Show Source tab** for check Title, TitleID, and location of the MATLAB source code for the check.
- **Show Exclusion tab** for checks that are excluded from the Model Advisor analysis.

By Product folder not displayed

By default, in the Model Advisor window, the **By Product** folder is not displayed. To display checks in the **By Product** folder, in the Model Advisor window, select **Settings > Preferences**. In the Model Advisor Preferences dialog box, select **Show By Product Folder**.

R2012b

Version: 8.0

New Features

Bug Fixes

Compatibility Considerations

New Simulink Editor

Tabbed windows and automatic window reuse to minimize window clutter

By default, the Simulink Editor uses:

- One window to display a model and its subsystems
- The same tab for each system that you open in a model

For example, if you open the `f14` model and then open the `Controller` subsystem, the Simulink Editor reuses the window and tab.
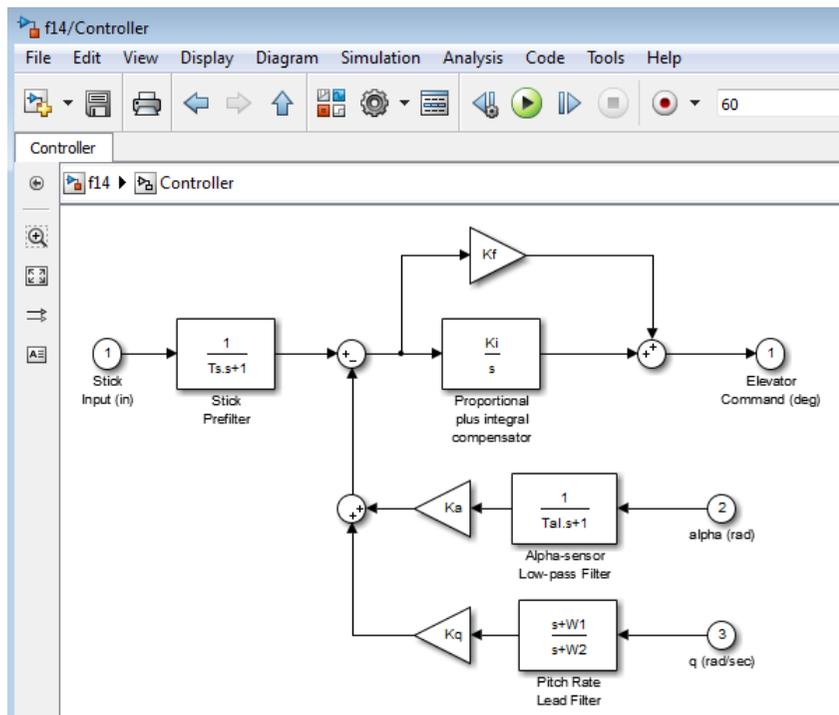
Window reuse and tabs:

- Conserve desktop space

- Keep the display of systems in a model together
- Provide easy navigation between systems in a model

To override the default window reuse behavior for a specific subsystem, right-click the subsystem and select either **Open in New Tab** or **Open in New Window**.

For more information, see Window Management.

Compatibility Considerations

The window display behavior for models created before R2012b remains the same in R2012b as it was in earlier releases. For example, if opening a model in an earlier release (such as R2011a) opened three Simulink Editor windows, then when you open that same model in R2012b, three Simulink Editor windows also open.

However, if you open an earlier model that has a *callback* that opens a subsystem, by default the subsystem opens in the same Simulink Editor window that is used for the model. If you want the callback to open a separate window for the subsystem, include an open_system call that uses the new `window` argument.

Smart signal routing that determines the simplest signal line path without overlapping blocks

When you draw lines to connect blocks, the Simulink Editor automatically routes the line to avoid other blocks and to minimize diagram clutter. You can manually modify the routing.

For more information, see Connect Blocks.

Explorer bar to help with navigating through a model

The Explorer bar in the Simulink Editor provides a breadcrumb that shows the nested path for the currently open system.
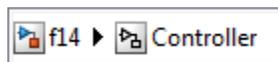
Select a system in the breadcrumb to open that system in the model window. If you click in the Explorer bar whitespace, you can edit the hierarchy. Also, the down arrow at the right side of the Explorer bar provides a model display history.

Simulation stepper to simulate and rewind a model one step at a time

With the new simulation stepper, you can:

- Step forward and back in time during a simulation
- Set time breakpoints
- Set conditional breakpoints on a scalar signal

For more information, see Simulation Stepping.

Ability to comment out blocks

Comment out blocks in your model if you want to exclude them during simulation. To exclude a block, right-click the selected block and select **Comment out**.

Commenting out blocks can be useful for several tasks, including to:

- Incrementally test parts of a model under development
- Debug a model without having to delete and restore blocks between simulation runs
- Test and verify the effects of certain model blocks on simulation results
- Improve simulation performance

Subsystem badges to identify and look under masked subsystems

Identify masked subsystems by the badge that appears in the lower-left corner of the mask (⬇). Click this badge to open the mask.

Note: The badge does not appear for masks of library links if the library is locked or its `LockLinksToLibrary` property is set to `true`.

Reorganized menu to fit common Model-Based Design workflow

The new Simulink Editor provides a top-level menu structure that reflects the steps of the model-based design process that you perform in the Simulink environment. The process is iterative and involves using different menus in different orders. The following table indicates the main menus associated with each Model-Based Design step.

Model-Based Design Step	Corresponding Menus
Build the Simulink block diagram.	• **File** • **Edit** • **View** • **Diagram**
Run the simulation.	**Simulation**
Validate the simulation results.	• **View** • **Display** • **Analysis**

The **Help** and **Tools** menus provide information and tools that apply throughout the steps of the process.

The **Code** menu provides options relating to code generation.

For more information, see Simulink Editor.

Compatibility Considerations

The names and menu paths for some menu items have changed from what they were in the R2012a Simulink Editor. For a summary of the changes to menus, keyboard and mouse shortcuts, badges, and Simulink preferences, see "Simulink Editor Changes" on page 11-7.

Palette for commonly used actions

You can use a palette of icons (to the left of the canvas area) to perform common actions, such as adding annotations or marquee zooming.

Panning and zooming

Pan through a model by pressing the mouse scroll wheel and dragging the mouse or by pressing the space bar while dragging the mouse. Zoom using the mouse scroll wheel. To change the default action of the mouse scroll wheel, use Simulink preferences.

For more information, see Zoom and Pan Block Diagrams.

Display of overlapping blocks

Control which block appears on top, when there are overlapping blocks. From a block context menu, select **Arrange**and then choose either **Bring To Front** or **Send to Back**. (In the R2012a Simulink Editor, Simulink automatically set the display order, based on alphabetical order of the block name.)

Unification of Simulink and Stateflow Editors

The Simulink and Stateflow Editors now share most menu items. Additional aspects of the editor unification include:

- **Unified canvas:** Edit models and Stateflow charts in the same window.
- **Unified Model Browser tree:** Model Browser tree shows complete hierarchy, including Stateflow.

Also, you can now access the same editor functionality on Windows, UNIX®, and Mac platforms.

Simulink Editor preferences

The Simulink Preferences dialog box now includes preferences to control the look and behavior of the Simulink Editor. For example, you specify how the scroll wheel behaves, the look of the diagram (the diagram theme), and the toolbar configuration at a specific level (for example, whether or not to display the **Simulation** toolbar).

To access the Simulink Editor preferences from the Simulink Editor, select **File** > **Simulink Preferences** > **Editor Defaults**. To access those preferences from the Library Browser, select **File** > **Preferences** > **Editor Defaults**.

Compatibility Considerations

The following R2012a preferences do not appear in R2012b:

- **Window reuse**
- **Display Defaults for New Models** > **Browser visible**

In the new Simulink Editor, you can control these editor behaviors directly from within the editor. For details, see "Simulink Preferences Changes" on page 11-19.

Toolbar and status bar control

You can control the hiding or displaying of toolbars and the status bar by using Simulink Editor Default preferences. These preferences persist across editor sessions.

Compatibility Considerations

The model parameters `StatusBar` and `ToolBar` have no effect in the Simulink Editor. These parameters will be removed in a future release.

Visual editing based on model objects

The Simulink Editor provides tools such as smart guides and automated line routing that work based on the relative locations of model objects. This approach replaces pixel-oriented visual editing.

Compatibility Considerations

The model parameters `GridSpacing` and `ShowGrid` have no effect in the Simulink Editor. These parameters will be removed in a future release.

Improved callback error handling

When an interactive operation triggers a callback that causes an error in MATLAB, Simulink:

- Undoes the operation
- Issues an error message

Compatibility Considerations

Before R2012b, for an interactive operation that triggered a callback error in MATLAB, Simulink reported a warning (not an error), even though Simulink stopped the callback operation at the point of failure.

Simulink Editor Changes

- "Mapping from R2012a Simulink Editor to the New Simulink Editor" on page 11-8

Mapping from R2012a Simulink Editor to the New Simulink Editor

The following tables list the new Simulink Editor menu bar items that are different from the R2012a Simulink Editor.

The tables do *not* include menu bar items that:

- Have not changed
- Appear only in the new Simulink Editor

The arrows (>) indicate nested menu paths.

File Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Preferences	**File > Simulink Preferences** and **File > Stateflow Preferences**.
Export to Web	**File > Export Model to Web**.
Print Details	**File > Print > Print Details**.
Print Setup	**File > Print > Printer Setup**.
Enable Tiled Printing	**File > Print > Enable Tiled Printing**.

Edit Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Copy Model to Clipboard	**Edit > Copy Current View to Clipboard** (copies the open Model Editor window contents to the clipboard).
Create Subsystem	**Diagram > Subsystem & Model Reference > Create Subsystem from Selection**.
Create Mask	If the block is not already masked, the dynamic menu path is **Diagram > Mask > Create Mask**. If the block is already masked, the dynamic menu path is **Diagram > Mask > Edit Mask**.
Look Under Mask	**Diagram > Mask > Look Under Mask**.
Link Options	**Diagram > Library Link**.
Links and Model Blocks > Refresh	**Diagram > Subsystem & Model Reference > Refresh Selected Model Block** or, for library links, **Diagram > Refresh Blocks**.
Links and Model Blocks > Model Block Normal Mode Visibility	**Diagram > Subsystem & Model Reference > Model Block Normal Mode Visibility**.
Update Diagram	**Simulation > Update Diagram**.

View Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Back **Forward** **Go To Parent**	Access each option using **View > Navigate**. **Go To Parent** changed to **Up to Parent**.
Model Browser Options	**View > Model Browser**.
Block Data Tip Options	**Display > Blocks > Tool Tip Options**.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
	The submenu item **Block Description** changed to **Description**.
Requirements	**View > Requirements at This Level**.
Signal Hierarchy	**Diagram > Signals & Ports > Signal Hierarchy**.
Sample Time Legend	**Display > Sample Time > Sample Time Legend**.
Zoom In **Zoom Out** **Fit System To View** **Normal (100%)**	Access each option using **View > Zoom**. **Fit System to View** changed to **Fit to View**. **Normal (100%)** changed to **Normal View (100%)**.
Show Page Boundaries	**File > Print > Show Page Boundaries**.
Port Values submenus, such as **Show When Hovering**	**Display > Data Display in Simulation**.
Highlight/Remove Highlighting	**Display > Highlight Signal to Source** or **Display > Highlight Signal to Destination** or **Display > Remove Highlighting**.

Simulation Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Configuration Parameters	**Simulation > Model Configuration Parameters**.
Normal **Accelerator** **Rapid Accelerator** **Software-in-the-Loop (SIL)** **Processor-in-the-Loop**	Access each option using **Simulation > Mode**.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
External	
Start	**Simulation > Run**.

Format Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Font	**Diagram > Format > Font Style**. On Mac platforms, supported fonts must be in both the X11 and Mac font manager.
Text Alignment Enable TeX Commands	Access each option using **Diagram > Format**.
Show Name	**Diagram > Format > Show Block Name**.
Show Drop Shadow	**Diagram > Format > Block Shadow**.
Show Port Labels	**Diagram > Format > Port Labels**.
Foreground Color Background Color	Access each option using **Diagram > Format**.
Screen Color	**Diagram > Format > Canvas Color**.
Show Smart Guides	**View > Smart Guides**.
Align Blocks Distribute Blocks Resize Blocks	Access using block alignment, distribution, and resizing options using **Diagram > Arrange**.
Flip Name	**Diagram > Rotate & Flip > Flip Block Name**.
Flip Block	**Diagram > Rotate & Flip > Flip Block**.
Rotate Block	Access block rotation options using **Diagram > Rotate & Flip**.
Port/Signal Displays	Access most port and signal display options using **Display > Signals & Ports**.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
	Signal Resolution Indicators changed to **Signal to Object Resolution Indicator**.
Block Display > Sorted Order	**Display > Blocks > Sorted Execution Order**.
Block Display > Model Block Version	**Display > Blocks > Block Version for Referenced Models**.
Block Display > Model Block I/O Mismatch	**Display > Blocks > Block I/O Mismatch for Referenced Model**.
Block Display > Execution Context Indicator	**Display > Blocks > Sorted Execution Order**.
Library Link Display	Access library link display options using **Display > Library Links**.
Sample Time Display	Access sample time options using **Display > Sample Time**. **None** changed to **Off**.

Tools Menu

The tools that appear in the **Tools** menu and other menus reflect the products for which you have a license.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Compare Simulink XML Files	Access these tools using **Analysis**.
Control Design	The menu text for these items has changed:
Coverage Settings	• **Coverage Settings** changed to **Coverage**.
Design Verifier	• **Fixed-Point** changed to **Fixed Point Tool** (i.e., there is no hyphen now)
Fixed-Point Tool	
Model Advisor	• **Model Dependencies > View/Edit Manifest Contents** changed to **Model Dependencies > Edit Manifest Contents**.
Model Dependencies	
Parameter Estimation	

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Profiler **Requirements**	• **Profiler** changed to **Show Profiler Report**.
Simulink Debugger	**Simulation > Debug > Debug Model**.
Bus Editor **Lookup Table Editor**	Access these tools using **Edit**.
Inspect Logged Signals	**Simulation > Output > Simulation Data Inspector**.
Signal & Scope Manager	**Diagram > Signals & Ports > Signal & Scope Manager**.
Code Generation **External Mode Control Panel** **HDL Code Generation** **Simulink Code Inspector** **Verification Wizards**	Access these options using **Code**. **Code Generation** changed to **C/C++ Code**. **HDL Code Generation** changed to **HDL Code**.
Define Data Classes **Data Object Wizard**	Access these options using **Code > Data Objects**. **Define Data Classes** changed to **Design Data Classes**.
Response Optimization	**Analysis > Response Optimization**.

Help Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Using Simulink	**Help > Simulink > Simulink Help**.
Blocks **Blocksets**	**Help > Simulink > Blocks & Blocksets Reference**.
Block Support Table	**Help > Simulink > Block Data Types & Code Generation Support**.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Shortcuts	**Help > Keyboard Shortcuts**.
S-Functions	**Help > Simulink > S-Functions**.
Demos	**Help > Simulink > Examples** and **Help > Stateflow > Examples**.
About Simulink	**Help > About > Simulink**.

Simulink Editor Context Menu Changes

From the Canvas

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Back **Forward** **Go to Parent**	Not available from context menu. From the menu bar, select the appropriate menu item from **View > Navigate**.
Configuration Parameters	**Model Configuration Parameters**.
Format > Wide Nonscalar Lines **Format > Signal Dimensions** **Format > Port Data Types**	Access these options using **Display > Signals & Ports**.
Link Options	Not available from the canvas context menu. Access either from a block context menu or from the menu bar, using **Diagram > Library Link**.
Requirements	**Requirements at This Level**.
Screen Color	**Canvas Color**.
Signal & Scope Manager	Not available from context menu. From the menu bar, use **Diagram > Signals & Ports > Signal & Scope Manager**.

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Fixed-Point Tool	Fixed Point Tool.

From a Block

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Block Properties	Properties.
Foreground Color Background Color	Access these options using **Format**.
Convert to Model Block (for Subsystem block context menu)	Subsystem & Model Reference > Convert Subsystem to > Referenced Model.
Format > Flip Block Format > Flip Name Format > Rotate Block	Access these options using **Rotate & Flip**.
Format > Font	Format > Font Style.
Format > Show Drop Shadow	Format > Block Shadow.
Format > Show Name	Format > Show Block Name.
Format > Show Port Labels	Format > Port Labels.
HDL Code Generation	HDL Code.
Link Options	Not available from context menu. From the menu bar, use **Diagram > Library Link**.
Mask Subsystem	If the block is *not* already masked, the dynamic menu path is **Mask > Create Mask**. If the block is already masked, the dynamic menu path is **Mask > Edit Mask**.
Look Under Mask	Mask > Look Under Mask.
Port Signal Properties	Signals & Ports.

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Signal & Scope Manager	Not available from context menu. From the menu bar, use **Diagram > Signals & Ports > Signal & Scope Manager**.
Create Subsystem	**Subsystem & Model Reference > Create Subsystem from Selection**.
Code Generation > Generate Protected Model (from a Model block)	From a Model block: **C/C++ Code > Generate Protected Model** Note that you can also access this option from the main menu: **Code > C/C++ Code > Generate Protected Model**.
Refresh (from a Model block)	**Subsystem & Model Reference > Refresh Selected Model Block**

From a Signal

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Connect to Existing Viewer	**Connect to Viewer**.
Disconnect & Delete Viewer	Use a combination of **Disconnect Viewer** and **Delete Viewer**.
Fixed-Point Tool	Not available from context menu. From the menu bar, use **Analysis > Fixed Point**.
Highlight to Source **Highlight to Destination**	**Highlight Signal to Source**. **Highlight Signal to Destination**.
Linearization Points	**Linear Analysis Points**.

Simulink Editor Mouse and Keyboard Shortcut Changes

Mouse Scroll Wheel

By default, in the new Simulink Editor, rolling the mouse scroll wheel up zooms in on a model, and rolling the wheel down zooms out.

To scroll left and right using the mouse scroll wheel, press **Shift** while rolling the wheel. To scroll up and down, press **Ctrl** while rolling the wheel.

You can change the default scroll wheel behavior. In the Simulink Preferences dialog box, clear **Editor Defaults** > **Scroll wheel controls zooming**.

Model Viewing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Zoom in	**r**	Use the mouse scroll wheel or **Ctrl++** (the plus sign)
Zoom out	**v**	Use the mouse scroll wheel or **Ctrl+-** (the minus sign).
Zoom to normal view	**1**	**Alt+1**
Pan left	**d** or **Ctrl+Left Arrow**	If scroll bars are visible, then with nothing selected, use **Shift+Left Arrow** or for finer panning, just the **Left Arrow**.
Pan right	**g** or **Ctrl+Right Arrow**	If scroll bars are visible, then with nothing selected, use **Shift +Right Arrow** or for finer panning, just the **Right Arrow**.
Pan up	**e** or **Ctrl+Up Arrow**	If scroll bars are visible, then with nothing selected, use **Shift+Up Arrow** or for finer panning, just the **UpArrow**.
Pan down	**c** or **Ctrl+Down Arrow**	If scroll bars are visible, then with nothing selected, use **Shift +Down Arrow** or for finer panning, just the **Down Arrow**.
Fit selection to screen	**f**	You can also use marquee zoom to fit a selection to the screen.

Block Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Move a block from one model to another model	**Shift**, press the left mouse button, and drag block to different model. The block	In the new Simulink Editor, **Shift**, press the left mouse button, and drag block to

11-17

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
	is disconnected moved from the source model to the other model.	different model copies the block, but does not remove it from the source model.

Line Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Create line segments	Move the cursor to the end of line and drag the line.	New Simulink Editor performs autorouting. You can manually control the line segment by clicking the arrow guides. The arrow guides appear when you edit a previously disconnected signal from its endpoint. During new line creation, let go of the mouse button and then move it off the newly created endpoint.
Create diagonal line segments	Click anywhere on a line, **Shift**, and move the cursor.	For an existing line, click a bend (corner) or solder (joint) point, **Shift**, and move the cursor. For new lines, use **Shift** and the arrow guides.

Signal Label Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Delete signal label	**Shift** and click label. Then press **Delete**.	Right-click the label and from the context menu, select **Delete Label**.

Annotation Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Delete annotation	**Shift** and select annotation. Then press **Delete**.	Right-click the annotation and from the context menu, select **Delete**.

Simulink Editor Badges Changes

Badges are the icons that appear in the Simulink Editor to provide information about how you have configured a model.

Badge	R2012a Simulink Editor Badge	New Simulink Editor Badge
Signal viewer		
Library link active		
Library link inactive		
Library link locked		
Library link parameterized		
Model protected		

Simulink Preferences Changes

Two R2012a Simulink preferences no longer appear in R2012b. In the new Simulink Editor, you can control these editor behaviors directly from within the editor.

R2012a Simulink Preference	New Simulink Editor Equivalent
Window reuse	The Simulink Editor opens subsystems in the same window that it uses for the model that contains the subsystem. There is no global setting to change that behavior. However, you can control the window and tab behavior when opening a specific subsystem. For details, see Navigate Model Hierarchy.
Browser visible	To display or hide the Model Browser, select or clear the **View** > **Model Browser** > **Show Model Browser** option.

Also, the MATLAB Preferences dialog box **Figure Copy Template** > **Copy Options** preferences no longer apply to copying a model from the clipboard to a third-party application.

Simulink and Stateflow Editor Customization Changes

Customizing the new Simulink and Stateflow editors is as it was in R2012a, with the following exceptions:

- The addition of custom menu functions to the ends of top-level menus depends on the active editor:

  - Menus bound to `Simulink:FileMenu` only appear when the Simulink Editor is active.

  - Menus bound to `Stateflow:FileMenu` only appear when the Stateflow Editor is active.

  - To have a menu to appear in both of the editors, call `addCustomMenuFcn` twice, once for each tag. Check that the code works in both editors.

- If a filter is applied to the `Simulink` tag, then a menu item that existed in Simulink and Stateflow editors in R2012a is filtered, regardless of the active editor type. However, if the filter is applied to the `Stateflow` tag, then the menu item is only filtered in the Stateflow Editor.

- If a menu item tag has changed in R2012b, you do not need to change the tag to the R2012b tag.

For more information about customizing menus, see Add Items to Model Editor Menus and Disable and Hide Model Editor Menu Items.

Connection to Educational Hardware

Support for Arduino and PandaBoard hardware

- "Support for Arduino Mega 2560 and Arduino Uno hardware" on page 11-21
- "Support for PandaBoard hardware" on page 11-22

Support for Arduino Mega 2560 and Arduino Uno hardware

Run Simulink models on Arduino Mega 2560 and Arduino Uno hardware. For more information, see the Arduino topic.

To use this capability, first run Target Installer and install support for Arduino hardware. To run Target Installer, enter targetinstaller in the MATLAB Command Window.

After installing support, you can use the Simulink "**Target for Use with Arduino Hardware**" block library. To open this block library, enter `arduinolib` in the MATLAB Command Window.

This block library contains the following blocks:

- Arduino Analog Input
- Arduino PWM
- Arduino Digital Input
- Arduino Digital Output
- Arduino Serial Receive
- Arduino Serial Transmit
- Arduino Standard Servo Read
- Arduino Standard Servo Write
- Arduino Continuous Servo Write

After you install support, Target Installer displays the following examples:

- Getting Started with Arduino Hardware
- Communicating with Arduino Hardware (Arduino Mega 2560 only)
- Servo Control

- Drive with PID Control

Support for PandaBoard hardware

Run Simulink models on PandaBoard hardware. For more information, see the PandaBoard topic.

To use this capability, first run Target Installer and install support for PandaBoard hardware. To run Target Installer, enter targetinstaller in the MATLAB Command Window.

After installing support, you can use the Simulink "**Target for Use with PandaBoard Hardware**" block library. To open this block library, enter `pandaboardlib` in the MATLAB Command Window.

This block library contains the following blocks:

- PandaBoard UDP Receive
- PandaBoard UDP Send
- PandaBoard ALSA Audio Capture
- PandaBoard ALSA Audio Playback
- PandaBoard V4L2 Video Capture
- PandaBoard SDL Video Display

Bluetooth download to LEGO MINDSTORMS NXT hardware

You can use a Bluetooth® connection instead of a USB cable to download a Simulink model from your host computer to the LEGO MINDSTORMS NXT hardware. Previously, a USB cable was the only connection available for downloading models to the NXT hardware. For more information, see:

- Run Model on NXT Brick
- Set Up A Bluetooth Connection

Performance

Simulation Performance Advisor that analyzes your model and provides advice on how to increase simulation performance

Use the Performance Advisor to check models for conditions and configuration settings that can result in inefficient simulation of the system that the model represents. The Performance Advisor produces a report that lists the suboptimal conditions or settings that it finds, suggesting better model configuration settings where appropriate. It also provides mechanisms for automatically fixing warnings and failures or allowing you to fix them manually. For more information, see Consult the Performance Advisor.

Project and File Management

Simulink default file format SLX that uses the OPC standard

In R2012b, Simulink has a new default file format for models, SLX, with the file extension `.slx`. In R2012a, SLX was available as an option.

The SLX file format contains the same information as an MDL file and is a compressed package that conforms to the Open Packaging Conventions (OPC) interoperability standard. SLX stores model information using Unicode UTF-8 in XML and other international formats.

Saving Simulink models in the SLX format:

- Typically reduces file size. The file size reduction between MDL and SLX varies depending on the model.
- Solves some problems in previous releases with loading and saving MDL files containing Korean and Chinese characters.
- Supports new features in future releases not supported with MDL format.

You can still choose to save model files as MDL, and the MDL format will remain available for the foreseeable future.

For more information, see Saving Models in the SLX File Format.

Compatibility Considerations

If you upgrade an MDL file to SLX file format, the file contains the same information as the MDL file, and you always have a backup file. All functionality and APIs that currently exist for working with models, such as the `get_param` and `set_param` commands, are also available when using the SLX file format.

The MDL file format will continue to be supported, but, after R2012b, new features might be available only if you use the SLX file format.

The new file extension `.slx` might cause compatibility issues if your scripts contain hard-coded references to file names with extension `.mdl`. To check for problems, verify that your code works with both the MDL and SLX formats. If you find any places in your

scripts that need to be updated, use functions like `which` and `what` instead of strings with `.mdl`.

Caution: If you use third-party source control tools, be sure to register the model file extension `.slx` as a binary file format. If you do not, these third-party tools might corrupt SLX files when you submit them.

Operations with Possible Compatibility Considerations	What Happens	Action
Hard-coded references to file names with extension `.mdl`.	Scripts cannot find or process models saved with new file extension `.slx`.	Make your code work with both the `.mdl` and `.slx` extension. Use functions like `which` and `what` instead of strings with `.mdl`.
Third-party source control tools that assume a text format by default.	Binary format of SLX files can cause third-party tools to corrupt the files when you submit them.	Register `.slx` as a binary file format with third-party source control tools.

The format of content within MDL and SLX files is subject to change. Use documented APIs (such as `get_param`, `find_system` and `Simulink.MDLInfo`) to operate on model data.

Simulink Upgrade Advisor to help migrate files to the current release

Use the Upgrade Advisor for help with using the current release to upgrade and improve models.

The Upgrade Advisor identifies cases where you can benefit by changing your model to use new features and settings in Simulink. The Advisor provides advice for transitioning to new technologies and upgrading a model hierarchy.

The Upgrade Advisor also identifies cases when a model will not work because changes and improvements in Simulink require changes to a model.

The Upgrade Advisor offers options to perform recommended actions automatically or instructions for manual fixes.

See Consult the Upgrade Advisor.

Built-in SVN adapter for Simulink Projects that provides connectivity to SVN and support for server-based repositories

Simulink Projects now provide built-in Subversion source control integration, reducing setup and providing faster performance and improved support for connecting to servers. You can now connect to servers that require login.

Previously, you had to install an additional command-line SVN client to use Projects with SVN. Now you can use SVN for project source control with no additional installation steps.

See Subversion Integration with Projects.

Simulink Project Tool dependency graph that provides highlights by file type, dependency type, and label

After you run dependency analysis on your project, you can use the Graph view to examine dependencies for impact analysis. You can now highlight dependencies by file type, dependency type, and file labels. For example, you might want to highlight model files and see which have the label To Review. Previously, you could highlight only upstream and downstream dependencies of the selected file. Now you can also highlight circular dependencies.

You can now also perform file operations in the Graph view, such as **Open**, **Add to Project**, **Add Label**, and **Remove from Project**. File operations can be useful when using the graphical dependency view to identify required changes to your project, such as identifying files that need removing or files that share a common label.

The Dependencies results list and Graph view are now separate tree node views for efficient workflow, instead of tabs within the Dependency Analysis view. These views also now display a time stamp to identify when the analysis was performed.

See Analyze Project Dependencies.

Redesigned graphical tool for efficient Simulink Projects workflow

The Simulink Project Tool is redesigned for more efficient workflow and access to tools.

- Simulink Projects are integrated with the MATLAB Toolstrip when you dock the tool, with new options to create and open recent projects from MATLAB.
- The toolstrip contains components that were previously available in menus and toolbars.
- All project views have new tools for improved browsing, searching, and filtering.
- There are new tree nodes for accessing the batch processing, dependency results, and dependency graph views.
- The new source control pane provides access to configuration management tasks and more space in other views.
- New archive options include the capability to create a new project from a zip archive.
- New project integrity checks assist with MDL to SLX upgrades, check for `slprj` folders added to projects, and provide **Fix** buttons for tasks that can be automated.

See Simulink Projects.

Batch operation support for files in a Simulink Project

The Simulink Project Tool has a new Batch Job view to help you create and run functions on selected project files. Batch tools provide guidance for creating your own batch functions. An example batch job function identifies and saves any model files that contain unsaved changes.

See Run Batch Functions on Project Files.

Create and open recent Simulink Projects from MATLAB

The MATLAB Toolstrip has new options to create Simulink Projects and open recent projects direct from the MATLAB Desktop.

Simulink Projects are integrated with the MATLAB Toolstrip when you dock the tool.

See:

- Create a New Simulink Project
- Open Recent Projects
- Create a New Project from an Archived Project
- Retrieve a Working Copy of a Project from Source Control

.

Block Enhancements

Menu item to convert configurable and normal subsystems to variant subsystems

Previously, to convert configurable or normal subsystems to variant subsystems, you had to create a new variant subsystem in your model and manually modify it to match the subsystem you were converting.

That method was error-prone. Manually matching a variant subsystem and its ports to the converted subsystem was also a time-consuming process. Moreover, configurable subsystems will not be supported in a future release.

In this release, you can do this conversion by right-clicking a subsystem and selecting **Subsystems and Model Reference** > **Convert Subsystem To** > **Variant Subsystem**.

Simulink creates a new variant subsystem and an appropriate number of inports and outports that match the converted subsystem.

Masking improvements, including the ability to reuse masks, delete existing masks on blocks, and use the shortcut operator || in mask callback code

Use classes Simulink.Mask and Simulink.MaskParameter to control masks programmatically. With these classes, you can perform the following mask operations:

- Create, copy, and delete masks
- Add, edit, and delete mask parameters
- Get mask owner and set properties for masks and mask parameters

In addition, in this release, you can use the OR operator ||, which was previously prohibited, in mask callback code.

Default output data type of Logic blocks changed to `boolean`

The following blocks now use a default value of `boolean` for **Output data type**. In previous releases, the blocks used `uint8` as the default output data type.

- Compare To Constant
- Compare To Zero
- Detect Change
- Detect Decrease
- Detect Fall Negative
- Detect Fall Nonpositive
- Detect Increase
- Detect Rise Nonnegative
- Detect Rise Positive

Signal Attributes tab of dialog box for Operator blocks renamed to Data Type

For the Logical Operator and Relational Operator blocks, the name of the **Signal Attributes** tab of the block dialog box has changed to **Data Type**.

Parameter name changes for Unit Delay block

For the Unit Delay block, the following parameters have been renamed:

Old Name	New Name
X0	InitialCondition
StateIdentifier	StateName
RTWStateStorageClass	CodeGenStateStorageClass
RTWStateStorageTypeQualifier	CodeGenStateStorageTypeQualifier

New variants of Delay block in Discrete library

The Discrete library now contains the following additional variants of the Delay block:

- Variable Integer Delay block
- Resettable Delay block

You can configure the Delay block to work in the same way as either of these variants.

With the source of the delay length set to `Input port`, the Delay block works as a Variable Integer Delay block.
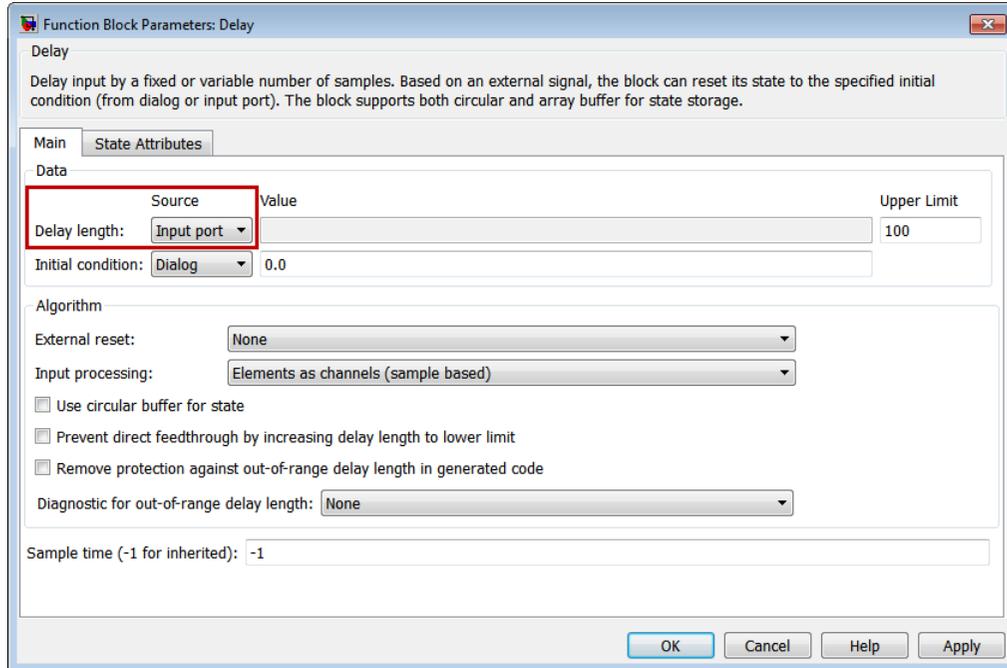
To configure the Delay block to have a resettable delay, set the source of the initial condition to `Input port` and the external reset algorithm to `Rising`.
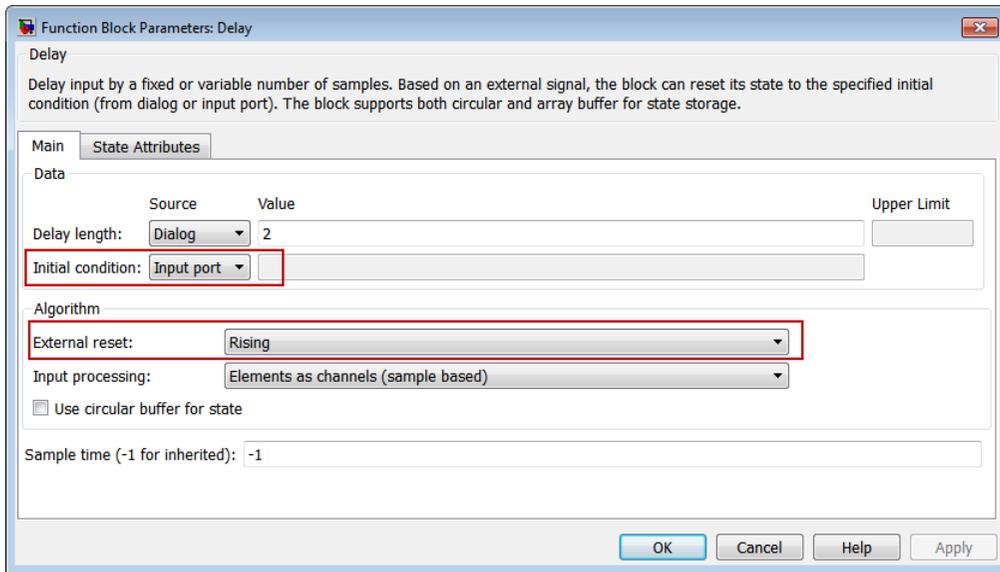
Some Probe block parameters no longer support `boolean` data type

The following parameters of the Probe block no longer support the `boolean` data type:

- Data type for width
- Data type for sample time
- Data type for signal dimensions

If the width, sample time, or dimensions of the input signal has a value greater than zero, the `boolean` data type implicitly represents the output as **1**, which is not a useful result. Setting any of the listed parameters to `Same as input` while the block's input signal data type is `boolean` results in a simulation error.

Internationalization of block dialog box titles and buttons and block tooltips

To enable translation in localized versions of the Simulink software, in this release, the following items relating to Simulink blocks were internationalized:

- Titles of block dialog boxes

- Text for block tooltips

As a result of these changes, in the Japanese version of Simulink, and in future localized versions of the product, these items will display in translated form.

Enabled and triggered subsystems

For triggered and enabled subsystems, the Simulink software now performs zero-crossing detection and zero-crossing state updates of the trigger port outside the enable check.

In previous releases, for triggered and enabled subsystems, the Simulink software performed zero-crossing detection and zero-crossing state updates of the trigger port inside the enable check. This behavior sometimes caused simulation and code generation data mismatches.

Data Management

Variable Editor access from within Model Explorer

In the Model Explorer **Contents** pane, you can use the Variable Editor to edit variables from the MATLAB workspace or model workspace. The Variable Editor is available for editing large arrays and structures.

To open the Variable Editor for a variable that is an array or structure:

1   Click the Value cell for the variable.
2   Select the Variable Editor icon.
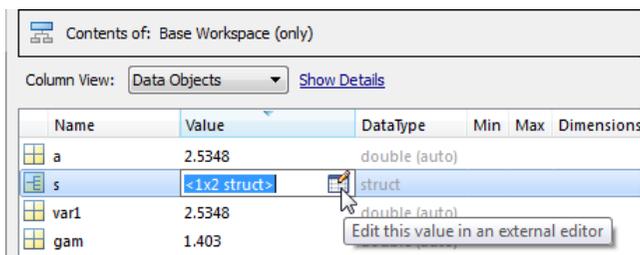
The Variable Editor opens:
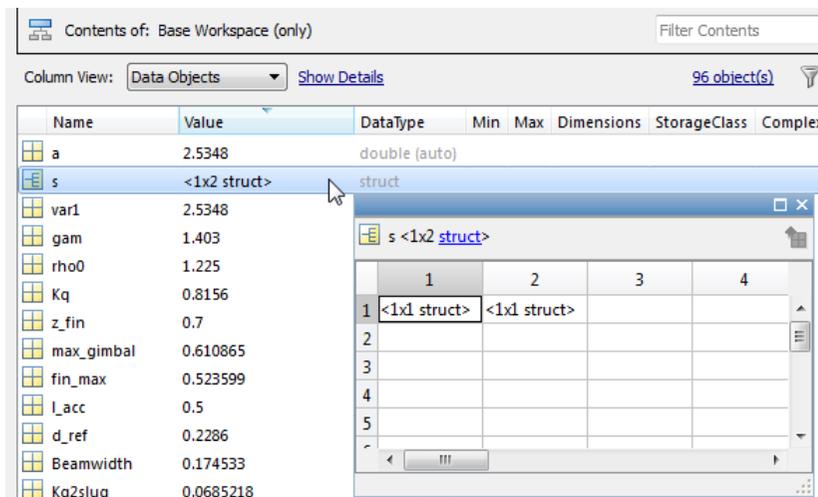
You can resize and move the Variable Editor. The **Contents** pane reflects the edits that you make in the Variable Editor.

For details, see Editing Workspace Variables.

Logged simulation data from Simulation Data Inspector accessible from Simulink toolbar

The record button for the Simulation Data Inspector tool is now accessible on the Simulink toolbar. Previously, the record button was a global setting for all models. The record button now applies per model. Click the record button to select it and then simulate the model to record and inspect logged signal data. You can open the Simulation Data Inspector tool by clicking the down arrow and selecting `Open Simulation Data Inspector`. For more information, see Record Simulation Data.

Compatibility Considerations

The record button no longer appears on the Simulation Data Inspector tool.

Specify verifySignalAndModelPaths action

You can specify the action that the verifySignalAndModelPaths method of the signal logging class `Simulink.SimulationData.ModelLoggingInfo` takes when it detects an invalid path.

Import and map data to root-level input ports

The Configuration Parameters dialog box **Data Import/Export > Input** parameter now has an **Edit Input** button. Use this button to start the Root Inport Mapping tool. This tool lets you import data from a MAT-file and automatically map that data to root-level input ports. For more information, see Import and Map Data to Root-Level Inports.

`Dataset` signal logging format for increased flexibility and ease of use

The default format for saving signal logging data is now `Dataset`.

With the `Dataset` format, you can do the following tasks, which you cannot do with the previous default format of `ModelDataLogs`:

- Work with logging data in MATLAB without a Simulink license
- Log multiple data values for a given time step, which can be important for Iterator subsystem and Stateflow signal logging
- Easily analyze logged signal data for models with deep hierarchies, bus signals, and signals with duplicate or invalid names
- Avoid the limitations of the `ModelDataLogs` format, which Bug Report 495436 describes.

To specify the signal logging format, use the **Configuration Parameters** > **Data Import/Export** > **Signal logging format** parameter. For more information, see Specify the Signal Logging Data Format.

Compatibility Considerations

Before R2012b, the default signal logging format was `ModelDataLogs`. In R2012b, the default format is `Dataset`. The `ModelDataLogs` format will be removed in a future release.

In R2012b, Simulink displays a warning if you run a model that meets *both* of the following conditions:

- The **Configuration Parameters** > **Data Import/Export** > **Signal logging format** parameter is set to `ModelDataLogs`.
- The model has signal logging enabled for at least one signal or uses signal viewer scopes.

Use the Upgrade Advisor to upgrade a model to use `Dataset` format, using *one* of these approaches:

- In the Simulink Editor, select **Analysis** > **Model Advisor** > **Upgrade Advisor**
- From the MATLAB command line, use the upgradeadvisor function.

For more information about how to update models to use `Dataset`, including how to address issues that you might encounter after converting a model to use `Dataset` format, see Migrate from ModelDataLogs to Dataset Format.

Data type field displays user-defined data types

Previously, the data type fields in dialog boxes for various data entities such as data objects and blocks displayed only built-in data types. For `mpt` signal and parameter

objects, you could customize this list to include user-defined data types. For this customization, you had to modify the Simulink customization file `sl_customization.m` to add user-defined data types to the list.

In R2012b, the data type field displays both user-defined and built-in data types, provided these user-defined data types exist in the base workspace. This enhancement is not restricted to `mpt` data objects. All dialog boxes that contain the data type field will display user-defined data types.

Any modifications you make to `sl_customization.m` in order to display user-defined data types will still be supported.

`Simulink.VariableUsage` to get variable information

In R2012b, use Simulink.VariableUsage to determine which blocks use a variable defined in the model, mask, or base workspace.

Previously, you used Simulink.WorkspaceVar to get this information.

Compatibility Considerations

`Simulink.WorkspaceVar` will not be supported in a future release. If you use `Simulink.WorkspaceVar` in your code to programmatically get variable information, replace it with `Simulink.VariableUsage`.

Customizable line specification in Simulation Data Inspector

In the Simulation Data Inspector tool, the line specification includes customizable color selection and more marker specifiers for plotting data points. To view the line specification, in the Signal Browser table, click the **Line** column of a signal. For more information, see Specify the Line Configuration.

Simulation Data Inspector report includes harness model information

A Simulation Data Inspector report of a recorded simulation of a Simulink Verification and Validation harness model now includes model information and a model diagram of the system under test and the test harness model. For more information on generating a report, see Create Simulation Data Inspector Report.

Component-Based Modeling

Model configuration for targets with multicore processors

This capability has the following changes:

- Models configured for concurrent execution can now contain blocks that require implicit ODE solvers, such as physical modeling blocks. In previous releases, models that contained such blocks returned an error message during simulation and code generation.
- Code generation for models configured for concurrent execution, and which contain a large number of continuous states, now have improved performance and decreased memory usage.

New **Simulink.GlobalDataTransfer** class

To configure data transfers for models configured for concurrent execution, use the Simulink.GlobalDataTransfer class. This class contains the properties:

- `DefaultTransitionBetweenSyncTasks`
- `DefaultTransitionBetweenContTasks`
- `DefaultExtrapolationMethodBetweenContTasks`
- `AutoInsertRateTranBlk`

To access the properties of this class, use the `get_param` function to get the handle for this class, then use dot notation to access the properties, for example:

```
dt=get_param(gcs,'DataTransfer');
dt.DefaultTransitionBetweenContTasks

ans =

Ensure deterministic transfer (minimum delay)
```

Reduced memory usage in models with many library links

Simulink now saves memory by closing partially loaded libraries on subsequent simulations. In previous releases, Simulink loaded linked blocks by partially loading

their source libraries when you opened, updated, or simulated the model. These partially loaded libraries were never closed again, resulting in unnecessary memory use. Now Simulink closes partially loaded libraries once they are no longer needed, after loading the linked blocks. Reducing memory use can increase performance in large models with many library links.

Compatibility Considerations

If you have scripts that assume libraries are loaded and try to access the library, these scripts will now produce errors. You must update scripts to load libraries using load_system before running commands such as set_param on a library.

Configuration Reference dialog box to propagate and undo configuration settings to all referenced models

To share a configuration reference among referenced models in a model hierarchy, the Configuration Reference Propagation dialog box provides:

• A list of referenced models in the top model

• The ability to select only specific referenced models for propagation

• After propagation, a display of the status for the converted configuration for each referenced model

• The ability to undo the configuration reference and restore the previous configuration settings for a referenced model

For more information, see Manage Configuration Reference Across Referenced Models and Share a Configuration Across Referenced Models.

Context-dependent function-call subsystem input handling improved

Executing a function-call subsystem that has context-dependent inputs can result in nondeterministic simulation results. Detecting dependent input at simulation time helps to avoid unexpected code generation results.

Before R2012b, if you wanted Simulink to flag such cases as errors, you needed to set the HiliteFcnCallInpInsideContext model parameter each time you load the model. You could not save the setting for that parameter in the model.

In R2012b, to generate an error whenever Simulink has to compute any of a function-call subsystem's inputs directly or indirectly during execution of the function-call subsystem, you can use the new `FcnCallInpInsideContextMsg` parameter argument setting of `EnableAllAsError`. The parameter setting is stored with the model. Set the `FcnCallInpInsideContextMsg` parameter with the **Configurations Parameters** > **Diagnostics** > **Connectivity** > Context-dependent inputs parameter.

Compatibility Considerations

In R2012b, the `HiliteFcnCallInpInsideContext` parameter has been removed. The new `FcnCallInpInsideContextMsg` parameter settings eliminate the need for the `HiliteFcnCallInpInsideContext` parameter, which you could not store with the model.

In R2012b, the `FcnCallInpInsideContextMsg` parameter setting of `Enable All` argument has been replaced by two settings: `EnableAllAsWarning` and `EnableAllAsError`. The `EnableAllAsError` setting is now the default. In R2012a and R2011b, `Enable All` was the default, and in R2011a and earlier, `Use local settings` was the default.

If you have existing code that set the `HiliteFcnCallInpInsideContext` parameter, you need to change that code in R2012b for the following conditions.

Existing Code	R2012b Equivalent Code

`HiliteFcnCallInpInsideContext` set to `on`
 	
`FcnCallInpInsideContextMsg` set to `Enable All`
	Set `FcnCallInpInsideContextMsg` to `EnableAllAsError` and remove `HiliteFcnCallInpInsideContext`.

`HiliteFcnCallInpInsideContext` set to `off`
 	
`FcnCallInpInsideContextMsg` set to `Enable All`
	Set `FcnCallInpInsideContextMsg` to `EnableAllAsWarning` and remove `HiliteFcnCallInpInsideContext`

`FcnCallInpInsideContextMsg` set to `Use local settings`	`FcnCallInpInsideContextMsg` set to `UseLocalSettings`
`FcnCallInpInsideContextMsg` set to `Disable All`	Set `FcnCallInpInsideContextMsg` to `DisableAll`.

> **Note:** The `FcnCallInpInsideContextMsg` settings of `Use local settings` and `Disable all` are maintained for backward compatibility, but may be deprecated in a future release. If code from before R2012b used `get_param` with the `FcnCallInpInsideContextMsg` parameter for the string comparison, then when you run that code in R2012b, the returned results of `UseLocalSettings` and `DisableAll` no longer match the `Use local settings` and `Disable all` strings in the earlier code.

The Model Advisor **Check usage of function-call connections** checks based on the settings of the **Configurations Parameters** > **Diagnostics** > **Connectivity** > **Invalid function-call connection** and **Configurations Parameters** > **Diagnostics** > **Connectivity** > **Context-dependent inputs** parameters. In R2012b, the recommended action was to set **Context-dependent inputs** to `Enable All`. In R2012b, the recommended action is to set it to `Enable all as errors`.

When you save a model that has `FcnCallInpInsideContextMsg` parameter set to `EnableAllAsWarning` or `EnableAllAsError` to an earlier release, Simulink saves the earlier-release model with the `Enable all` setting. The `EnableAllAsError` behavior of generating an error message is not available in the earlier-release model.

`Simulink.Variant` object and the model `InitFcn`

`Simulink.Variant` objects used by a model must be created before simulation is started. If you create or update a `Simulink.Variant` object in the model's callback `InitFcn` function, then at diagram update time, Simulink ignores that object creation or update.

Compatibility Considerations

In earlier versions of Simulink, if you created or updated `Simulink.Variant` objects with the `InitFcn` function, Simulink incorrectly processed the object creation or update, which could lead to incorrect model behavior.

Signal Management

Sample time propagation changes

The way that Simulink software uses the sample time of an enable signal during sample time propagation has been improved for models that contain enabled subsystems with:

- No Inport blocks
- All blocks inside the enabled subsystem specifying an inherited sample time

Simulink now sets the sample times of the contents of enabled subsystems with these conditions to the sample times of their Enable blocks. In previous releases, Simulink did not propagate the sample time of the enable signal to the subsystem contents. Instead, Simulink determined the sample time of the subsystem contents using backpropagation from outside the subsystem.

Compatibility Considerations

This change helps you avoid unintentional multirate enabled subsystems. However, if existing models have Merge blocks whose inputs are driven by enabled subsystem outputs, Model Advisor checks might return errors. Follow the Model Advisor guidelines to resolve the issue.

If you want your model to behave as before, manually set the sample times in your enabled subsystems.

Signal Builder

Signal Builder has the following changes:

- You can now import signal data formatted in a custom format to the Signal Builder block. In previous releases, you could import data only if it complied with the existing format guidelines. For more information, see Importing Data with Custom Formats.
- The Signal Builder block has had minor graphical updates. For more information, see Signal Groups.
- The signalbuilder function now enables you to get the active group label.

User Interface Enhancements

Model Advisor Dashboard

The Model Advisor dashboard provides a way for you to efficiently check that your model complies with modeling guidelines. You can use the Model Advisor dashboard to run a set of checks on your model without opening the Model Advisor window and reloading checks, saving analysis time. To open the Model Advisor dashboard, from the Model Editor, you can either:

•   Select **Analysis** > **Model Advisor** > **Model Advisor Dashboard**.

•
    Select `Model Advisor Dashboard` from the Model Editor toolbar drop-down list.
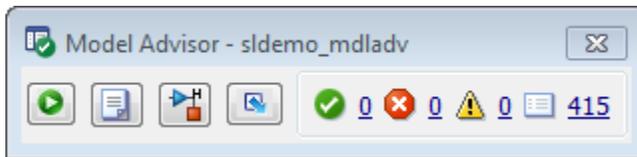
When you use the Model Advisor dashboard, you can select and view checks by clicking the **Switch to Model Advisor** toggle button (). For more information, see Overview of the Model Advisor Dashboard.

Show partial or whole model hierarchy contents

By default, the Model Explorer displays objects for the system that you select in the Model Hierarchy pane. It does not display data for child systems.

Now you can override that default, so that the Model Explorer displays objects for the whole hierarchy of the currently selected system. To toggle between displaying only the current system and displaying the whole system hierarchy of the current system. Use one of these techniques:

•   Select **View** > **Show Current System and Below**.

- Click the **Show Current System and Below** button at the top of the **Contents** pane.
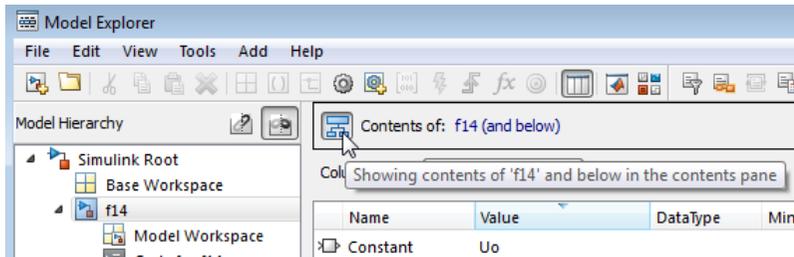
To indicate that you have selected the **Show Current System and Below** option, the Model Explorer:

- In the **Model Hierarchy** pane, highlights in pale blue the current system and its child systems
- After the path in the **Contents of** field, includes `(and below)`
- Changes the **Show Current System and Below** button at the top of the **Contents** pane and in the **View** menu
- In the status bar, indicates the scope of the displayed objects when you hover over the **Show Current System and Below** button

Loading very large models for the current system and below can be slow. To stop the loading process at any time, either click the **Show Current System and Below** button or click another node in the tree hierarchy.

If you show the current system and below, you may want to change the view to better reflect the displayed system contents.

The setting for the **Show Current System and Below** option is persistent across Simulink sessions.

For details, see Displaying Partial or Whole Model Hierarchy Contents.

Compatibility Considerations

The Model Explorer search bar no longer provides the `Search Current System and Below` option. Instead, the search honors the **Show Current System and Below** setting that you set with the **View** menu or **Show Current System and Below** button.

Improved icons for model objects

Improved icons for model objects that the Model Explorer displays (for example, blocks, signals, variables) better represent the objects and are more consistent with icons used in other Simulink tools.

Simulink Debugger

The following capabilities are not available in the debugger. For more information on the debugger, see Introduction to the Debugger:

- Animations
- Adding breakpoints while in the initialization phase
- Displaying I/O while in the initialization phase

Multiple modifiers for custom accelerators

You can now use multiple modifiers when defining custom accelerators. In previous releases, you used only the **Ctrl** key as a modifier for customer accelerators. In R2012b, you can use multiple modifiers for custom accelerators, for example **Ctrl+Alt+T**. See Add Items to Model Editor Menus.

Model Advisor Checks

Verify Syntax of Library Models

There are Model Advisor checks available to verify the syntax of library models. When you use the Model Advisor to check a library model, the Model Advisor window indicates (~) checks that do not check libraries. To determine if you can run the check on library models, you can also refer to the check documentation, in the Capabilities and Limitations section. You cannot use checks that require model compilation. If you have a Simulink Verification and Validation license, you can use an API to create custom checks that support library models.

MATLAB Function Blocks

New toolbox functions supported for code generation

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

Computer Vision System Toolbox

- integralImage

Image Processing Toolbox

- bwlookup
- bwmorph

Interpolation and Computational Geometry

- interp2

String Functions

- deblank
- hex2num
- isletter
- isspace
- isstrprop
- lower
- num2hex
- strcmpi
- strjust
- strncmp
- strncmpi
- strtok
- strtrim
- upper

Trigonometric Functions

- atan2d

New System objects supported for code generation

The following System objects are now supported for code generation. To see the list of System objects supported for code generation, see System Objects Supported for Code Generation.

Communications System Toolbox

- comm.ACPR
- comm.BCHDecoder
- comm.CCDF
- comm.CPMCarrierPhaseSynchronizer
- comm.GoldSequence
- comm.LDPCDecoder
- comm.LDPCEncoder
- comm.LTEMIMOChannel
- comm.MemorylessNonlinearity
- comm.MIMOChannel
- comm.PhaseNoise
- comm.PSKCarrierPhaseSynchronizer
- comm.RSDecoder

DSP System Toolbox

- dsp.AllpoleFilter
- dsp.CICDecimator
- dsp.CICInterpolator
- dsp.IIRFilter
- dsp.SignalSource

R2012a

Version: 7.9

New Features

Bug Fixes

Compatibility Considerations

Component-Based Modeling

Interactive Library Forwarding Tables for Updating Links

Use the new Forwarding Table to map old library blocks to new library blocks. In previous releases, you could create forwarding tables for a library only at the command line. Now you can interactively create forwarding tables for a library to specify how to update links in models to reflect changes in the parameters. To set up a forwarding table for a library, select **File** > **Library Properties**.

You can also specify transformation functions to update old link parameter data using a MATLAB file on the MATLAB path. Transforming old link parameter data for the new library block enables you to load old links and preserve parameter data.

After you specify the forwarding table, any links to old library blocks are updated when you open a model containing links to the library. Library authors can use the forwarding tables to automatically transform old links into updated links without any loss of functionality and data. Use the forwarding table to solve compatibility issues with models containing old links that cannot load in the current version of Simulink. Library authors do not need to run `slupdate` to upgrade old links, and can reduce maintenance of legacy blocks.

For details, see Making Backward-Compatible Changes to Libraries in the Simulink documentation.

Automatic Refresh of Links and Model Blocks

When you save changes to a library block, Simulink now automatically refreshes all links to the block in open Model Editor windows. You no longer need to manually select **Edit** > **Links and Model Blocks** > **Refresh**. When you edit a library block (in the Model Editor or at the command line), Simulink now indicates stale links which are open in the Model Editor by showing that the linked blocks are unavailable. When you click the Model Editor window, Simulink refreshes any stale links to edited blocks, even if you have not saved the library yet.

For details, see Updating a Linked Block in the Simulink documentation.

Model Configuration for Targets with Multicore Processors

You can now configure models for concurrent execution using configuration reference objects or configuration sets. In the previous release, you could use only configuration sets. Existing configuration sets continue to work.

The workflow to configure a model for concurrent execution has these changes:

- To preserve existing configuration settings for your model, in Model Explorer, expand the model node. Under the model, right-click **Configuration**, then select the **Show Concurrent Execution** option. This action updates the Solver pane to display a **Concurrent execution options** section.
- To create new configuration settings, in Model Explorer, right-click the model and select **Configuration** > **Add Configuration for Concurrent Execution**. This action updates the Solver pane to display a **Concurrent execution options section**.

The following changes have also been made:

- The **Ensure deterministic transfer (minimum delay)** data transfer now supports continuous and discrete signals. In the previous release, this data transfer type supported only continuous signals.
- Data transfer has been enhanced to allow signal branch points outside of referenced models. In previous releases, signal branching was supported only within referenced models.
- The following Simulink.SoftwareTarget.TaskConfiguration methods have new names. Use the new method names.

Old Name	New Name
addAperiodicTaskGroup	addAperiodicTrigger
deleteTaskGroup	deleteTrigger
findTaskGroup	findTrigger

- The sldemo_concurrent_execution demo has been updated to reflect the updated software. It also now contains an example of how to configure the model for an asynchronous interrupt.
- In the Concurrent Execution dialog box, the Map Blocks To Tasks node has changed to Tasks and Mapping. The Periodic and Interrupt nodes are now hierarchically under the Tasks and Mapping node.

For more information, see Configuring Models for Targets with Multicore Processors in the Simulink User's Guide.

MATLAB Function Blocks

Integration of MATLAB Function Block Editor into MATLAB Editor

There is now a single editor for developing all MATLAB code, including code for the MATLAB Function block.

Code Generation for MATLAB Objects

There is preliminary support in MATLAB Function blocks for code generation for MATLAB classes targeted at supporting user-defined System objects. For more information about generating code for MATLAB classes, see Code Generation for MATLAB Classes. For more information about generating code for System objects, see the DSP System Toolbox™, Computer Vision System Toolbox™, or the Communications System Toolbox™ documentation.

Specification of Custom Header Files Required for Enumerated Types

If data in your MATLAB Function block uses an enumerated type with a custom header file, include the header information in the **Simulation Target** > **Custom Code** pane of the Configuration Parameters dialog box. In the **Header file** section, add the following statement:

```
#include "<custom_header_file_for_enum>.h"
```

Compatibility Considerations

In earlier releases, you did not need to include custom header files for enumerated types in the Configuration Parameters dialog box.

Data Management

New Infrastructure for Extending Simulink Data Classes Using MATLAB Class Syntax

Previously, you could use only the Data Class Designer to create user-defined subclasses of Simulink data classes such as `Simulink.Parameter` or `Simulink.Signal`.

The Data Class Designer, which is based on *level-1 data class infrastructure*, allows you to create, modify, or delete user-defined packages containing user-defined subclasses.

In a future release, support for level-1 data class infrastructure is being removed.

There are disadvantages to using the level-1 data class infrastructure:

- The syntax for defining data classes using this infrastructure is not documented.
- The data classes are defined in P-code.
- The infrastructure offers limited capability for defining data classes.

  - It does not allow you to add methods to your data classes.
  - It does not allow you to add private or protected properties to your data classes.

- It permits partial property matching and does not enforce case sensitivity. For example, after creating a `Simulink.Parameter` data object

  ```
  a = Simulink.Parameter;
  ```
  you could set the property `Value` of the data object by using the following command:

  ```
  a.value = 5;
  ```

In R2012a, a replacement called *level-2 data class infrastructure* is being introduced. This infrastructure allows you to extend Simulink data classes using MATLAB class syntax.

Features of level-2 data class infrastructure:

- Ability to upgrade data classes you defined using level-1 data class infrastructure. To learn how to upgrade, see Upgrade Level-1 Data Classes to Level-2.
- Complete flexibility in defining your data classes, which can now have their own methods and private properties.

- Simplified mechanism for defining custom storage classes for your data classes.
- Strict matching for properties, methods, and enumeration property values.
- Ability to define data classes as readable MATLAB code, not P-code. With class definitions in MATLAB code, it is easier to understand how these classes work, to integrate code using configuration management, and to perform peer code reviews.

See the detailed example, Define Level-2 Data Classes Using MATLAB Class Syntax.

Compatibility Considerations

When you migrate your level-1 data classes to level-2 data classes, the way that MATLAB code is generated and model files are loaded remains the same. However, you may encounter errors if your code includes the following capabilities specific to level-1 data classes:

- Inexact property names such as `a.value` instead of the stricter `a.Value`.
- The `set` method to get a list of allowable values for an enumeration property.
- Vector matrix containing `Simulink.Parameter` and `Simulink.Signal` data objects. Previously, using level-1 data classes, you could define a vector matrix `v` as follows:

```
a = Simulink.Signal;
b = Simulink.Parameter;
v = [a b];
```
However, level-2 data classes do not support such mixed vector matrices.

In these cases, modify your code to replace these capabilities with those supported by level-2 data classes. For more information on how to make these replacements, see MATLAB Object Oriented Programming.

Change in Behavior of isequal

Previously, when you used function `isequal` to compare two Simulink data objects, the function compared only the handles of the two objects. This behavior was incorrect and did not conform to the intended behavior of isequal in MATLAB. Consider the following example:

```
a = Simulink.Parameter;
b = Simulink.Parameter;
isequal(a,b);
```

```
ans = false
```

In R2012a, the behavior of `isequal` has changed to conform to the intended behavior of isequal in MATLAB. Now, `isequal` compares two Simulink data objects by comparing their individual property values. Based on the above example, provided objects `a` and `b` have similar property values, the new result will be as follows.

```
a = Simulink.Parameter;
b = Simulink.Parameter;
isequal(a,b);
ans = true
```

isContentEqual Will Be Removed in a Future Release

Previously, you could use method `isContentEqual` to compare the property values of two Simulink data objects.

In this release, the behavior of `isequal` has been changed so that it can replace `isContentEqual`.

In a future release, support for `isContentEqual` will be removed. Use `isequal` instead.

Compatibility Considerations

If you are using the `isContentEqual` method in your MATLAB code to compare Simulink data objects, replace all instances of `isContentEqual` with `isequal`.

Change in Behavior of int32 Property Type

Previously, when you created `int32` properties for a level-1 data class using the Data Class Designer, the property value was stored as a double-precision value.

In R2012a, the behavior of `int32` properties has changed. Now, `int32` properties for a level-2 data classes are stored as a single-precision values.

RTWInfo Property Renamed

In R2012a, the property `RTWInfo` of a Simulink data object has been renamed as `CoderInfo`.

Compatibility Considerations

If your code uses the `RTWInfo` property to access data object parameters such as `StorageClass`, replace instances of `RTWInfo` in your code with `CoderInfo`. Your existing code will continue to work as before.

deepCopy Method Will Be Removed in a Future Release

Previously, you could use a Simulink data object's `deepCopy` method to create a copy of the data object along with its properties.

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

In a future release, the `deepCopy` method will be removed. Use the `copy` method instead.

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

The `copy` does not create a reference. To create a reference, use the following commands.

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

New Methods for Querying Workspace Variables

Previously, you could query model workspace variables using the `evalin` method, but you had to resave your model after using this method.

In R2012a, use two new `Simulink.Workspace` methods to query workspace variables without having to resave your model:

- hasVariable: Determines if a variable exists in the workspace.
- getVariable: Gets the value of a variable from the workspace.

Default Package Specification for Data Objects

In R2012a, you can specify a default data package other than **Simulink**. Set the default package in the Data Management Defaults pane of the Simulink Preferences dialog box.

Simulink applies your default package setting to the Model Explorer, Data Object Wizard, and Signal Properties dialog box.

Simulink.Parameter Enhancements

The following enhancements have been made to `Simulink.Parameter` data objects:

- You can now explicitly specify the data type property of a `Simulink.Parameter` object as `double`.
- When casting values to specified data types, the values of `Simulink.Parameter` objects are now cast using the casting diagnostic used for block parameters.

  - Scalar value is cast to fixed-point data type.
  - Array or matrix value is cast to fixed-point data type.
  - Structure value is cast to bus data type.

  For more information on creating typesafe models, see Data Typing Rules

Custom Storage Class Specification for Discrete States on Block Dialog Box

Previously, you could specify custom storage classes (CSCs) for discrete states only by creating a signal object in the base workspace, associating it with the discrete state, and assigning the CSC to the signal object.

In R2012a, you can specify CSCs for discrete states directly on the block dialog box.

For example, you can specify a CSC for the discrete state of a Unit Delay block as follows:

1 Open the block dialog box.
2 Click the **State Attributes** tab.
3 Select a **Package**.
4 Select the desired CSC from the `Storage class` drop-down list.
5 Set **Custom attributes** for the storage class.

Enhancement to set_param

Previously, when you used `set_param` to make changes to the value of a parameter, Simulink allowed the change to be committed even if the `set_param` operation failed.

Consequently, an invalid value persisted in the parameter and an error was generated during model simulation.

In this release, the behavior of set_param has been enhanced so that Simulink does not change the value of a parameter if the set_param operation fails. Instead, the parameter retains its original value.

Compatibility Considerations

1 Sections of your code might not work if they depend on the value of a parameter set using set_param within a try-catch block. Consider revising such sections of code to account for the new behavior.

2 If you are setting dependent parameters using separate set_param commands for each parameter, consider revising the code so that all dependent parameters are set using a single command. Setting individual parameters might cause the set_param operation to fail for the dependent parameters.

For example, consider three dependent parameters of the Integrator block: Lower Saturation, Upper Saturation, and Initial Condition. The dependency condition among these parameters is as follows: Lower Saturation ≤ Initial Condition ≤ Upper Saturation.

Here, setting one parameter at a time might cause the set_param operation to fail if a dependency condition is not satisfied. It might be better to set all parameters together using a single set_param command.

Avoid

```
try
  set_param(Handle, 'Param1', Value1)
end
  set_param(Handle, 'Param2', Value2)
```
Better

```
set_param(Handle, 'Param1', Value1, 'Param2', Value2)
```

Simulink.findVars Support for Active Configuration Sets

Simulink.findVars now searches for variables that are used in a model's active configuration set. For example, you can now use Simulink.findVars to search for

variables that are used to specify configuration parameters such as Start time and Stop time.

Use either the Model Explorer or the Simulink.findVars command-line interface to search for variables used by an active configuration set.

Bus Support for To File and From File Blocks

The To File block supports saving virtual and nonvirtual bus data.

The From File block supports loading nonvirtual bus data.

Bus Support for To Workspace and From Workspace Blocks

The To Workspace block supports saving bus data, with a new default save format, `Timeseries`. For bus data, the `Timeseries` format uses a structure of MATLAB timeseries objects, and for non-bus data, a MATLAB timeseries object.

The From Workspace block now supports loading bus data. To do so, specify a bus object as the output data type.

Logging Fixed-Point Data to the To Workspace Block

If you configure the To Workspace block to log fixed-point data as `fi` objects, then the workspace variable should use the same data type as the input. To preserve the data type of scaled doubles inputs, Simulink logs them to `fi` objects.

Compatibility Considerations

In releases prior to R2012a, when you configured the To Workspace block to log fixed-point data as `fi` objects and the input data type was scaled doubles, then Simulink discarded the scaled doubles data type and logged the data as doubles.

Improved Algorithm for Best Precision Scaling

In R2012a, using best-precision scaling is less likely to result in proposed data types that could result in overflows. The new algorithm prevents overflows for all rounding modes except `Ceiling`.

For example, consider a Data Type Conversion block with an **Output minimum** of `-128.6`, and rounding mode `Floor`, and data type specified as `fixdt(1,8)`. In previous releases, for an input signal value of `-128.6`, best precision scaling set the output data type to `fixdt(1,8,0)` which resulted in an overflow. In R2012a, for the same model, best precision scaling now prevents such overflows by setting the output data type to `fixdt(1,8,-1)` and the signal value becomes `-128`.

Compatibility Considerations

Best-precision scaling in R2012a calculates a different data type from that calculated in R2011b only if the value being scaled is between (`RepMin` - `LSB`) and `RepMin`, where `RepMin` is the representable minimum of the proposed data type. Under these conditions, the output data type used by the block might change to avoid overflow. This change might reduce precision and result in the propagation of different data types. It might also affect the data types proposed by the Fixed-Point Advisor and Fixed-Point Tool.

Enhancement of Mask Parameter Promotion

The manner in which promoted variables are named in the mask editor has changed.

Consider the following example.

You mask a subsystem that contains two parameters that are candidates for promotion: **Upper Limit** and **Lower Limit**. You promote parameter **Upper Limit**, but later decide to promote a different parameter. So you remove your original promotion of **Upper Limit** and promote parameter **Lower Limit** instead.

In previous releases, even though you changed the promotion to parameter **Lower Limit**, the auto-generated name and prompt for this parameter remained **UpperLimit**.

In this release, when you change the promotion to parameter **Lower Limit**, the variable name and the prompt of the parameter change to **LowerLimit**. However, if you had manually changed the variable name for your originally promoted parameter **Upper Limit** to **ChangedLimit**, the variable name for the new promotion will also be **ChangedLimit**.

File Management

SLX Format for Model Files

In R2012a, Simulink provides a new option to save your model file in the SLX format, with file extension `.slx`. The SLX file format contains the same information as an MDL file and is a compressed package that conforms to the Open Packaging Conventions (OPC) interoperability standard. SLX stores model information using Unicode UTF-8 in XML and other international formats.

Saving Simulink in the SLX format:

- Typically reduces file size. The file size reduction between MDL and SLX varies depending on the model.
- Solves some problems in previous releases with loading and saving MDL files containing Korean and Chinese characters.
- Supports new features in future releases not supported with MDL format.

The default file format remains MDL, and the MDL format will remain available for the foreseeable future.

To use the SLX format, see File format for new models and libraries in the Simulink Preferences documentation.

Compatibility Considerations

SLX will become the default file format in a future release. In R2012a you can optionally save your models in the SLX format. All functionality and APIs that currently exist for working with models, such as the `get_param` and `set_param` commands, are also available when using the SLX file format.

The MDL file format will continue to be supported, but, after R2012a, new features might be available only if you use the SLX file format.

When you use the SLX file format, the new file extension `.slx` might cause compatibility issues if your scripts contain hard-coded references to file names with extension `.mdl`. To check for future problems, verify that your code works with both the MDL and SLX formats. If you find any places in your scripts that need to be updated, use functions like `which` and `what` instead of strings with `.mdl`.

Caution: If you use third-party source control tools, be sure to register the model file extension `.slx` as a binary file format. If you do not, these third-party tools might corrupt SLX files when you submit them.

Release	Operations with Possible Compatibility Considerations	What Happens	Action
R2012a	**In 12a, SLX is optional. No compatibility considerations, unless you choose to save as SLX.**	Nothing, unless you choose to try SLX. If you try SLX, see the following rows for possible impact.	None.
Future release with SLX default.	Hard-coded references to file names with extension `.mdl`.	Scripts cannot find or process models saved with new file extension `.slx`.	Make your code work with both the `.mdl` and `.slx` extension. Use functions like `which` and `what` instead of strings with `.mdl`.
	Third-party source control tools that assume a text format by default.	Binary format of SLX files can cause third-party tools to corrupt the files when you submit them.	Register `.slx` as a binary file format with third-party source control tools.

For more information, see Saving Models in the SLX File Format in the Simulink documentation.

The format of content within MDL and SLX files is subject to change. Use documented APIs (such as `get_param`, `find_system`, and `Simulink.MDLInfo`) to operate on model data.

Simulink Project Enhancements

In R2012a, Simulink projects include the following enhancements:

- New export to Zip file capability to package and share project files.
- Dependency analysis graph views to visualize project file dependencies.
- Easily compare and merge project file labels and shortcuts to resolve conflicts during peer review workflow.
- New ability to load a project and use the project API to run setup tasks on a MATLAB worker.
- Extended source control support with the Source Control Adapter SDK for authoring integration with third-party tools.

For more information on using projects, see Managing Projects in the Simulink documentation.

Compatibility Considerations

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
getRootDirectory	Warns	getRootFolder	Replace all instances of getRootDirectory with getRootFolder

See the Simulink.ModelManagement.Project.CurrentProject reference page.

Signal Management

Signal Hierarchy Viewer

To display the signal hierarchy for a signal:

1 Right-click a signal.

2 Select the **Signal Hierarchy** option to open the new Signal Hierarchy Viewer.

For details, see Signal Hierarchy Viewer.

Signal Label Propagation Improvements

Prior to R2012a, signal label propagation behaved inconsistently in different modeling contexts. Signal label propagation is the process that Simulink uses when it passes signal labels to downstream connection blocks (for example, Subsystem and Signal Specification blocks).

In R2012a, signal label propagation is consistent:

- For different modeling constructs (for example, different kinds of signals, different kinds of buses, model referencing, variants, and libraries)

- In models with or without hidden blocks, which Simulink inserts in certain cases to enable simulation

- At model load, edit, update, and simulation times

For details, see Signal Label Propagation.

Compatibility Considerations

In the Signal Properties dialog box, for the **Show propagated signals** parameter, you can no longer specify the `all` option. When you save a pre-R2012a model in R2012a, Simulink changes the `all` settings to `on`.

The following blocks no longer support signal label propagation. When you open legacy models that have signal label propagation enabled for these blocks, Simulink does not display a warning or error, and does not propagate the signal label.

- Assignment
- Bus Assignment
- Bus Creator
- Bus Selector
- Demux
- Matrix Concatenate
- Mux
- Selector
- Vector Concatenate
- Bus-capable blocks (Memory, Merge, Multiport Switch, Permute Dimensions, Probe, Rate Transition, Reshape, S-Function, Switch, Unit Delay, Width, and Zero-Order Hold)

You can name the output of a Bus Creator block and choose to have that name propagated to any downstream connection blocks.

To view the hierarchy for any bus signal, use the new Signal Hierarchy Viewer.

Frame-Based Processing: Inherited Option of the Input Processing Parameter Now Provides a Warning

Some Simulink blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will have a new parameter that allows you to specify the appropriate processing behavior.

To prepare for this change, many blocks received a new **Input processing** parameter in previous releases. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending upon the type of processing you want. The third choice, `Inherited (this choice will be removed - see release notes)`, is a temporary selection that is available to help you migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

In this release, your model provides a warning when the following conditions are all met for any block in your model:

- The **Input processing** parameter is set to `Inherited (this choice will be removed - see release notes)`.
- The input signal is frame-based.
- The input signal is a vector, matrix, or N-dimensional array.

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the slupdate function. The function detects all blocks that have `Inherited (this choice will be removed - see release notes)` selected for the **Input processing** parameter. It then asks you whether you would like 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)`.

In a future release, the frame bit and the `Inherited (this choice will be removed - see release notes)` option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels (frame based)` or `Elements as channels (sample based)`. The option set will depend on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Logging Frame-Based Signals

In this release, a new warning message appears when a Simulink model is logging frame-based signals and the **Signal logging format** is set to `ModelDataLogs`. In `ModelDataLogs` mode, signals are logged differently depending on the status of the frame bit, as shown in the following table.

Status of Frame Bit	Today	When Frame Bit Is Removed
Sample-based	3-D array with samples in time in the third dimension	3-D array with samples in time in the third dimension

Status of Frame Bit	Today	When Frame Bit Is Removed
Frame-based	2-D array with frames in time concatenated in the first dimension	3-D array with samples in time in the third dimension

This warning advises you to switch your **Signal logging format** to `Dataset`. The `Dataset` logging mode logs all 2-D signals as 3-D arrays, so its behavior is not dependent on the status of the frame bit.

When you get the warning message, to continue logging signals as a 2-D array:

1 Select **Simulation > Configuration Parameters > Data Import/Export**, and change **Signal logging format** to `Dataset`. To do so for multiple models, click the link provided in the warning message.

2 Simulate the model.

3 Use the `dsp.util.getLogsArray` function to extract the logged signal as a 2-D array.

Frame-Based Processing: Model Reference

In this release, the Model block has been updated so that its operation does not depend on the frame status of its input signals.

Compatibility Considerations

In a future release, signals will not have a frameness attribute, therefore models that use the Model block must be updated to retain their behavior. If you are using a model with a Model block in it, follow these steps to update your model:

1 For both the child and the parent models:

- In the **Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
- Change the **Block behavior depends on input frame status** parameter to `warning`.

2 For both the child and the parent models, run the Simulink Upgrade Advisor. For details, see "Consult the Upgrade Advisor".

3 For the child model only:

- In the **Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
- Change the **Block behavior depends on input frame status** parameter to `error`.

Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay

This release phases out support for buses with mixed sample and frame-based elements on the Unit Delay and Delay blocks in Simulink. When the frame bit is removed in a future release, any Delay block that has a bus input of mixed frameness will start producing different results. This incompatibility is phased over multiple releases. In R2012a the blocks will start warning. In a future release, when the frame bit is removed, the blocks will error.

Block Enhancements

Delay Block Accepts Buses and Variable-Size Signals at the Data Input Port

In R2012a, the Delay block provides the following support for bus signals:

- The data input port u accepts virtual and nonvirtual bus signals. The other input ports do not accept bus signals.
- The output port has the same bus type as the data input port u for bus inputs.
- Buses work with:
    - Sample-based and frame-based processing
    - Fixed and variable delay length
    - Array and circular buffers

To use a bus signal as the input to a Delay block, you must specify the initial condition in the dialog box. In other words, the initial condition cannot come from the input port x0.

In R2012a, the Delay block also provides the following support for variable-size signals:

- The data input port u accepts variable-size signals. The other input ports do not accept variable-size signals.
- The output port has the same signal dimensions as the data input port u for variable-size inputs.

The rules for variable-size signal support depend on the input processing mode of the Delay block. See the block reference page for details.

n-D Lookup Table Block Has New Default Settings

In R2012a, the default values of the **Table data** and **Breakpoints 3** parameters of the n-D Lookup Table block have changed:

- **Table data** — reshape(repmat([4 5 6;16 19 20;10 18 23],1,2),[3,3,2])
- **Breakpoints 3** — [5, 7]

The default values of all other block parameters remain the same.

Blocks with Discrete States Can Specify Custom Storage Classes in the Dialog Box

In R2012a, the following blocks have additional parameters on the **State Attributes** tab to support specification of custom storage classes:

- Data Store Memory
- Delay
- Discrete Filter
- Discrete State-Space
- Discrete Transfer Fcn
- Discrete Zero-Pole
- Discrete-Time Integrator
- Memory
- PID Controller
- PID Controller (2 DOF)
- Unit Delay

In previous releases, specifying a custom storage class for a block required creating a signal object in the base workspace. In R2012a, you can specify the custom storage class on the **State Attributes** tab of the block dialog box.

Inherited Option of the Input Processing Parameter Now Provides a Warning

Some Simulink blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will have a new parameter that allows you to specify the appropriate processing behavior. To prepare for this change, many blocks received a new **Input processing** parameter in previous releases. See Version 7.8 (R2011b) Simulink Software for details. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending on the type of processing you want. The third choice, `Inherited`, is a temporary selection that is available to help you migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

In this release, your model will provide a warning for the following blocks when the **Input processing** parameter is set to `Inherited`, the input signal is frame-based, and the input signal is a vector, matrix, or N-dimensional array:

- Unit Delay
- Delay
- Bias
- Tapped Delay

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the slupdate function. The function detects all blocks that have `Inherited` selected for the **Input processing** parameter, and asks you whether you would like 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)`.

In a future release, the frame bit and the `Inherited` option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Also, after the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, upgrade your existing models using `slupdate` as soon as possible.

User Interface Enhancements

Model Advisor: Highlighting

When a Model Advisor analysis is complete, you can specify that the Model Advisor highlight blocks in a model diagram relevant to warning and failure conditions reported for individual Model Advisor checks. When you click a check, in the model window you can easily see which objects pass, receive a warning, or fail the check.

See Consulting the Model Advisor.

Model Explorer: Grouping Enhancements

In the object property table, you can now group data objects by the object type property. (As in earlier releases, you can also group data objects by other property columns.)

1   Right-click the empty heading in the first column (the column that displays icons such as the block icon (⬚)).

2   In the context menu, select **Group By This Column**.

The object property table also displays the number of objects in each group.

For details about grouping, see Grouping by a Property.

Model Explorer: Row Filter Button

You can access the row filter options by using the new **Row Filter** button, which is to the right of the object count at the top of the **Contents** pane.
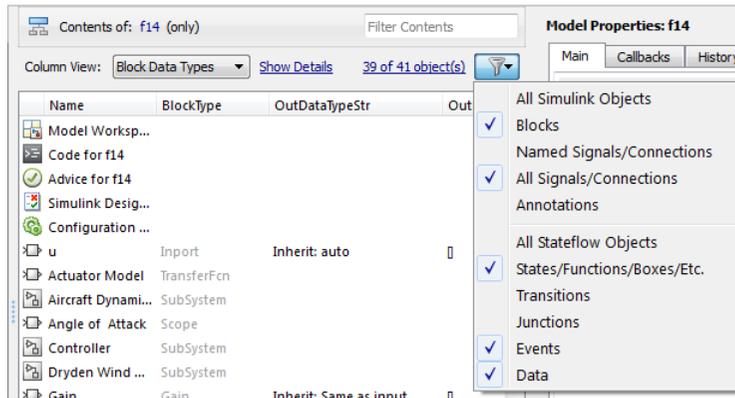
As an alternative, you also can access the row filter options by selecting **View** > **Row Filter**.

For details, see Using the Row Filter Option.

Simulation Data Inspector Enhancements

Signal Data Organization

In R2012a, the **Group Signals** option allows you to customize the organization of the signal data in the Signal Browser table. By default, the data is first grouped by Run Name. You can then group the signal data by model hierarchy or by the logged variable name. Choose options that help you more easily find signals for viewing or comparing. For more information, see Modify Grouping in Signal Browser Table.

Block Name Column

The Signal Browser Table now includes a **Block Name** column. For the signal data, the **Block Name** column displays the name of the block that feeds the signal. To add this column to the table, right-click the Signal Browser table, and from the **Columns** list, select Block Name. For more information, see Add/Delete a Column in the Signal Browser Table.

Plot Check Box Moved

In the Signal Browser table, the plot check box is no longer in a separate **Plot** column. To select a signal for plotting, go to the left-most column where the plot check box is now located.

Parallel Simulation Support

The Simulation Data Inspector API now works with parallel simulations using the `parfor` command. To use the Simulation Data Inspector to record and view the results from parallel simulations, you can use the following methods to get and set the location of the Simulation Data Inspector repository:

- Simulink.sdi.getSource
- Simulink.sdi.setSource
- Simulink.sdi.refresh

For more information, see Record Data During Parallel Simulations

Port Value Displays

The behavior of port value displays for blocks has changed. In addition to performance improvements, the changes include:

- Port values are now port-based instead of block-based.
- **Block Output Display Options** dialog box has been changed to **Value Label Display Options**.
- **Show none** display option name has been changed to **Remove All**.
- You can now right-click a signal line and select **Show Port Value**. In previous releases, you enable port value displays only through the **Block Output Display Options** dialog box.
- The port value display is an empty box you toggle or hover on a block and have not yet run the simulation. In previous releases, it displayed `xx.xx`.
- The port value displays the string `wait` when you toggle or hover on a block that Simulink has optimized out of the simulation.

For more information, see Displaying Port Values in the Simulink User's Guide.

Modeling Guidelines

Modeling Guidelines for High-Integrity Systems

Following are the new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance
- hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance
- hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
- hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance
- hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance
- hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance
- hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance
- hisl_0403: Use of char data type improve MISRA-C:2004 compliance

For more information, see Modeling Guidelines for High-Integrity Systems.

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow

The MathWorks Automotive Advisory Board (MAAB) working group created Version 2.2 of the MAAB Guidelines Using MATLAB, Simulink, and Stateflow. For more information, see MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow.

Execution on Target Hardware

New Feature for Running Models Directly from Simulink on Target Hardware

Use the new Run on Target Hardware feature to automatically run a Simulink model on target hardware.

The feature supports the following target hardware:

- BeagleBoard™
- LEGO MINDSTORMS NXT™

As part of the Run on Target Hardware feature, use the Target Installer to download and install support for your target hardware. To start the Target Installer, enter `targetinstaller` in a MATLAB Command Window, or open a Simulink model and select **Tools** > **Run on Target Hardware** > **Install/Update Support Package...**.

When the Target Installer utility has finished, demos and a block library for your target hardware are available.

To view the demos, enter `doc` in the MATLAB Command Window. In the product help window that opens, look for **Other Demos** near the bottom, under the Contents tab.

To view the block library, enter `simulink` in the MATLAB Command Window. This action will launch the Simulink Library Browser. When the Simulink Library Browser opens, look for one of the following block libraries:

- Target for Use with BeagleBoard Hardware
- Target for Use with LEGO MINDSTORMS NXT Hardware

For more information, you can read the documentation for Running Models on Target Hardware.

R2011b

Version: 7.8

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Accelerator Mode Now Supports Algebraic Loops

The Accelerator mode now works with models that contain algebraic loops. In previous releases, using Accelerator mode for models that contained algebraic loops returned error messages.

Component-Based Modeling

For Each Subsystem Support for Continuous Dynamics

For Each Subsystem blocks support continuous dynamics. This feature simplifies modeling a system of identical plant models.

The continuous dynamics support includes:

- Non-trigger sample time, including multi-rate and multitasking
- Continuous states
- Algebraic loops
- Blocks in the SimDriveline™, SimElectronics®, and SimHydraulics® products

To see an example using continuous dynamics with a For Each Subsystem block, run the `sldemo_metro_foreach` demo.

Enable Port as an Input to a Root-Level Model

You can add an Enable port to the root level of a model. The referenced model can also use a Trigger port.

Using a root-level Enable port takes advantage of model referencing benefits, without your having to do either of these extra steps:

- Put all blocks in an enabled subsystem into a referenced model
- Put the entire enabled subsystem in a referenced model

Compatibility Considerations

When you add an enable port to the root-level of a model, if you use the **File** > **Save As** option to specify a release before R2011b, then Simulink replaces the enable port with an empty subsystem.

Finder Option for Looking Inside Referenced Models

The Finder tool has a new **Look inside referenced models** option that allows you to search within a model reference hierarchy.

For details, see Specifying Kinds of Systems To Search.

Improved Detection for Rebuilding Model Reference Targets

To determine when to rebuild model reference simulation and Simulink Coder targets, Simulink uses structural checksums of file contents. The use of checksums provides more accurate detection of file changes that require a rebuild. Checksum checking is particularly valuable in environments that store models in content management systems.

For details, see Rebuild.

Model Reference Target Generation Closes Unneeded Libraries

When building model reference simulation and Simulink Coder targets, Simulink opens any unloaded libraries necessary for the build. Before R2011b, Simulink did not close the libraries that it opened during the build process.

In R2011b, Simulink closes all libraries no longer needed for target generation or simulation. Simulink leaves the following kinds of libraries open:

- Libraries used by referenced models running in Normal mode
- Libraries that were already open at the start of the target generation process

Concurrent Execution Support

This release extends the modeling capabilities within the Simulink product to capture and simulate the effects of deploying your design to multicore systems. In addition, you can deploy your designs to an actual multicore system using Simulink Coder, Embedded Coder, and Simulink Real-Time software. You can:

- Create a new model configuration or extend existing configurations for concurrent execution.
- Use Model blocks to define potential opportunities for concurrency in your design.
- Easily set up and configure concurrent on-target tasks using a task editing interface.
- Use either the GUI or command-line APIs to iteratively map design partitions (based on `Model` blocks) to tasks to find optimal concurrent execution scenarios.
- Generate code that leverages threading APIs for Windows, Linux, VxWorks®, and Simulink Real-Time platforms for concurrent on-target execution.

For further information, see Configuring Models for Targets with Multicore Processors in the Simulink User's Guide.

Finer Control of Library Links

Libraries and links have been enhanced with the following features:

- New option to lock links to libraries. Lockable library links enable control of end user editing, to prevent unintentional disabling of these links. This feature ensures robust usage of mature stable libraries.
- New check for read-only library files when you try to save, and option to try to make library writable.
- New options in Links Tool to push or restore individual edited links, in addition to existing option to push or restore entire hierarchies of links.
- `get_param` and `set_param` enhanced to perform loading of library links, making programmatic manipulation of models easier and more robust. For example, Simulink now loads links consistently if you use either `load_system` or `open_system` before using `get_param`.

For details, see Working with Library Links in the Simulink documentation.

Mask Built-In Blocks with the Mask Editor

You can now mask built-in blocks with the Mask Editor to provide custom icons and dialogs. In previous releases, you could mask only Subsystem, Model, and S-Function blocks. Now, in the Mask Editor, you can choose to promote any underlying parameter of any block to the mask. For subsystems, you can choose to promote parameters from any child blocks. You can associate a single mask parameter with multiple promoted parameters if they are of the same type. Changing the value of the mask parameter also sets the value of the associated promoted parameters.

You cannot mask blocks that already have masks. For example, some Simulink blocks, such as Ramp and Chirp Signal in the Sources library, cannot be masked.

For details, see Masking Blocks and Promoting Parameters in the Simulink documentation.

Parameter Checking in Masked Blocks

Masked blocks now prevent you entering invalid parameter values by reporting an error when you edit the mask dialog values. Now the parameter checking behavior of built-in and masked blocks is unified. Both block types check for valid parameter values when

you change block dialog values. Parameter checking at edit time prevents you saving a model with syntax errors or invalid values.

Previously only built-in blocks reported an error at the time you enter an invalid parameter with a syntax error, but masked blocks accepted invalid values. In previous releases you could enter invalid values in masked block dialogs and not see an error until you compiled the model. If the model was not compiled you could save a model with syntax errors or other invalid values. In R2011b, parameter checking prevents this problem.

Parameter checking applies both in the mask dialog and at the command line for invalid parameters due to syntax errors (e.g. a blank parameter or invalid parameter names). Parameter checking only applies in the mask dialog for errors defined by the block. Blocks can define valid parameters, for example, the upper limit must be higher than the lower limit, or the frequency of a signal cannot be negative etc. This type of parameter checking does not apply to changes you make at the command line. This allows you to set up blocks with multiple calls to `set_param`, without requiring that each step checks for errors.

Menu Options to Control Variants

You can now select or open Model Variants and Variant Subsystems with the **Edit** and context menus. You can use the menus to open any variant choice or override the block using any variant choice. These options were previously accessible only by opening the block dialog boxes.

For details, see Modeling Variant Systems in the Simulink documentation.

MATLAB Function Blocks

Simulation Supported When the Current Folder Is a UNC Path

In R2011b, you can simulate models with MATLAB Function blocks when the current folder is a UNC path. In previous releases, simulation of those models required that the current folder not be a UNC path.

Simulink Data Management

Default Design Minimum and Maximum are []/[], Not -inf/inf

In R2011b, the default design minimum and maximum values for `Simulink.Signal`, `Simulink.Parameter`, `Simulink.BusElement`, and all blocks are `[]`/`[]` instead of the previous default `-inf`/`inf`. You can no longer specify design minimum and maximum values of `-inf`/`inf` for blocks and these data objects.

Compatibility Considerations

Simulink generates a warning or error depending on the scenario that led to `-inf`/`inf` being specified as design minimum and maximum values. The following scenarios are possible.

- When a Simulink data object is loaded from an old MAT-file or MATLAB file in which the design maximum and minimum values of the data object were specified as `-inf`/`inf`, Simulink generates a warning that `-inf`/`inf` is not supported and changes the design values to the new default, namely, `[]`/`[]`.
- If you set the design minimum and maximum values for the above mentioned data objects as `-inf`/`inf`, Simulink generates a warning that `-inf`/`inf` is not supported and changes the design values to the new default, namely, `[]`/`[]`.
- If the design minimum and maximum values evaluate to `-inf`/`inf` during compilation or at run-time, Simulink generates an error that `-inf`/`inf` is not supported.
- If your model contains an embedded signal object with design minimum and maximum values specified as `-inf`/`inf`, Simulink generates a warning that `-inf`/`inf` is not supported.

Bus Elements Now Have Design Minimum and Maximum Properties

In previous releases, you could specify design minimum and maximum for any data, including data with a bus data type. This was done by specifying the minimum and maximum parameters on the associated blocks or data objects.

In R2011b, you can specify design minimum and maximum for each element of a bus object. You can use this capability to check the values of the corresponding data elements during update diagram and simulation. With this change, Simulink no longer checks minimum or maximum specified on block dialogs or data objects for the whole bus.

Compatibility Considerations

If you specify the minimum or maximum for bus data on block dialogs or data objects, even if these values are scalar, Simulink generates a warning and does not use the minimum or maximum for checking the values of the corresponding data elements.

Compiled Design Minimum and Maximum Values Exposed on Block Inport and Outport

In R2011b, you can view the compiled design minimum and maximum values at a block outport from the Model Editor. See Design Ranges. In addition, you can access the compiled design minimum and maximum values for a block's inport and outport from the command line. See Common Block Parameters.

Command-Line Interface for Accessing Compiled Design Minimum and Maximum

Use parameters `CompiledPortDesignMax` and `CompiledPortDesignMin` to access the design minimum of port signals at compile time. You must place the model in the compile state before querying this parameter. For example, to obtain the compiled design minimum at the outport of a block, use the following set of commands:

```
feval(gcs, [],[],[],'compile');
ports = get_param(gcb,'PortHandles');
outportMax = get_param(ports.Outport, 'CompiledPortDesignMax')
feval(model, [],[],[],'term');
```

`CompiledPortDesignMax` and `CompiledPortDesignMin` return different values depending on the type of signal.

- `[]` if none of the signals has compiled minimum or maximum
- scalar if all signals have the same specified compiled minimum or maximum
- cell array for Mux signals
- when the model is set to strict bus mode: structure for bus signals
- when the model is not set to strict bus mode: `[]` for virtual bus signals

Back-Propagated Minimum and Maximum of Portion of Wide Signal Are Now Ignored

In previous releases, Simulink back-propagated the design minimum and maximum of a portion of a wide signal to the source port of that portion. The back-propagated design minimum and maximum were used in range checking.

In R2011b, Simulink generates a warning and ignores the back-propagated design minimum and maximum of a portion of a wide signal during range checking.

If you want to use the back-propagated design minimum and maximum for range checking of a portion of a wide signal, insert a **Signal Conversion** block with its **Output** parameter set to `Signal copy` in front of that portion.

Easier Importing of Signal Logging Data

You can load logged signal data into a model more easily in R2011b.

You can load elements of a Simulink.SimulationData.Signal object. When you set the **Configuration Parameters** > **Data Import/Export** > **Signal logging format** parameter to `Dataset`, the signal logging output includes `Simulink.SimulationData.Signal` objects. You can then use the `Simulink.SimulationData.Dataset.getElement` method to specify signal elements for the **Configuration Parameters** > **Data Import/Export** > **Input** parameter.

For an example of loading logged signal data into a model, open the `sldemo_mdlref_bus` demo. For more information, see Importing Signal Logging Data.

Partial Specification of External Input Data

You can load external data for a subset of root-level Inport ports, without having to create data structures for the ports for which you want to use ground values.

Using the **Configuration Parameters** > **Data Import/Export** Input parameter, in the comma-separated list, enter an empty matrix to specify ground values for a port.

Using an empty matrix for ports for which you want to use ground values simplifies the specification of external data to input. Also, you can use an empty matrix for an array of buses signal, which you cannot load into a root-level Inport block.

Command-Line Interface for Signal Logging

You can now use the MATLAB command line to perform the same signal logging tasks that you can perform with the Signal Logging Selector tool.

To configure signal logging from the command line, use methods for the following classes:

Simulink.SimulationData Class	Signal Logging Configuration Component
ModelLoggingInfo	Signals to log for a given simulation. Use to override the logging settings stored within a given model or referenced model.
SignalLoggingInfo	Logging settings for a single signal within a model.
LoggingInfo	Collection of signal logging properties. Use to change logging settings, such as decimation, for a signal.

For more information, see Command-Line Interface for Overriding Signal Logging Settings.

Access to the Data Import/Export Pane from the Signal Logging Selector

The Signal Logging Selector toolbar includes a button (🐾) to open the **Configuration Parameters** > **Data Import/Export** pane. Use the **Data Import/Export** pane to configure the export of output signal and state data to the MATLAB® workspace during simulation.

Inexact Property Names for User-Defined Data Objects Will Not Be Supported in a Future Release

In previous releases, you could access a property of a user-defined data object using an inexact property name. For example, after creating a `Simulink.Parameter` data object

```
a = Simulink.Parameter;
```
you could set the property `Value` of the data object by using the following command.

```
a.v = 5;
```
In R2011b, Simulink generates a warning if you access a property of a user-defined data object using an inexact property name. While Simulink accesses the property using the inexact match, support for this type of matching will be removed in a future release.

Based on the example above, set the `Value` of the data object using the following command instead.

```
a.Value = 5;
```

Alias Types No Longer Supported with the slDataTypeAndScale Function

Simulink no longer supports calls to `slDataTypeAndScale` when:

- The first argument is a `Simulink.AliasType` object
- The first argument is a `Simulink.NumericType` object with property `IsAlias` set to true

Compatibility Considerations

If your model calls the internal function `slDataTypeAndScale`, you might encounter a compilation error for this model even though it previously compiled successfully. In this case, follow the advice of the error message to update your model to remove the call to `slDataTypeAndScale`.

Simulink.StructType Objects Will Not Be Supported in a Future Release

In a future release, support for `Simulink.StructType` objects will be removed. Use structured parameters or arrays of buses instead.

Old Block-specific Data Type Parameters No Longer Supported

In R2011b, Simulink generates a warning if you try to access any of these old block-specific data type parameters: `DataType`, `DataTypeMode`, `DataTypeScalingMode`, and `Scaling`. In a future release, support for these data type parameters will be removed. Use `DataTypeStr` instead.

Simulink.Signal and Simulink.Parameter Will Not Accept Input Arguments

Simulink generates an error if you pass an input argument to the classes `Simulink.Signal` and `Simulink.Parameter`.

Compatibility Considerations

`Simulink.Signal` and `Simulink.Parameter` classes accepted input arguments in previous versions. However, the arguments were ignored for both classes.

Data Import/Export Pane Changes

The following parameters of the **Configuration Parameters** > **Import/Export** pane have changed to improve their usability.

Pre-R2011b Parameter Name	Changed R2011b Parameter Name
Signal Logging Selector	**Configure Signals to Log**
Return as single object	**Save simulation output as single object**
Inspect signal logs when simulation is paused/stopped	**Record and inspect simulation output**

Simulation Data Inspector Tool Replaces Time Series Tool

The Simulation Data Inspector is now the default browser for logged simulation results. Use the Simulation Data Inspector for viewing all Simulink logged data results, including as a replacement for the Time Series tool.

Compatibility Considerations

In R2011b, the Time Series tool (tstool) no longer supports Simulink data results.

Simulink File Management

Project Management

Organize large modelling projects with new Simulink Projects. Find all your required files, manage and share files, settings, and user-defined tasks, and interact with source control.

Projects can promote more efficient team work and local productivity by helping you:

- Find all the files that belong with your project
- Share projects using integration with external source control tool Subversion
- View and label modified files for peer review workflows
- Create standard ways to initialize and shutdown a project
- Create, store and easily access common operations

You can use projects to manage:

- Your design (.mdl, .m, .mat, and other files, source code for S-functions, data)
- The results or artifacts (simulation results, generated code, logfiles from code generation, reports).
- A set of user-defined actions to use with your project (e.g., run setup code; open models, simulate; build; run shutdown code).
- Change sets of modified files for review and interaction with source control (such as check out, compare revisions, tag or label, and check in)

For more information and a demo project to try, see Managing Projects.

Simulink Signal Management

Signal Conversion Block Enhancements

The **Output** parameter of the Signal Conversion block now has a `Signal copy` option that replaces the pre-R2011b `Contiguous copy` and `Bus copy` options. The `Signal copy` option handles both non-bus and bus input signals, so that you do not need to update the setting if the input signal changes from a non-bus to a bus signal, or from a bus to a non-bus signal.

Also, setting the **Output** parameter to `Nonvirtual bus` enables the **Data type** parameter. You can use the **Data type** parameter to specify a Simulink.Bus object as the output data type for the Signal Conversion block. Using a bus object:

- Eliminates the need to use a `Simulink.Bus` object as the data type of an upstream Bus Creator block.
- Enables you to pass a virtual bus signal from a Bus Selector block and then create a nonvirtual bus signal.

Compatibility Considerations

The `Virtual bus` and `Nonvirtual bus` options for the **Output** parameter continue to work as they did in previous releases.

For models created in a release before R2011b, two compatibility issues can occur. Both of the compatibility issues occur when the Signal Conversion block a virtual bus as its input and has its **Output** parameter set to `Contiguous copy`.

The first compatibility issue occurs if the output of the Signal Conversion block has a Simulink.Signal object associated with it.

- Prior to R2011b, Simulink automatically performed a bus-to-vector conversion and did not report an error.
- If you open the pre-R2011b model in R2011b, then Simulink converts the `Contiguous copy` option setting to `Signal copy` and does not convert the bus signal to a vector. Because you cannot associate a `Simulink.Signal` object with a virtual bus signal, Simulink reports an error.

The second compatibility issue occurs if a virtual bus signal from a Signal Conversion block that has its **Output** parameter set to `Contiguous copy` is input to a Bus Creator block that has a Simulink.Bus object as its output data type.

- Prior to R2011b, Simulink considered the virtual bus signal to be a vector (as in the first compatibility issue), and did not report an error.
- If you open the model in R2011b, Simulink considers the virtual bus signal to be a bus signal. That bus signal and the bus object associated with Bus Creator block are inconsistent, so Simulink reports an error.

To avoid each of these compatibility issues, insert a Bus to Vector block at the input of the Signal Conversion block.

Environment Controller Block Support for Non-Bus Signals

You can use a non-bus signal as an input to the Environment Controller block, even if you set the **Configuration Parameters** > **Diagnostics** > **Connectivity** > **Non-bus signals treated as bus signals** diagnostic to `error`.

Sample Time Propagation Changes

The way that Simulink software propagates sample time has been improved for models with the **Optimization** > **Signals and Parameters** > **Inline parameters** check box cleared (off). This change:

- Reduces the difference in sample time propagation results between when **Inline parameters** is off and on.
- Improves the performance of your model.

Compatibility Considerations

This change is beneficial to the performance of your model. Not all models are affected by the sample time propagation change. To determine if your model is affected, see Sample Time Propagation and Inline Parameters Incompatibility. That page provides guidelines, including information about a script, to help you evaluate your models.

To do this without the script,

- If **Inline parameters** is on for your model, your model is not affected by this change.

- If **Inline parameters** is off in your model, in R2011a or earlier, use the following procedure for each block in your model:

  1 With **Inline parameters** off for your model, select **Edit > Update Diagram**.

  2 Use get_param to collect the `CompiledSampleTime` value of the block.

  3 Turn **Inline parameters** on for your model.

  4 Update the diagram again.

  5 Use get_param to collect the `CompiledSampleTime` value of the block.

  6 Compare the results from steps 2 and 5. If they are different, and the result from step 5 is not `inf`, your model might be affected. To determine for certain if your model is affected, perform steps 1 and 2 in R2011b and compare the results with those of steps 1 and 2 from R2011a or earlier.

If you prefer the sample time propagation results from R2011a and earlier with **Inline parameters** off, you can ensure the desired sample times by manually specifying them on the affected block. If the block does not have a sample time parameter, use the Signal Specification block to specify sample times on the input or output signal.

Frame-Based Processing

In signal processing applications, you often need to process sequential samples of data at once as a group, rather than one sample at a time. Simulink documentation refers to the former as frame-based processing, and to the latter as sample-based processing. A frame is a collection of samples of data, sequential in time.

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 by means of 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 the single line sample-based signal.

Beginning in R2010b, MathWorks started to significantly change the handling of frame-based processing. In the future, frame status will no longer be a signal attribute. Instead, individual blocks will control whether they treat inputs as frames of data or as samples of data. To learn how a particular block handles its input, you can refer to the block reference page.

To make the transition to the new paradigm of frame-based processing, the following Simulink blocks have a new **Input processing** parameter:

- Delay
- Detect Change
- Detect Decrease
- Detect Fall Negative
- Detect Fall Nonpositive
- Detect Increase
- Detect Rise Nonnegative
- Detect Rise Positive
- Difference
- Discrete Derivative
- Transfer Fcn Real Zero
- Unit Delay

You can specify three options with the **Input processing** parameter:

- `Elements as channels (sample-based)`
- `Columns as channels (frame-based)`
- `Inherited`

For more information about R2011b changes relating to frame-based processing, in the DSP System Toolbox release notes, see Frame-Based Processing.

Compatibility Considerations

When you choose the `Inherited` option for the **Input processing** parameter and the input signal is frame-based, Simulink® will generate a warning or error in future releases.

Block Enhancements

New Delay Block That Upgrades the Integer Delay Block

In R2011b, the new Delay block in the Discrete library supports:

- Variable delay length
- Specification of initial condition from input port
- Reset of the state to the initial condition using an external reset signal
- State storage
- Use of a circular buffer instead of an array buffer for state storage

Function Block Parameters: Delay

Delay

Delay input by a fixed or variable number of samples. Based on an external signal, the block can reset its state to the specified initial condition (from dialog or input port). The block supports both circular and array buffer for state storage.

| Main | State Attributes |

Data

	Source	Value	Upper limit
Delay length:	Input por ▼		100
Initial condition:	Input por ▼		

Algorithm

External reset: Rising ▼

Input processing: Elements as channels (sample based) ▼

☐ Use circular buffer for state

☐ Prevent direct feedthrough by increasing delay length to lower limit

☐ Remove protection against out-of-range delay length in generated code

Diagnostic for out-of-range delay length: None ▼

Sample time (-1 for inherited): -1

| OK | Cancel | Help | Apply |

When you open models created in previous releases, the new Delay block replaces each instance of the Integer Delay block, which no longer appears in the Discrete library. The Delay block is an upgrade of the Integer Delay block. Every parameter from the Integer Delay block maps directly to a parameter in the Delay block.

Compatibility Considerations

The following incompatibilities might affect simulation of pre-R2011b models that use the Integer Delay block:

Source of Incompatibility	Behavior in the Integer Delay Block	Behavior in the Delay Block	Rationale	How to Avoid an Error
Initial condition for input signals of N-by-1 or 1-by-N dimensions and sample-based processing	Suppose that delay length is D. For **Initial condition**, the Integer Delay block supports signal dimensions of D-by-N and N-by-D.	For **Initial condition**, the Delay block supports signal dimensions of N-by-1-by-D or 1-by-N-by-D. Using any other format causes an error during simulation.	The Delay block prevents misinterpretation of the dimensions for **Initial condition** by accepting only one format for signal dimensions.	Verify that **Initial condition** uses N-by-1-by-D or 1-by-N-by-D for the format of signal dimensions.
Initial condition for input signals of M-by-N dimensions and sample-based processing	Suppose that the delay length is D. For **Initial condition**, the Integer Delay block supports signal dimensions of D-by-M-by-N, M-by-N-by-D, and M-by-D-by-N.	For **Initial condition**, the Delay block supports signal dimensions of M-by-N-by-D. Using any other format causes an error during simulation.	The Delay block prevents misinterpretation of the dimensions for **Initial condition** by accepting only one format for signal dimensions.	Verify that **Initial condition** uses M-by-N-by-D for the format of signal dimensions.
Sample time	For **Sample time**, the Integer Delay block supports 0. In this case, the block output has continuous sample	Setting **Sample time** to 0 for the Delay block causes an error during simulation.	Because the Delay block belongs to the Discrete library, it should not support continuous sample time.	Use a discrete sample time, or set **Sample time** to −1 to inherit the sample time.

Source of Incompatibility	Behavior in the Integer Delay Block	Behavior in the Delay Block	Rationale	How to Avoid an Error
	time, but fixed in minor time step.			
Rate transition usage	The Integer Delay block handles rate transitions for sample- and frame-based signals.	The Delay block handles rate transitions only for sample-based signals. For frame-based signals, simulation stops due to an error.	This usage of the Delay block is not recommended.	Do not use the Delay block for rate transitions with frame-based signals.

Sqrt and Reciprocal Sqrt Blocks Support Explicit Specification of Intermediate Data Type

In R2011b, both the Sqrt and Reciprocal Sqrt blocks enable specifying the data type for intermediate results. In previous releases, specifying this data type was available for the Reciprocal Sqrt block, but not the Sqrt block.

The Reciprocal Sqrt block now provides additional options for specifying the data type for intermediate results:
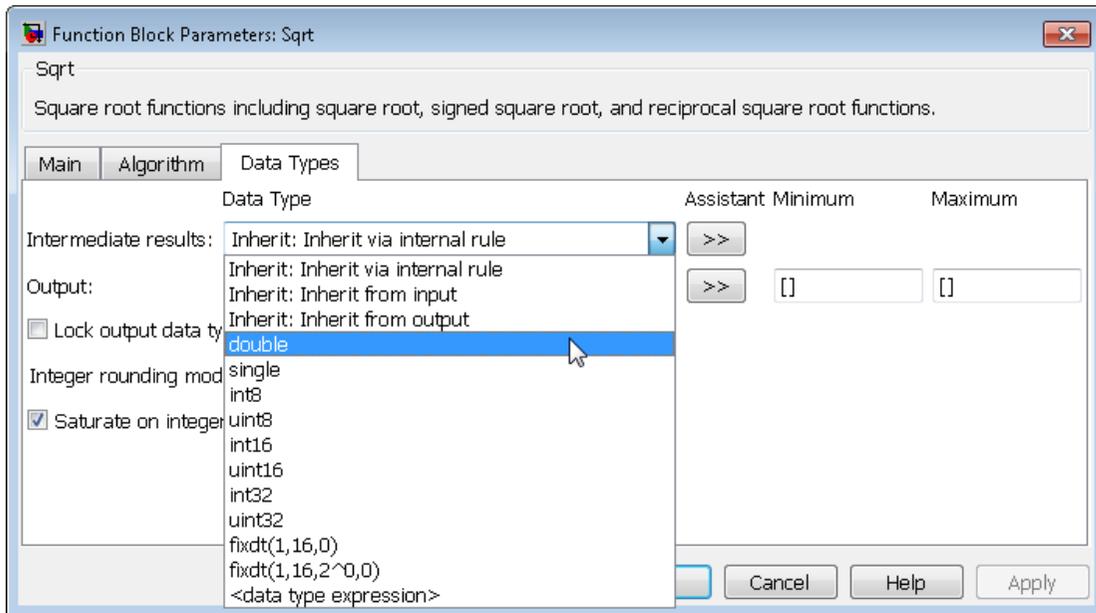
This enhancement enables explicit specification of the data type. In previous releases, specification of this data type was limited to inheritance rules.

For a summary of data type configurations that are valid (input, output, and intermediate results), refer to the block reference page.

Discrete Zero-Pole Block Supports Single-Precision Inputs and Outputs

The Discrete Zero-Pole block now accepts and outputs signals of `single` data type.

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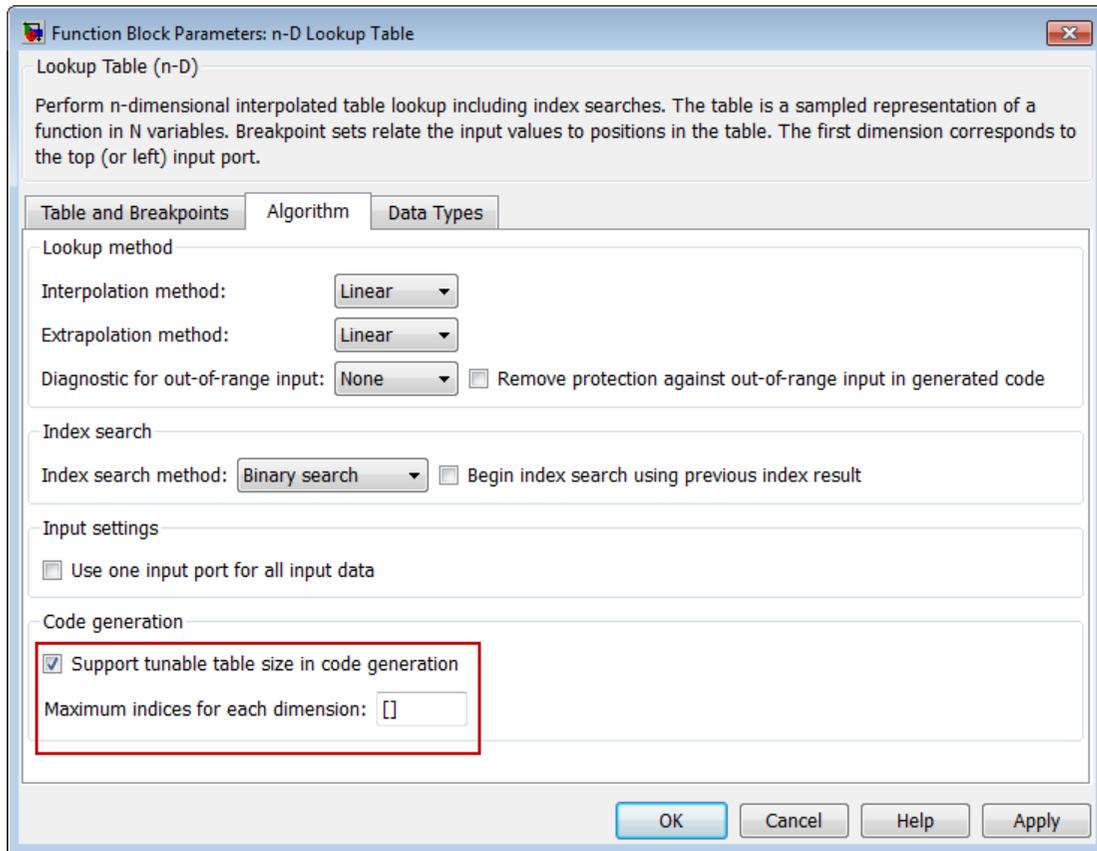
This enhancement enables you to change the size and values of your lookup table and breakpoint data without regenerating or recompiling the code.

Boolean Output Data Type Support for Logic Blocks

The following blocks now enable specification of **Output data type**, which can be `uint8` or `boolean`:

- Detect Change
- Detect Decrease
- Detect Fall Negative
- Detect Fall Nonpositive

- Detect Increase
- Detect Rise Nonnegative
- Detect Rise Positive

This enhancement enables you to specify the output data type to be `boolean`. In previous releases, the blocks always used `uint8` for the output data type.

Derivative Block Parameter Change

The block **Linearization Time Constant s/(Ns + 1)** parameter has changed to **Coefficient c in the transfer function approximation s/(c.s + 1) used for linearization**. Correspondingly, the command-line parameter has changed from `LinearizePole` to `CoefficientInTFapproximation`.

User Interface Enhancements

Model Explorer: First Two Columns in Contents Pane Remain Visible

In the object property table, the behavior of the first two columns (the object icon and the Name property) has changed. These columns now remain visible, regardless of how far you scroll to the right. For an example that illustrates this feature, see Horizontal Scrolling in the Object Property Table.

Model Explorer: Subsystem Code View Added

Model Explorer provides an additional **Column View** option: Subsystem Code. The Subsystem Code view displays Subsystem block code generation properties. For details about views, see The Model Explorer: Controlling Contents Using Views.

Model Explorer: New Context Menu Options for Model Configurations

R2011b provides new context menu options for a model in the Model Hierarchy pane. A new menu option, **Configuration**, organizes previous and new model configuration operations. To view these configuration options, in the Model Hierarchy pane, right-click a model node and select **Configuration**. The following table describes the available configuration options.

To...	Select...
Load an existing configuration set to the model	**Import**
Save the model's active configuration set to a: • .m file (as MATLAB function or script) or • .mat file (Simulink.ConfigSet object)	**Export Active Configuration Set**
Attach a new configuration set to the model	**Add Configuration**
Create a configuration reference and attach it to the model	**Add Configuration Reference**

To...	Select...
Create a concurrent execution configuration set	**Add Configuration for Concurrent Execution**
Convert the model's active configuration set to a configuration reference, which then becomes active for the model	**Convert Active Configuration to Reference**

Two new context menu options are available for a configuration set node under a model node in the Model Hierarchy pane.

To...	Use...
Convert an active configuration set to a configuration reference	**Convert to Configuration Reference** (only enabled for active configuration sets)
Convert a configuration set to a concurrent execution configuration set	**Convert to Configuration for Concurrent Execution**

In R2011b, if a model has an active configuration reference, you can create a copy of the configuration reference for each referenced model. To perform this operation, in the Model Hierarchy pane, right-click the active configuration reference node and select **Propagate to Referenced Models**.

For more information on model configurations, see Managing Model Configurations.

Simulation Data Inspector Enhancements

Command-Line Interface

The Simulation Data Inspector command-line interface is now available to view and compare signal data, and compare two simulation runs of data. For more information, see Record and Inspect Signal Data Programmatically.

Report Generation

Using the Simulation Data Inspector tool or the command-line interface, you can now generate a report of a Simulation Data Inspector session. To generate a report using the GUI, see Create Simulation Data Inspector Report. To generate a report using the command-line interface, see the Simulink.sdi.report function.

Support of Scope, To File, and To Workspace Blocks

The Simulation Data Inspector now supports output from the following blocks:

- Scope and Floating Scope (`Structure with time` and `Array` format)
- To File (`Timeseries` format)
- To Workspace (`Structure with time` format)

Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in Simulink.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a `try`/`catch` statement and performs an action based on a specific error identifier.

For example, the `MATLAB:eigs:NonPosIntSize` identifier has changed to `MATLAB:eigs:RoundNonIntSize`. If your code checks for `MATLAB:eigs:NonPosIntSize`, you must update it to check for `MATLAB:eigs:RoundNonIntSize` instead.

To determine the identifier for a warning, run the following command just after you see the warning in the MATLAB Command Window.

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following command just after you see the error in the MATLAB Command Window.

```
exception = MException.last;
MSGID = exception.identifier;
```

Note: Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs warning-free.

New Modeling Guidelines

Modeling Guidelines for High-Integrity Systems

Following are the new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance
- hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance
- hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance
- hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance

Following are the new high-integrity modeling guidelines for configuration parameter diagnostics:

- hisl_0301: Configuration Parameters > Diagnostics > Compatibility
- hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters
- hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block
- hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization
- hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging
- hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals
- hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses
- hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls
- hisl_0309: Configuration Parameters > Diagnostics > Type Conversion
- hisl_0310: Configuration Parameters > Diagnostics > Model Referencing
- hisl_0311: Configuration Parameters > Diagnostics > Stateflow

For more information, see Modeling Guidelines for High-Integrity Systems.

Modeling Guidelines for Code Generation

Following are the new modeling guidelines for code generation:

- cgsl_0104: Modeling global shared memory using data stores
- cgsl_0105: Modeling local shared memory using data stores

For more information, see Modeling Guidelines for Code Generation.

R2011a

Version: 7.7

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Restore SimState in Models Created in Earlier Simulink Versions

Simulink 7.7 supports the restoring of a SimState from a MAT file saved in a previous version of Simulink. During this operation, Simulink restores as much of the SimState object as possible and automatically resets the simulation start time to the stop time of the SimState object.

You can choose to receive a warning or an error by setting a new diagnostic, **SimState object from earlier release**, on the Diagnostic Pane of the Configuration Parameters dialog.

Improved Absolute Tolerance Implementation

The processing of the absolute tolerance parameter in the Solver configuration pane, and of the absolute tolerance parameters for continuous blocks and S-functions with continuous states, has been enhanced. As a result, these parameters provide a more robust and consistent behavior. These error tolerances are used by variable-step solvers to control integration error for continuous states in a model.

A new SimStruct function `ssSetStateAbsTol` has been introduced to allow for setting the absolute tolerances for the S-Function continuous states in models using a variable-step solver. Use of `ssGetAbsTol` to either get or set absolute tolerances is not recommended. Instead, use `ssGetStateAbsTol` and `ssSetStateAbsTol` to get and set tolerances, respectively.

Component-Based Modeling

Refreshing Linked Blocks and Model Blocks

You can refresh linked blocks and Model blocks in a library or model using the Simulink Editor. Select the **Edit** > **Links and Model Blocks** > **Refresh**.

Refreshing the linked blocks updates the linked blocks to reflect any changes to the original library block. In releases before R2011a, to update linked blocks, you had to take one of the following actions:

- Close and reopen the library that contains the linked blocks that you want to refresh.
- Update the diagram (**Edit** > **Links and Update Diagram** or **Ctrl+D**).

You can update a specific Model block by right-clicking the Model block and selecting **Refresh**.

Compatibility Considerations

The new menu option, **Edit** > **Links and Model Blocks** > **Refresh** menu item replaces **Edit** > **Model Blocks** > **Refresh Model Blocks**. Both the old and new options update Model blocks in the same way.

Enhanced Model Block Displays Variant Model Choices

The Model Variants block now displays model names for all variant choices, making it easier to select and configure available variants.

See Setting Up Model Variants.

Creating a Protected Model Using the Simulink Editor

You can protect a model using the Simulink Editor. Right-click the Model block that references the model for which you want to generate protected model code. In the context menu, select **Code Generation** > **Generate Protected Model**. For details, see Creating a Protected Model.

In earlier releases, you had to use the `Simulink.ModelReference.protect` command to create a protected model.

MATLAB Function Blocks

Embedded MATLAB Function Block Renamed as MATLAB Function Block

In R2011a, Embedded MATLAB Function blocks were renamed as MATLAB Function blocks in Simulink models. The block also has a new look:
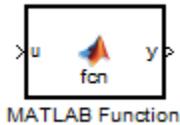
Compatibility Considerations

If you have scripts that refer to Embedded MATLAB library blocks by path, you need to update the script to reflect the new block name. For example, if your script refers to `simulink/User-Defined Functions/Embedded MATLAB Function` or `eml_lib/Embedded MATLAB Function`, change `Embedded MATLAB Function` to `MATLAB Function`.

Support for Buses in Data Store Memory

MATLAB Function blocks now support buses as shared data in Data Store Memory blocks.

Simulink Data Management

Signal Logging Selector

The Signal Logging Selector is a new centralized signal logging tool for:

- Reviewing all signals in a model hierarchy that are configured for logging (set with the Signal Properties dialog box)
- Overriding signal logging settings for specific signals
- Controlling signal logging throughout a model reference hierarchy in a more streamlined way than in previous releases

You can use the Signal Logging Selector with Simulink and Stateflow signals.

To open the Signal Logging Selector, in the **Configuration Parameters** > **Data Import/Export** pane, select the **Signal Logging Selector** button. For a Model block, you can right-click the block and select the **Log Referenced Signals** menu item. (The Signal Logging Selector replaces the Model Reference Signal Logging dialog box.)

See Overriding Signal Logging Settings and Using the Signal Logging Selector to View the Signal Logging Configuration.

Dataset Format Option for Signal Logging Data

You can now select a format for signal logging data. Use the **Configuration Parameters** > **Data Import/Export** > **Signal logging format** parameter to select the format:

- `ModelDataLogs` — Simulink.ModelDataLogs format (default; before R2011a, this format was the only one supported)
- `Dataset` — Simulink.SimulationData.Dataset format

The `Dataset` format:

- Uses MATLAB timeseries objects to store logged data (rather than Simulink.Timeseries and Simulink.TsArray objects). MATLAB `timeseries` objects allow you to work with logging data in MATLAB without a Simulink license.
- Supports logging multiple data values for a given time step, which can be important for Iterator subsystem and Stateflow signal logging.

- Provides an easy-to-analyze format for logged signal data for models with deep hierarchies, bus signals, and signals with duplicate or invalid names.
- Supports the Simulation Data Inspector.
- Avoids the limitations of the `ModelDataLogs` format. For example, for a virtual bus, `ModelDataLogs` format logs only one of multiple signals that share the same source block. For a description of `ModelDataLogs` format limitations, see Bug Report 495436.

To convert a model that contains referenced models to use the `Dataset` format throughout the model reference hierarchy, use the Simulink.SimulationData.updateDatasetFormatLogging function.

If you have logged signal data in the `ModelDataLogs` format, you can use the Simulink.ModelDataLogs.convertToDataset function to convert the `ModelDataLogs` data to `Dataset` format.

To work with `Dataset` format data, you can use properties and methods of the following classes:

- Simulink.BlockPath
- Simulink.SimulationData.BlockPath
- Simulink.SimulationData.Dataset
- Simulink.SimulationData.Signal
- Simulink.SimulationData.DataStoreMemory

For information about the signal logging format, see Specifying the Signal Logging Data Format

From File Block Supports Zero-Crossing Detection

The From File block allows you to specify zero-crossing detection.

Signal Builder Block Now Supports Virtual Bus Output

You can now define the type of output to use on the Signal Builder block now outputs signals. With this release, the Signal Builder block has two options:

- `Ports`

Sends individual signals from the block. An output port named Signal *N* appears for each signal *N*. This option is the default setting. In previous releases, the block uses this type of signal output.

- `Bus`

  Sends single, virtual, nonhierarchical bus of signals from the block. An output port named Bus appears. This `Bus` option enables you to change your model layout without having to reroute Signal Builder block signals. You cannot use this option to create a bus of nonvirtual signals.

For more information, see Defining Signal Output in the Simulink User's Guide

Signal Builder Block Now Shows the Currently Active Group

The Signal Builder block now shows the currently active group on its block mask.

signalbuilder Function Change

The signalbuilder function has a new command, `'annotategroup'`. This command enables the display of the current group name on the Signal Builder block mask.

Range-Checking Logic for Fixed-Point Data During Simulation Improved

The logic that Simulink uses to check whether design minimum and maximum values are within the specified data type range is now consistent with the logic that it uses to calculate best-precision scaling.

- Simulink now checks both real-world values and quantized values for a block parameter, `Simulink.Parameter` object, or `Simulink.Signal` object against design minimum and maximum values. Prior to R2011a, Simulink checked only real-world values against design minimum and maximum values.

- When Simulink checks the design minimum and maximum values for a `Simulink.Signal` object against the data type minimum and maximum values, it obtains the data type range in one of the following ways.

  1 If the data type for a `Simulink.Signal` object is set, Simulink uses the range defined in the specification of that data type

2 If the data type for a `Simulink.Signal` object is set to `auto`, Simulink uses the range for the data type inferred from the initial value of the signal's `fi` object

Prior to R2011a, Simulink only used the data type range defined in the specification of that data type.

- Simulink now checks the run-time parameter value of an S-function against the design minimum and maximum values when the parameter is updated at run-time and during compilation. Prior to R2011a, Simulink checked run-time parameter values of an S-function against the design minimum and maximum only at run-time.

For more information about block parameter range checking, see Checking Parameter Values.

Compatibility Considerations

- An error is generated if the quantized value of a block parameter, `Simulink.Parameter` object, or `Simulink.Signal` object in your model is different from the real-world value and if this difference causes the quantized value to lie outside the design minimum and maximum range.
- An error is generated if the initial value of a `Simulink.Signal` object in your model is a `fi` object and if this initial value is outside the range associated with that `fi` object.
- An error is generated at compile time if the run-time parameter value of an S-function in your model is outside the design minimum and maximum range.

Data Object Wizard Now Supports Boolean, Enumerated, and Structured Data Types for Parameters

In this release, the Data Object Wizard is enhanced to suggest parameter objects for variables with the following data types:

- Boolean
- Enumerations
- Structures

For information, see Working with Data Objects and Data Object Wizard.

Error Now Generated When Initialized Signal Objects Back Propagate to Output Port of Ground Block

Prior to this release, Simulink generated an error when the output of a Ground block was a signal object with an initial value, but did not do the same for such signal objects back propagated to the output port of a Ground block. As of R2011a, Simulink generates an error under both conditions.

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;
.
.
.
```

Global Data Stores Now Treat Vector Signals as One or Two Dimensional

Simulink now uses the **Dimensions** attribute of a source signal object to determine whether to register a global data store as a vector (1-D) or matrix (2-D). For example, if the **Dimensions** attribute of a source signal object is set to [1 N] or [N 1], Simulink registers the global data store as a matrix. Prior to R2011a, Simulink treated all global data stores as vectors.

The following table lists possible signal object dimension settings with what Simulink registers for a corresponding global data store:

Source Signal Object Dimensions	Registered for Global Data Store
−1	Get dimensions from *InitialValue* and interpret vectors as 1-D
N	Vector with N elements
[1 N]	1xN matrix
[N 1]	Nx1 matrix

Compatibility Considerations

If you specify the dimensions of the source signal object for a global data store as [1 N] or [N 1], Simulink now registers the data store as a matrix. Although this change has no impact on numeric results of simulation or execution of generated code, the change can affect the following:

- Propagation of dimensions (for example, signals might propagate as [1 N] or [N 1] instead of N).
- Signal and state logging
  - Vectors are logged as 2D matrices – [*nTimeSteps width*]
  - 2-D matrices are logged as 3-D matrices – [M N *nTimeSteps*]

No Longer Able to Use Trigger Signals Defined as Enumerations

You can no longer use trigger signals that are defined as enumerations. A trigger signal represents an external input that initiates execution of a triggered subsystem. Prior to R2011a, Simulink supported enumerated trigger signals for simulation, but produced

an error during code generation. This change clarifies triggered subsystem modeling semantics by making them consistent across simulation and code generation.

Compatibility Considerations

Use of enumerated trigger signals during simulation now generates an error. To work around this change, compare enumeration values, as appropriate, and apply the resulting Boolean or integer signal value as the subsystem trigger.

Conversions of Simulink.Parameter Object Structure Field Data to Corresponding Bus Element Type Supported for double Only

If you specify the `DataType` field of a `Simulink.Parameter` object as a bus, you must specify `Value` as a numeric structure. Prior to R2011a, Simulink would convert the data types of all fields of that structure to the data types of corresponding bus elements. As of R2011a, Simulink converts the data type of structure fields of type `double` only. If the data type of a field of the structure does not match the data type of the corresponding bus element and is not `double`, an error occurs.

This change does not affect the `InitialValue` field of `Simulink.Signal` objects. Data types of fields of a numeric structure for an initial condition *must* match data types of corresponding bus elements.

Compatibility Considerations

If the data type of a field of a numeric structure that you specify for `Simulink.Parameter` does not match the data type of the corresponding bus element and is not `double`, an error occurs. To correct the condition, set the data types of all fields of the structure to match the data types of all bus elements or set them to type `double`.

For more information, see Simulink.Parameter.

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.

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.

Simulink Signal Management

Data Store Support for Bus Signals

The following blocks support the use of bus and array of buses signals with data stores:

- Data Store Memory
- Data Store Read
- Data Store Write

Benefits of using buses and arrays of buses with data stores include:

- Simplifying the model layout by associating multiple signals with a single data store
- Producing generated code that represents the data store data as structures that reflect the bus hierarchy
- Writing to and reading from data stores without creating data copies, resulting in more efficient data access

For details, see Using Data Stores with Buses and Arrays of Buses.

Compatibility Considerations

Pre-R2011a models that use data stores work in R2011a without any modifications.

To save a model that uses buses with data stores to a pre-R2011a version, you need to restructure that model to not rely on using buses with data stores.

Accessing Bus and Matrix Elements in Data Stores

You can select specific bus or matrix elements to read from or write to a data store. To do so, use the **Element Selection** pane of the Data Store Read block and the **Element Assignment** pane of the Data Store Write block. Selecting bus or matrix elements offers the following benefits:

- Reducing the number of blocks in the model. For example, you can eliminate a Data Store Read and Bus Selector block pair or a Data Store Write and Bus Assignment block pair for each specific bus element that you want to access.
- Faster simulation of models with large buses and arrays of buses.

See Accessing Data Stores with Simulink Blocks.

Array of Buses Support for Permute Dimensions, Probe, and Reshape Blocks

The following blocks now support the use of an array of buses as an input signal:

- Permute Dimensions
- Probe
- Reshape

For details about arrays of buses, see Combining Buses into an Array of Buses.

Using the Bus Editor to Create Simulink.Parameter Objects and MATLAB Structures

You can use the Bus Editor to:

- Define or edit a `Simulink.Parameter` object with a bus object for its data type. In the Bus Editor, select the parameter and use one of these approaches:

  - Select the **File** > **Create/Edit a Simulink.Parameter object** menu item.
  - Click the **Create/Edit a Simulink.Parameter object** icon () from the toolbar.

  You can then edit the `Simulink.Parameter` object in the MATLAB Editor.

- Invoke the `Simulink.Bus.createMATLABStruct` function for a bus object for which you want to create a full MATLAB structure. In the Bus Editor, select the bus object and use one of these approaches:

  - Select the **File** > **Create a MATLAB structure** menu item.
  - Click the **Create a MATLAB structure** icon () from the toolbar.

  You can then edit the MATLAB structure in the MATLAB Editor.

Block Enhancements

Lookup Table, Lookup Table (2-D), and Lookup Table (n-D) Blocks Replaced with Newer Versions in the Simulink Library

In R2011a, the following lookup table blocks have been replaced with newer versions, which differ from the previous versions as follows:

Block	Enhancements to the Previous Version	Other Changes
Lookup Table	- Default integer rounding mode changed from `Floor` to `Simplest` - Support for the following features: - Specification of parameter data types different from input or output signal types - Reduced memory use and faster code execution for nontunable breakpoints with even spacing - Cubic-spline interpolation and extrapolation - Table data with complex values - Fixed-point data types with word lengths up to 128 bits - Specification of data types for fraction and intermediate results - Specification of index search method - Specification of diagnostic for out-of-range inputs	- Block renamed as 1-D Lookup Table - Icon changed
Lookup Table (2-D)	- Default integer rounding mode changed from `Floor` to `Simplest` - Support for the following features: - Specification of parameter data types different from input or output signal types - Reduced memory use and faster code execution for nontunable breakpoints with even spacing - Cubic-spline interpolation and extrapolation	- Block renamed as 2-D Lookup Table - Icon changed

Block	Enhancements to the Previous Version	Other Changes
	• Table data with complex values • Fixed-point data types with word lengths up to 128 bits • Specification of data types for fraction and intermediate results • Specification of index search method • Specification of diagnostic for out-of-range inputs • Check box for **Require all inputs to have the same data type** now selected by default	
Lookup Table (n-D)	• Default integer rounding mode changed from `Floor` to `Simplest`	• Block renamed as n-D Lookup Table • Icon changed

When you load models from earlier versions of Simulink that contain the Lookup Table, Lookup Table (2-D), and Lookup Table (n-D) blocks, those versions of the blocks appear. In R2011a, the new versions of the lookup table blocks appear only when you drag the blocks from the Simulink Library Browser into new models.

When you use the add_block function to programmatically add the Lookup Table, Lookup Table (2-D), or Lookup Table (n-D) blocks to a model, those versions of the blocks appear. If you want to add the *new* versions of the blocks to your model, change the source block path for `add_block` as follows:

Block	Old Block Path	New Block Path
Lookup Table	`simulink/Lookup Tables/Lookup Table`	`simulink/Lookup Tables/1-D Lookup Table`
Lookup Table (2-D)	`simulink/Lookup Tables/Lookup Table (2-D)`	`simulink/Lookup Tables/2-D Lookup Table`
Lookup Table (n-D)	`simulink/Lookup Tables/Lookup Table (n-D)`	`simulink/Lookup Tables/n-D Lookup Table`

To upgrade your model to use new versions of the Lookup Table and Lookup Table (2-D) blocks, follow these steps:

Step	Description	Reason
1	Run the Simulink Model Advisor check for **Check model, local libraries, and referenced models for known upgrade issues requiring compile time information**.	Identify blocks that do not have compatible settings with the new 1-D Lookup Table and 2-D Lookup Table blocks.
2	For each block that does not have compatible settings: • Decide how to address each warning. • Adjust block parameters as needed.	Modify each Lookup Table or Lookup Table (2-D) block to make them compatible with the new versions.
3	Repeat steps 1 and 2 until you are satisfied with the results of the Model Advisor check.	Ensure that block replacement works for the entire model.
4	Run the slupdate function on your model.	Perform block replacement with the 1-D Lookup Table and 2-D Lookup Table blocks.

Note that after block replacement, the block names that appear in the model remain the same. However, the block icons match the new ones for the 1-D Lookup Table and 2-D Lookup Table blocks.

Compatibility Considerations

The Model Advisor check groups all Lookup Table and Lookup Table (2-D) blocks into three categories:

- Blocks that have compatible settings with the new 1-D Lookup Table and 2-D Lookup Table blocks
- Blocks that have incompatible settings with the new 1-D Lookup Table and 2-D Lookup Table blocks
- Blocks that have repeated breakpoints

Blocks with Compatible Settings

When a block has compatible parameter settings with the new block, automatic block replacement can occur without backward incompatibilities.

Lookup Method in the Lookup Table or Lookup Table (2-D) Block	Parameter Settings in the New Block After Automatic Block Replacement	
	Interpolation	Extrapolation
Interpolation-Extrapolation	Linear	Linear
Interpolation-Use End Values	Linear	Clip
Use Input Below	Flat	Not applicable

Depending on breakpoint characteristics, the new block uses one of two index search methods.

Breakpoint Characteristics in the Lookup Table or Lookup Table (2-D) Block	Index Search Method in the New Block After Automatic Block Replacement
Not evenly spaced	Binary search
Evenly spaced and tunable	A prompt appears, asking you to select Binary search or Evenly spaced points.
Evenly spaced and not tunable	

The new block also adopts other parameter settings from the Lookup Table or Lookup Table (2-D) block. For parameters that exist only in the new block, the following default settings apply after block replacement:

Parameter in the New Block	Default Setting After Block Replacement
Breakpoint data type	Inherit: Same as corresponding input
Diagnostic for out-of-range input	None

Blocks with Incompatible Settings

When a block has incompatible parameter settings with the new block, the Model Advisor shows a warning and a recommended action, if applicable.

- If you perform the recommended action, you can avoid incompatibility during block replacement.

- If you use automatic block replacement without performing the recommended action, you might see numerical differences in your results.

Incompatibility Warning	Recommended Action	What Happens for Automatic Block Replacement
The **Lookup Method** is `Use Input Nearest` or `Use Input Above`. The new block does not support these lookup methods.	Change the lookup method to one of the following: • `Interpolation - Extrapolation` • `Interpolation - Use End Values` • `Use Input Below`	The **Lookup Method** changes to `Interpolation - Use End Values`. In the new block, this setting corresponds to: • **Interpolation** set to `Linear` • **Extrapolation** set to `Clip` You also see a message that explains possible numerical differences.
The **Lookup Method** is `Interpolation - Extrapolation`, but the input and output are not the same floating-point type. The new block supports linear extrapolation only when all inputs and outputs are the same floating-point type.	Change the extrapolation method or the port data types of the block.	
The block uses small fixed-point word lengths, so that interpolation uses only one rounding operation. The new block uses two rounding operations for interpolation.	None	You see a message that explains possible numerical differences.

Blocks with Repeated Breakpoints

When a block has repeated breakpoints, the Model Advisor recommends that you change the breakpoint data and rerun the check. You cannot perform automatic block replacement for blocks with repeated breakpoints.

Magnitude-Angle to Complex Block Supports CORDIC Algorithm and Fixed-Point Data Types

The Magnitude-Angle to Complex block now supports the following parameters:
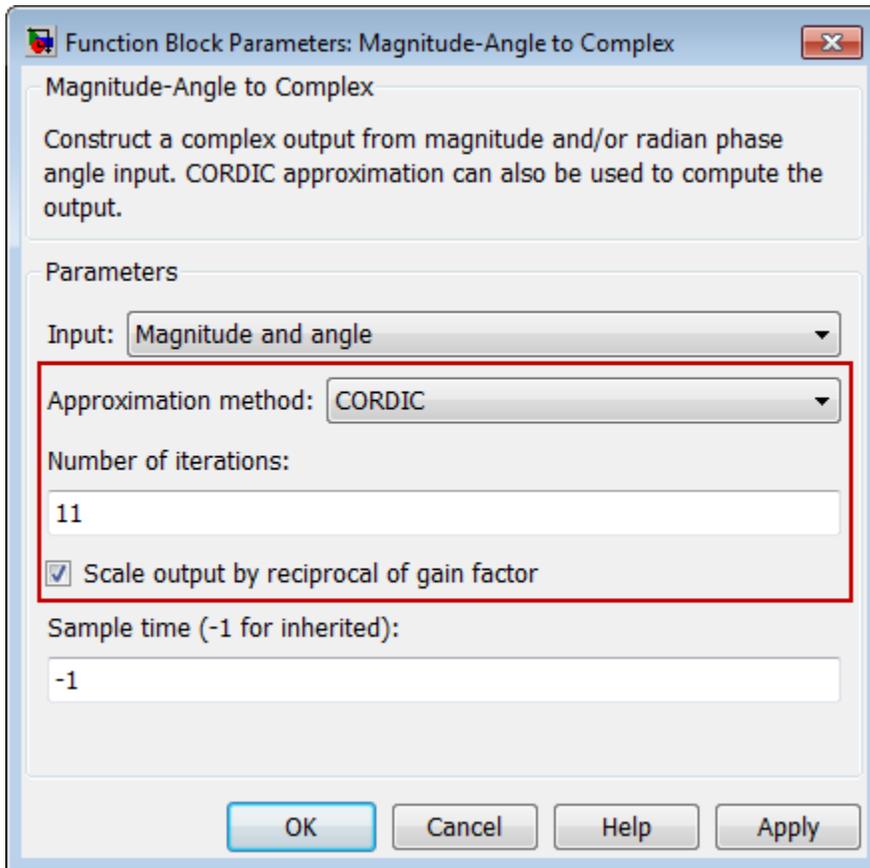
The benefits of the new block parameters are as follows:

New Block Parameter	Purpose	Benefit
Approximation method	Specify the type of approximation the block uses to compute output: None or CORDIC.	Enables you to use a faster method of computing block output for fixed-point and HDL applications.
Number of iterations	For the CORDIC algorithm, specify how many iterations to use for computing block output.	Enables you to adjust the precision of your block output.

New Block Parameter	Purpose	Benefit
Scale output by reciprocal of gain factor	For the CORDIC algorithm, specify whether or not to scale the real and imaginary parts of the complex output.	Provides a more accurate numerical result for the CORDIC approximation.

This block now accepts and outputs fixed-point signals when you set **Approximation method** to `CORDIC`.

Trigonometric Function Block Supports Complex Exponential Output

The Trigonometric Function block now supports complex exponential output: `cos + jsin`. This function works with the CORDIC algorithm.

This block also accepts inputs with unsigned fixed-point data types when you use the CORDIC approximation. In previous releases, only signed fixed-point inputs were supported.

Shift Arithmetic Block Supports Specification of Bit Shift Values as Input Signal

The Shift Arithmetic block now supports specification of bit shift values from an input port. Previously, you could specify bit shift values only on the dialog box. This enhancement enables you to change bit shift values without stopping a simulation.

The block now also supports the following functionality:

Enhancement	Benefit
Specification of diagnostic for out-of-range bit shift values	Flags out-of-range bit shift values during simulation
Option to check for out-of-range bit shift values in the generated code	Enables you to control the efficiency of the generated code

The following parameter changes apply to the Shift Arithmetic block. For backward compatibility, the old command-line parameters continue to work.

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of bits to shift right	**Bits to shift: Number**	`nBitShiftRight`	`BitShiftNumber`
Number of places by which binary point shifts right	**Binary points to shift: Number**	`nBinPtShiftRight`	`BinPtShiftNumber`

The read-only **BlockType** property has also changed from `SubSystem` to `ArithShift`.

Multiple Lookup Table Blocks Enable Removal of Range-Checking Code

When the breakpoint input to a Prelookup, 1-D Lookup Table, 2-D Lookup Table, or n-D Lookup Table block 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.
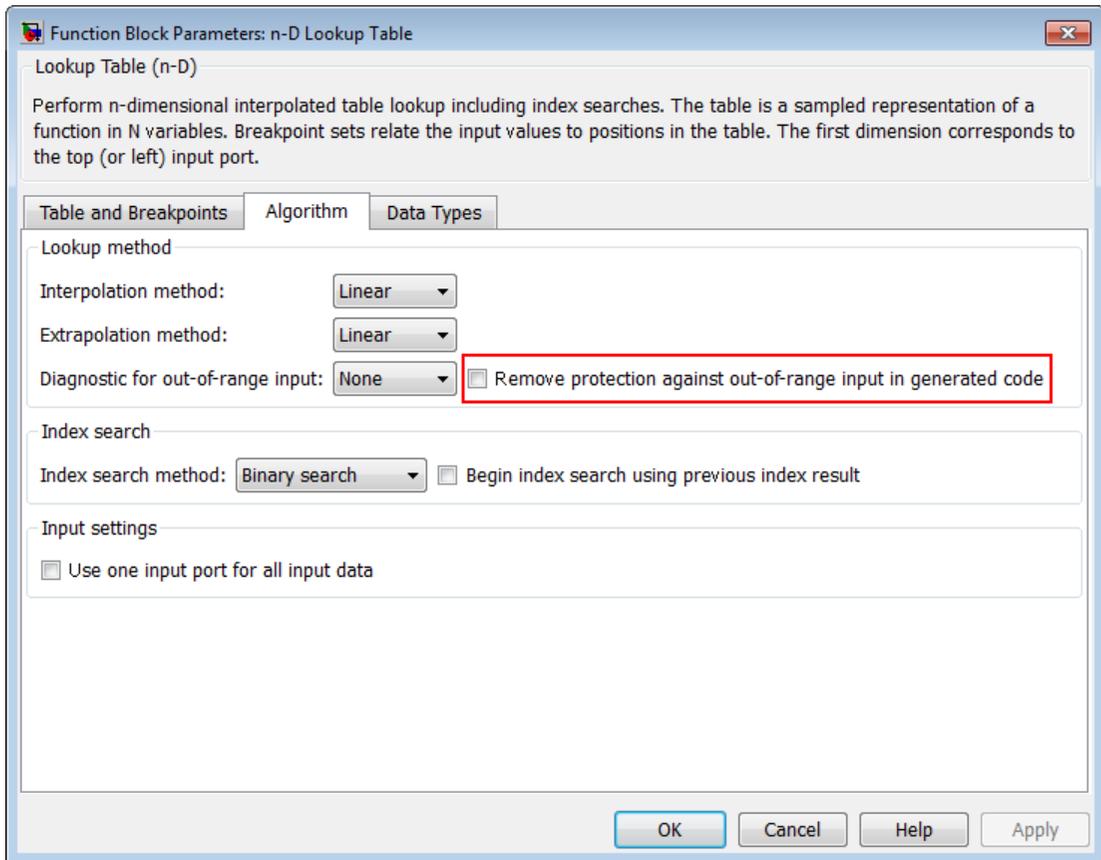
Similarly, when the index input to an Interpolation Using Prelookup block falls within the range of valid index values, you can disable range checking in the generated code. By selecting **Remove protection against out-of-range index in generated code** on the block dialog box, your code can be more efficient.
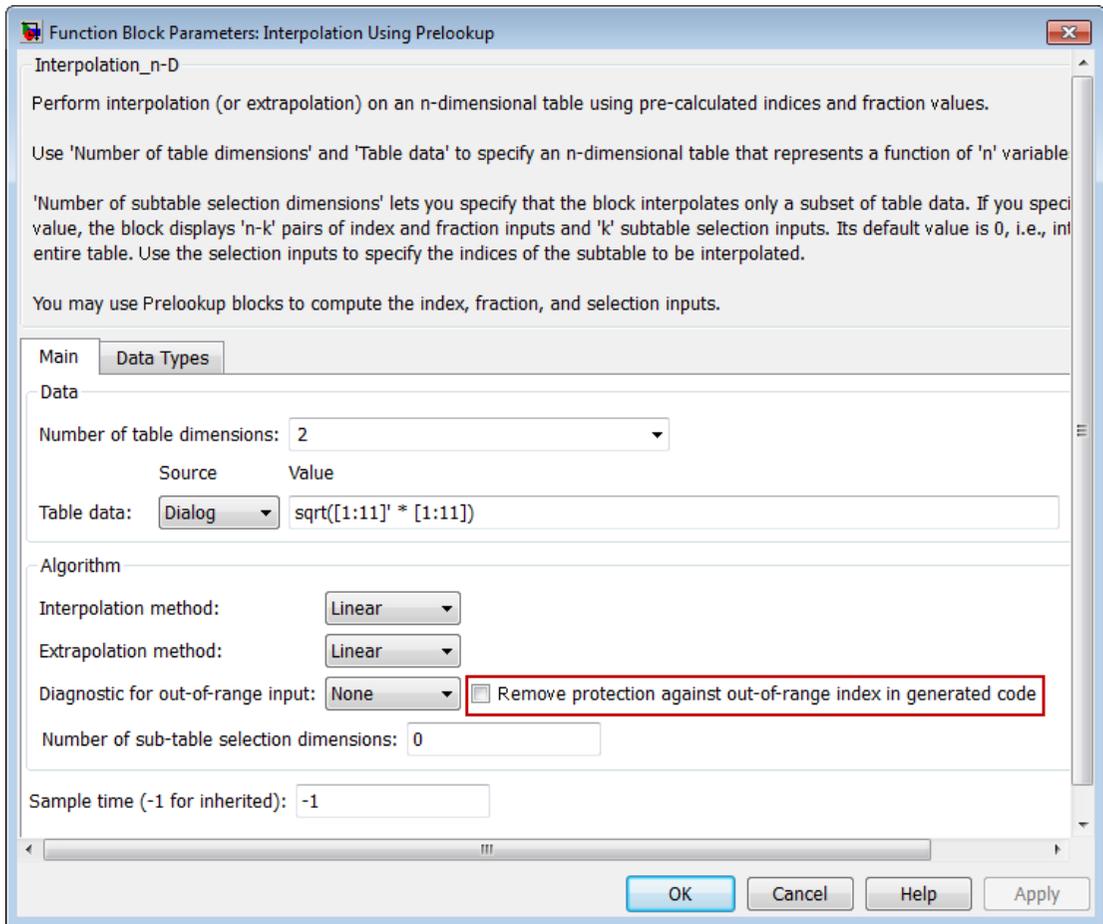
The **Remove protection against out-of-range index in generated code** check box replaces the **Check index in generated code** check box from previous releases. When you load models with the Interpolation Using Prelookup block from previous releases, the following parameter mapping applies:

Parameter Setting from Previous Releases	Parameter Setting for R2011a
Check index in generated code is selected.	**Remove protection against out-of-range index in generated code** is not selected.

Parameter Setting from Previous Releases	Parameter Setting for R2011a
Check index in generated code is not selected.	**Remove protection against out-of-range index in generated code** is selected.

For backward compatibility, the command-line parameter `CheckIndexInCode` continues to work.

Enhanced Dialog Layout for the Prelookup and Interpolation Using Prelookup Blocks

In R2011a, the dialog boxes for the Prelookup and Interpolation Using Prelookup blocks consolidate parameters related to data type attributes on a single tab named **Data Types**. This enhancement enables you to specify data type attributes more quickly on the block dialog box.

- For the Prelookup block, you can now specify breakpoint, index, and fraction attributes on a single tab.
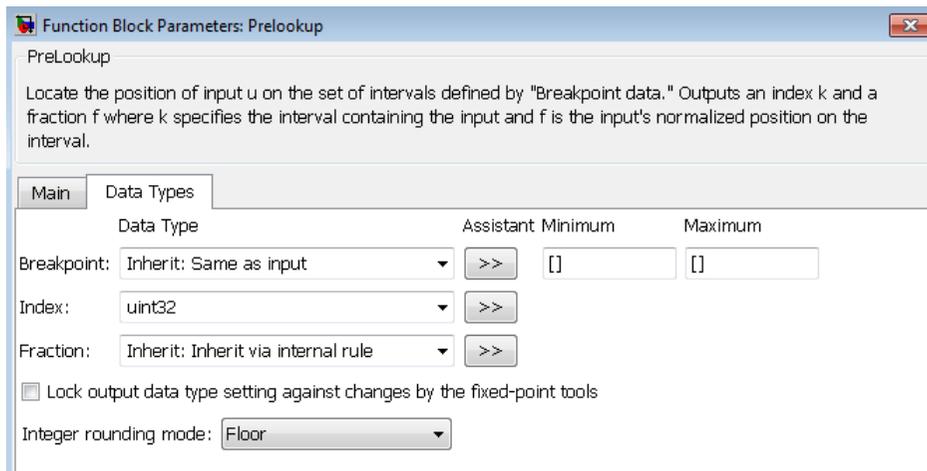
- For the Interpolation Using Prelookup block, you can now specify table, intermediate results, and output attributes on a single tab.
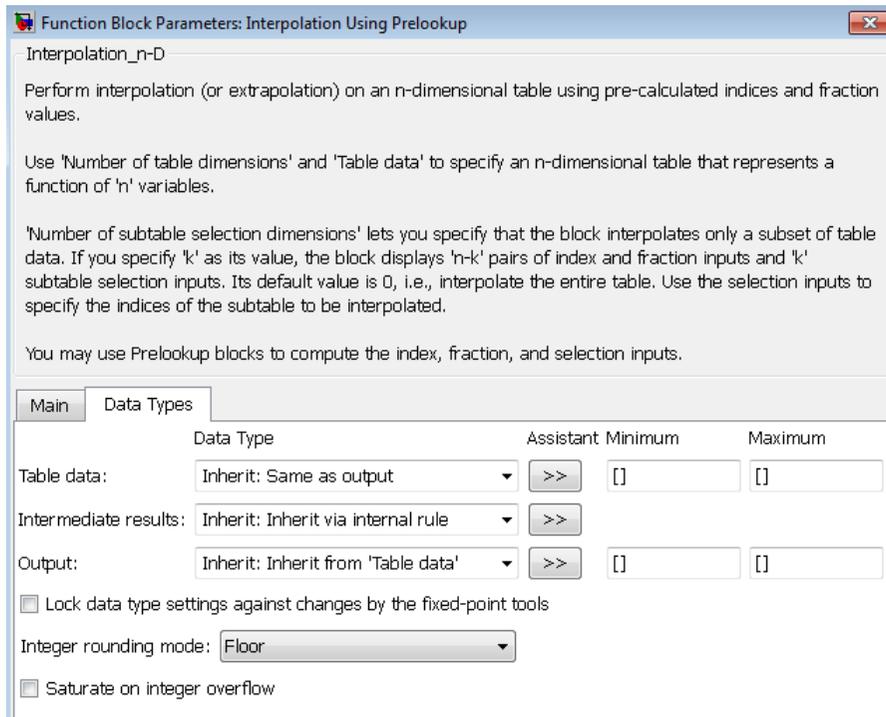
Product of Elements Block Uses a Single Algorithm for Element-Wise Complex Division

In previous releases, the Product of Elements block used two different algorithms for handling element-wise complex division. For example, for a matrix input with four elements (u1, u2, u3, and u4), the following behavior would apply:

- For inputs with built-in integer and floating-point data types, the order of operations was `1/(u1*u2*u3*u4)`.

- For inputs with fixed-point data types, the order of operations was `((((1/u1)/u2)/u3)/u4)`.

Starting in R2011a, the Product of Elements block uses a single algorithm for handling element-wise complex division. For inputs of integer, floating-point, or fixed-point type, the order of operations is always `(((((1/u1)/u2)/u3)/u4)…/uN)`.

Sign Block Supports Complex Floating-Point Inputs

The Sign block now supports complex inputs of type `double` or `single`. The block output matches the MATLAB result for complex floating-point inputs.

When the input `u` is a complex scalar, the block output is:
`sign(u) = u./ abs(u)`

When an element of a vector or matrix input is complex, the block uses the same formula that applies to scalar input.

MATLAB Fcn Block Renamed to Interpreted MATLAB Function Block

In R2011a, the MATLAB Fcn block has been renamed to Interpreted MATLAB Function block. The icon has also changed to match the new block name. However, all functionality and block parameters remain the same. The read-only **BlockType** property is also unchanged.

Existing scripts that use the add_block function to programmatically add the MATLAB Fcn block to models do not require any changes.

When you load existing models that contain the MATLAB Fcn block, the block name that appears in the model remains unchanged. However, the block icon matches the new one for the Interpreted MATLAB Function block.

Environment Controller Block Port Renamed from RTW to Coder

In R2011a, the Environment Controller block has renamed the `RTW` port to `Coder`. This enhancement better reflects the purpose of that input port, which designates signals to pass through the block when code generation occurs for a model.

Block Parameters on the State Attributes Tab Renamed

In R2011a, the block parameters **Real-Time Workshop storage class** and **Real-Time Workshop storage type qualifier** have been renamed to **Code generation storage class** and **Code generation storage type qualifier**, respectively. These two parameters appear on the State Attributes tab of the following block dialog boxes:

- Discrete Filter

- Discrete PID Controller
- Discrete PID Controller (2DOF)
- Discrete State-Space
- Discrete Transfer Fcn
- Discrete Zero-Pole
- Discrete-Time Integrator
- Memory
- Unit Delay

Block Parameters and Values Renamed for Lookup Table Blocks

In R2011a, the **Action for out-of-range input** parameter has been renamed as **Diagnostic for out-of-range input** for the following blocks:

- Direct Lookup Table (n-D)
- Interpolation Using Prelookup
- n-D Lookup Table
- Prelookup

Also, the **Process out-of-range input** parameter has been renamed as **Extrapolation method** for the Prelookup block.

For lookup table blocks that provide **Interpolation method** or **Extrapolation method** parameters, the following changes apply:

Parameter Value from Previous Releases	Parameter Value in R2011a
None - Flat	Flat
None - Clip	Clip

Performance Improvement for Single-Precision Computations of Elementary Math Operations

In R2011a, single-precision computations for elementary math operations are faster. This enhancement applies to the following simulation modes:

- Normal
- Accelerator

Dead Zone Block Expands the Region of Zero Output

In R2011a, the Dead Zone block expands the region of zero output, or the dead zone, to include inputs (U) that equal the lower limit (LL) or upper limit (UL):

Input	Output
U >= LL and U <= UL	Zero
U > UL	U − UL
U < LL	U − LL

In previous releases, the dead zone excluded inputs that equal the lower or upper limit.

Enhanced PID Controller Blocks Display Compensator Formula in Block Dialog Box

The PID Controller and PID Controller (2 DOF) blocks now display the current compensator formula in the block dialog box. This display reflects the current settings for controller type, controller form, and time domain.

Ground Block Always Has Constant Sample Time

In R2011a, the sample time of the Ground block is now constant (inf) regardless of the setting for **Inline parameters** in the Configuration Parameters dialog box.

Compatibility Considerations

Previously, if **Inline parameters** was off, the sample time of the Ground block depended on sample-time propagation. Now, the following conditions hold true:

- Function-call subsystem blocks that have an unconnected function-call port now have the correct sample time of constant (inf) regardless of the setting for **Inline parameters**.

- Function-call subsystem blocks that have a function-call port connected to a Ground block now have the correct sample time of `constant (inf)` regardless of the setting for **Inline parameters**.

- Function-call subsystem blocks that have the **Sample time type** set to `periodic` now correctly error out when they are connected to a Ground block or unconnected.

New Function-Call Feedback Latch Block

The Function-Call Feedback Latch block allows you to break a feedback loop involving data signals between function-call signals. You can use this block for two specific scenarios:

- If a loop involves parent and child function-call blocks (that is, the initiator of the child function-call block is inside the parent function-call block), then place this block on the feedback signal from the child to the parent. You can thus ensure that the value of the signal does not change during execution of the child.
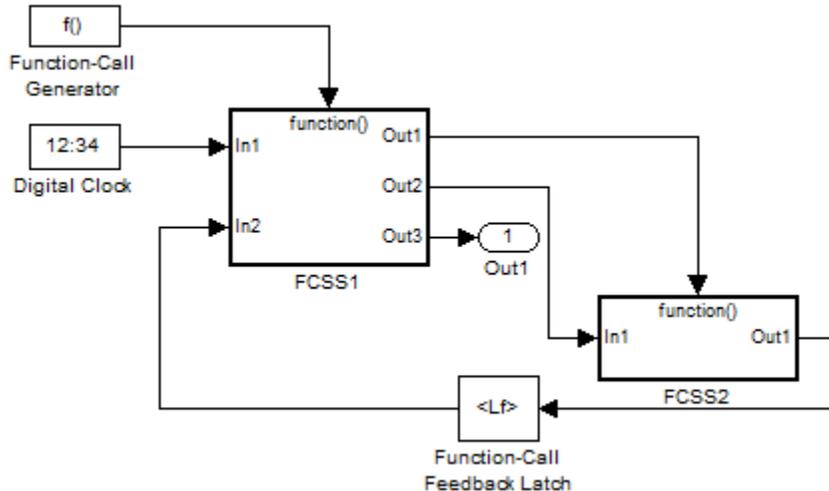
- If a loop involves function-call blocks connected to branches of the same function-call signal, then this block latches the signal at the input of the destination function-call block, and thereby allows it to execute prior to the source function-call block.
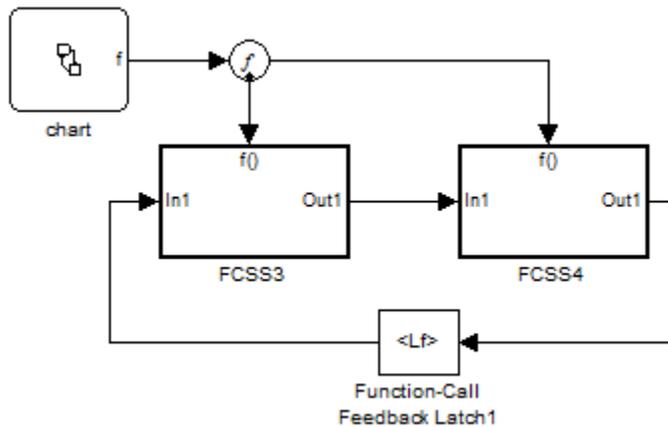
In either case, the latching results in the destination block reading a delayed signal from the previous execution of the source function-call block.

Outport Driving Merge Block Does Not Require IC in Simplified Initializaton Mode

If an Outport block of a conditionally executed subsystem directly drives a Merge block, then the Outport block no longer requires the specification of an Initial Condition (IC) in simplified initialization mode. Simulink still expects the Merge block to specify an IC. This enhancement applies only when the Outport and Merge blocks are in the same model.

Discrete Filter, Discrete FIR Filter, and Discrete Transfer Fcn Blocks Now Have Input Processing Parameter

The Discrete Filter, Discrete FIR Filter, and Discrete Transfer Fcn blocks now have an **Input processing** parameter. This parameter enables you to specify whether the block performs sample- or frame-based processing on the input. To perform frame-based processing, you must have a DSP System Toolbox license.

Model Blocks Can Now Use the GetSet Custom Storage Class

The GetSet custom storage class can now be used for the inports and outports of Model blocks. To assign a GetSet custom storage class to the inport or outport of a referenced model block, use one of the following methods.

1 Assign the GetSet custom storage class to the root-level inport or outport of the referenced model.

2 Assign the GetSet custom storage class to scalar signals entering an inport of the referenced model block in the parent model, provided one of the following conditions is met.

   a The referenced model uses function prototype control to specify that the inport should be passed by value instead of being passed by pointer to the Model block's step function.

   b The inport to which the GetSet custom storage class is assigned should be passed by value.

3 Assign the GetSet custom storage class to a scalar signal leaving one of the outports of the referenced model block in the parent model. In this case, the referenced model must use function prototype control to specify that the outport should be the returned value of the function.

User Interface Enhancements

Model Explorer: Hiding the Group Column

By default, the property column that you use for grouping (the group column) appears in the property table. That property also appears in the top row for each group.

To hide the group column, use one of the following approaches:

- From the **View** menu, clear the **Show Group Column** check box.
- Within the property table, right-click a column heading and clear the **Show Group Column** check box.

Simulation Data Inspector Enhancements

Multiple Plots in a View

The Simulation Data Inspector tool now supports the configuration of multiple plots into one *view*. On the **Inspect Signals** pane, on the View toolbar, select **Show Details** to display the View Details table.
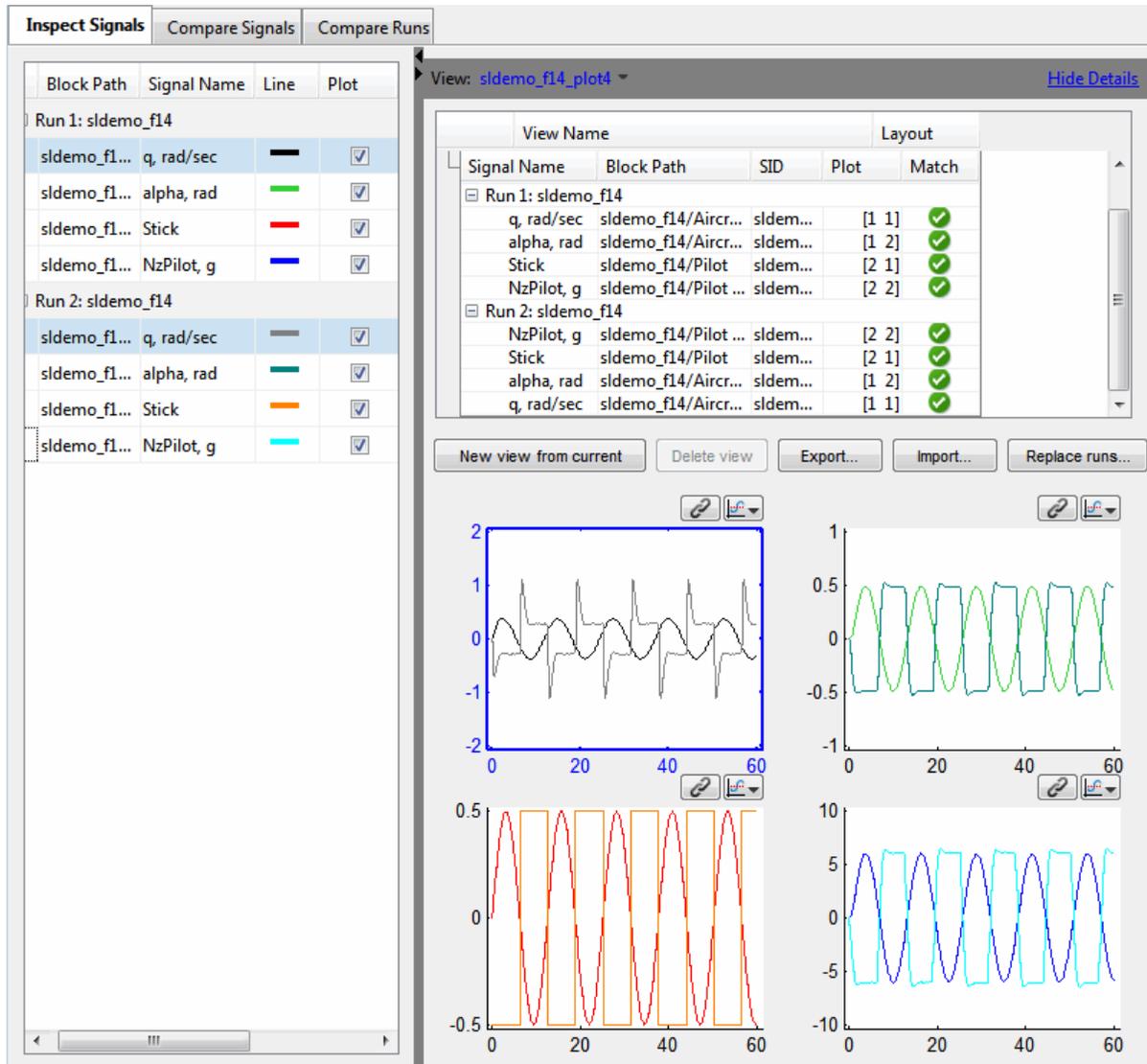
You can create multiple views by clicking the **New view from current** button. In each view, you can:

- Modify the number of plots by clicking the **Layout** column to display the plot matrix.

- Name, save, and reload the view using the corresponding buttons.
- Replace signal data for a run with the corresponding signal data of another run by clicking the **Replace runs** button.

For more information, see Visual Inspection of Signal Data in the Simulation Data Inspector Tool.

Display Run Properties

In R2011a, you can view the properties of a run. In the Signal Browser table, right-click a run name to view a list of options. To open the Run Properties dialog box, from the options list, select **Properties**.
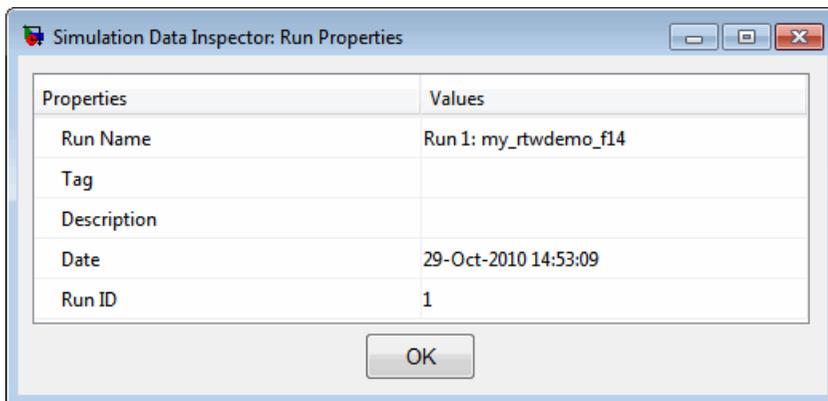
New Toolbar Icons

The Simulation Data Inspector toolbar includes a new icon 🔍 for zooming out a section of a plot. The previous zoom out icon ⛶ now performs a fit to view operation, which enlarges a plot to fill the graph. To perform either operation, select the icon, and click on a plot.

Model Advisor

In R2011a, the Model Advisor tool now includes easier control of the **By Product** and **By Task** folders. In the Model Advisor, select **View > Show By Product Folder** or **Show By Task Folder** to show or hide each folder. These settings are persistent across MATLAB sessions.

In the **By Task** folder, there are two new subfolders:

- **Modeling and Simulation**
- **Code Generation Efficiency**

For more information on the Model Advisor GUI, see Consulting the Model Advisor.

Configuration Parameters Dialog Box Changes

The Configuration Parameters dialog box layout is improved to better support your workflows. The **Optimization** pane is reorganized into three panes:

- **General**
- **Signals and Parameters**
- **Stateflow**

These panes make it easier to find parameters.

In R2011a, all tree nodes are collapsed by default. For details, see Configuration Parameters Dialog Box.

S-Functions

S-Functions Generated with legacy_code function and singleCPPMexFile S-Function Option Must Be Regenerated

Due to an infrastructure change, if you have generated an S-function with a call to legacy_code that defines the S-function option `singleCPPMexFile`, you must regenerate the S-function to use it with this release of Simulink.

For more information, see the description of legacy_code and Integrating Existing C Functions into Simulink Models with the Legacy Code Tool.

Compatibility Considerations

If you have generated an S-function with a call to legacy_code that defines the S-function option `singleCPPMexFile`, regenerate the S-function to use it with this release of Simulink.

R2010bSP2

Version: 7.6.2

Bug Fixes

R2010bSP1

Version: 7.6.1

Bug Fixes

R2010b

Version: 7.6

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Elimination of Regenerating Code for Rebuilds

For models that contain Model Reference blocks and that have not changed between Rapid Acceleration simulations, the rebuild process is more efficient.

Previously, if an Accelerator simulation or a Code Generation ERT/GRT was performed between two Rapid Acceleration simulations, then Simulink partially built the code a second time during the second Rapid Acceleration simulation.

Now, providing the model checksum remains constant, Simulink does not generate code for the second Rapid Accelerator simulation.

Component-Based Modeling

Model Workspace Is Read-Only During Compilation

During the compilation of a model, Simulink enforces that the model workspace is read-only, by issuing an error if there is an attempt to change a model workspace variable during compilation. This enforcement of a read-only workspace prevents the simulation from failing or producing incorrect results due to changes to the model workspace.

Compatibility Considerations

In previous releases, you could change model workspace variables when compiling a model (for example, this could occur when compiling referenced models). Rewrite any code that changes model workspace variables during compilation of a model.

Support for Multiple Normal Mode Instances of a Referenced Model

You can use Normal mode for multiple instances of a referenced model. Prior to R2010b, a model with model references could use Normal mode for at most one instance of each referenced model.

A referenced model must be in Normal mode for you to be able to use several important Simulink and Stateflow features, including linearization and model coverage analysis. Using Normal mode also can make editing and testing models more efficient.

In the `sldemo_mdlref_depgraph` demo, see the "Interacting with the Dependency Viewer in Instance View" section for an example of how to use multiple Normal mode instances of a referenced model. For additional information about using multiple Normal mode instances of a referenced model, see Using Normal Mode for Multiple Instances of Referenced Models.

Compatibility Considerations

The **Save As** feature preserves the **Simulation mode** setting of the Model block as much as possible.

If the both of the following conditions are true, then the saved model does not simulate:

- The R2010b model has multiple Normal mode instances of a referenced model.

- You use the **Save As** feature to save the model to a release earlier than R2010b that supports model reference Normal mode.

In this situation, the saved model does not simulate because only one instance of a referenced model could be in Normal mode in that earlier release.

Also, in releases before R2010b, you could select the **Refresh Model Blocks** menu item directly from the **Edit** menu in the Model Editor. In R2010b, access the **Refresh Model Blocks** menu item from the **Edit** > **Model Blocks** menu item.

New Variant Subsystem Block for Managing Subsystem Design Alternatives

A Variant Subsystem block provides multiple implementations for a subsystem where only one implementation is active during simulation. You can programmatically swap implementations without modifying the model. When the model is compiled, the Simulink engine chooses the active subsystem from a selection of subsystems. The active subsystem is determined by the values of the variant control variables and variant objects, which you define in the base workspace. By modifying the values of the variant control variables, you can easily specify which subsystem runs in your simulation.

For more information, see Setting Up Variant Subsystems. If you use the Model Advisor to check a system containing a variant subsystem, see Model Advisor Limitations, for more information.

Support for Bus and Enumerated Data Types on Masks

For the Masked Parameters dialog box, you can now create data type parameters that support the specification of bus or enumerated (`enum`) data types.

To create a data type parameter that supports bus data types, in the Mask Editor, select the **Parameters** pane.

For information about how to specify bus and enumerated data type parameters, see Data Type Control.

sl_convert_to_model_reference Function Removed

The `sl_convert_to_model_reference` function is obsolete and has been removed from the Simulink software.

To convert an atomic subsystem to a model reference, right-click the atomic subsystem and select the **Convert to Model Block** menu item, or use the Simulink.SubSystem.convertToModelReference function. See Atomic Subsystem and Converting a Subsystem to a Referenced Model for more information.

Verbose Accelerator Builds Parameter Applies to Model Reference SIM Target Builds in All Cases

For referenced models, the **Configuration Parameter** > **Optimization** > **Verbose accelerator build** parameter is no longer overridden by the **Configuration Parameter** > **Real-Time Workshop** > **Debug** > **Verbose build** parameter setting when building model reference SIM targets.

Embedded MATLAB Function Blocks

Specialization of Embedded MATLAB Function Blocks in Simulink Libraries

You can now create library instances of the same Embedded MATLAB Function block with distinct properties, including:

- Data type, size, complexity, sampling mode, range, and initial value
- Block sample time
- Fixed-point data type override mode
- Resolution to different MATLAB files on the path

With this capability, you can create custom block libraries using Embedded MATLAB Function blocks. For more information, see Creating Custom Block Libraries with MATLAB Function Blocks.

Support for Creation and Processing of Arrays of Buses

The Embedded MATLAB Function block now supports arrays of buses.

Ability to Include MATLAB Code as Comments in Generated C Code

You can now select to include MATLAB source code as comments in code generated for an Embedded MATLAB Function block. This capability improves traceability between generated code and the original source code.

Note: This option requires a Real-Time Workshop® license.

For more information, see MATLAB source code as comments in the Real-Time Workshop documentation.

Data Properties Dialog Box Enhancements

In R2010b, the following changes to the Data properties dialog box apply:

Parameters	Location in R2010a	Location in R2010b	Benefit of Location Change
Limit range · **Minimum** · **Maximum**	**Value Attributes** tab	**General** tab	Consistent with blocks in the Simulink library that specify these parameters on the same tab as the data type.
Save final value to base workspace	**Value Attributes** tab	**Description** tab	Consolidates parameters related to the data description.

Parameter Being Removed in Future Release

The **Save final value to base workspace** will be removed in a future release.

Simulink Data Management

Enhanced Support for Bus Objects as Data Types

The following blocks have added support for specifying a bus object as a data type:

- Constant
- Signal Specification

For the Constant block, if you use a bus object as a data type, you can set the **Constant value** to be one of these values:

- A full MATLAB structure corresponding to the bus object
- `0` to indicate a structure corresponding to the ground value of the bus object

See the Constant block reference page for an example that shows how to use a structure to simplify a model.

The following blocks and Simulink classes now use a consistent **Data type** parameter option, `Bus: <object name>`, for specifying a bus object as a data type:

- Constant block
- Bus Creator block
- Inport block
- Outport block
- Signal Specification block
- Simulink.BusElement class
- Simulink.Parameter class
- Simulink.Signal class

Compatibility Considerations

The interface for specifying a bus object as a data type is now consistent for the blocks that support that capability. Making the interface consistent involves removing some block parameters that existed in releases prior to R2010b. The following table summarizes the changes.

Block	Removed Pre-R2010b Parameters	Replacement R2010b Parameter
Bus Creator	**Bus object** **Specify properties via bus object**	**Output data type**
Inport	**Specify properties via bus object** **Bus object for validating input bus**	**Data type**
Outport	**Specify properties via bus object** **Bus object for validating input bus**	**Data type**

Enhancements to Simulink.NumericType Class

The Simulink.NumericType class now has the following methods:

- isboolean
- isdouble
- isfixed
- isfloat
- isscalingbinarypoint
- isscalingslopebias
- isscalingunspecified
- issingle

Importing Signal Data Sets into the Signal Builder Block

The Signal Builder block can now accept existing signal data sets. In previous releases, you had to enter existing signal data one by one in the Signal Builder dialog box or with the signalbuilder function.

In the Signal Builder block dialog box, you can now use the **File** > **Import from File** to import files that contain data sets. These data sets can contain test data that you have collected, or you can manually create these files. The block accepts the appropriately formatted file types:

- Excel (`.xls`, `.xlsx`)
- Comma-separated value (CSV) text files (`.csv`)
- MAT-files (`.mat`)

For further information, see Working with Signal Groups in the Simulink User's Guide.

signalbuilder Function Changes

The signalbuilder function has improved functionality:

To...	Use...
Add new groups to the Signal Builder block.	`'append'`
Append signals to existing signal groups in the Signal Builder block.	`'appendsignal'`
Make visible signals that are hidden in the Signal Builder block.	`'showsignal'`
Make invisible signals that are visible in the Signal Builder block.	`'hidesignal'`

From File Block Enhancements

The From File block includes the following new features:

- You can specify the method that the From File block uses to handle situations where there is not an exact match between a Simulink sample time hit and a time in the data file that the From File block reads.

  - In previous releases, the From File block automatically applied a linear interpolation and extrapolation method.

  - In R2010b, you can set the interpolation method independently for each of these situations:

- Data extrapolation before the first data point
- Data interpolation within the data time range
- Data extrapolation after the last data point
- The choices for the interpolation methods are (as applicable):
  - Linear interpolation
  - Zero-order hold
  - Ground value
- The From File block now can read signal data that has an enumerated (`enum`) data type, in addition to previously supported data types.

Finding Variables Used by a Model or Block

You can get a list of variables that a model or block uses.

In the Simulink Editor, right-click a block, subsystem, or the canvas and select the **Find Referenced Variables** menu item.

Simulink returns the results in the Model Explorer.

As an alternative, you can use the Model Explorer interface directly to find variables used by a model or block, as described in Finding Variables That Are Used by a Model or Block.

enumeration Function Replaced With MATLAB Equivalent

Starting with R2010b, when you invoke the enumeration function, you will be invoking a MATLAB equivalent of the Simulink function with the same name available in earlier releases.

See the description of the new MATLAB enumeration function introduced with new support for enumeration classes.

Programmatic Creation of Enumerations

The new Simulink.defineIntEnumType function provides a way to programmatically import enumerations defined externally—for example, in a data dictionary—to MATLAB. The function creates and saves a enumeration class definition file on the MATLAB path.

For more information, see the description of Simulink.defineIntEnumType and Enumerations and Modeling.

Simulink.Signal and Simulink.Parameter Objects Now Obey Model Data Type Override Settings

`Simulink.Signal` and `Simulink.Parameter` objects now honor model-level data type override settings. This capability allows you to share fixed-point models that use `Simulink.Signal` or `Simulink.Parameter` objects with users who do not have a Simulink Fixed Point™ license.

To simulate a model without using Simulink Fixed Point, use the Fixed-Point Tool to set the model-level **Data type override** setting to `Double` or `Single` and the **Data type override applies to** parameter to `All numeric types`. If you use `fi` objects or embedded numeric data types in your model, set the `fipref` `DataTypeOverride` property to `TrueDoubles` or `TrueSingles` and the `DataTypeOverrideAppliesTo` property to `All numeric types` to match the model-level settings. For more information, see fxptdlg in the Simulink documentation.

Simulink File Management

Autosave Upgrade Backup

New Autosave option to backup Simulink models when upgrading to a newer release. Automatically saving a backup copy can be useful for recovering the original file in case of accidental overwriting with a newer release.

You can set this Autosave option in the Simulink Preferences Window. See Autosave in the Simulink Graphical User Interface documentation.

Model Dependencies Tools

Enhanced file dependency analysis has the following new features:

- Find workspace variables that are required by your design but not defined by a file in the manifest
- Store code analysis warnings in the manifest
- Validate manifests before exporting to a ZIP file, to check for missing files and data
- Compare manifests with ZIP files and folders

For details see Model Dependencies.

Simulink Signal Management

Arrays of Buses

You can now use arrays of buses to represent structured data compactly, eliminating the need to include multiple copies of the same buses. You can iteratively process each element of the bus array, for example, by using a For Each subsystem.

The following blocks now support arrays of buses:

- Virtual blocks (see Virtual Blocks)
- These bus-related blocks:

  - Bus Assignment
  - Bus Creator
  - Bus Selector

- These nonvirtual blocks:

  - Merge
  - Multiport Switch
  - Rate Transition
  - Switch
  - Zero-Order Hold

- Assignment
- MATLAB Function (formally called Embedded MATLAB Function)
- Matrix Concatenate
- Selector
- Vector Concatenate
- Width
- Two-Way Connection (a Simscape block)

Create an array of buses with either a Vector Concatenate or Matrix Concatenate block. The input bus signals to these blocks must be nonvirtual and of the same type (that is, have the same names, hierarchies, and attributes for the leaf elements).

The generated code creates arrays of C structures that represent arrays of buses. You can use the Legacy Code Tool to integrate legacy C code that uses arrays of structures.

In an Embedded MATLAB® Function block, you can process arrays of bus signals using regular MATLAB syntax.

The use of arrays of buses does not support the following:

- Virtual buses
- Data loading or logging
- Stateflow action language

For details about using arrays of buses, see Combining Buses into an Array of Buses.

Compatibility Considerations

If you specify a bus object as the data type for a root Inport or Outport block, the **Dimensions** parameter is enabled, to allow you to specify dimensions other than 1 or -1 for an array of buses.

In previous releases, the **Dimensions** parameter was ignored if you specified a bus object as the data type for a root Inport or Outport block. If you specified a dimension other than 1 or -1, then do one of the following, depending on whether you want to use an array of buses or you want to output as a virtual bus:

- To use an array of buses:

  - In the **Signal Attributes** pane of the block parameters dialog box for a root Inport or Outport block, select the **Output as nonvirtual bus** option.
  - In the **Configuration Parameters** > **Diagnostics** > **Connectivity**>> pane, set **Mux blocks used to create bus signals** to `error`.
- To output as a virtual bus, set the **Dimensions** parameter to 1 or -1.

Loading Bus Data to Root Input Ports

You can now use MATLAB structures and `timeseries` objects when defining root-level input port signals. Using a structure of `timeseries` objects for bus signals simplifies loading bus data to root input ports.

To specify the input, use the **Configuration Parameters** > **Data Import/Export** > **Input** parameter. For more information, see Importing MATLAB timeseries Data and Importing Structures of MATLAB timeseries Objects for Bus Signals to a Root-Level Input Port.

Block Enhancements

Prelookup Block Supports Dynamic Breakpoint Data

The Prelookup block now supports specification of breakpoint data from an input port. Previously, you could specify breakpoint data only on the dialog box.

This enhancement enables you to change breakpoint data without stopping a simulation. For example, you can incorporate new breakpoint data if the physical system you are simulating changes.

Interpolation Using Prelookup Block Supports Dynamic Table Data

The Interpolation Using Prelookup block now supports specification of table data from an input port. Previously, you could specify table data only on the dialog box.

This enhancement enables you to change table data without stopping a simulation. For example, you can incorporate new table data if the physical system you are simulating changes.

Multiport Switch Block Supports Specification of Default Case for Out-of-Range Control Input

When the control input of the Multiport Switch block does not match any data port indices, you can specify the last data port as the default or use an additional data port. This enhancement enables you to avoid simulation errors and undefined behavior in the generated code.

Switch Block Icon Shows Criteria and Threshold Values

This enhancement helps you identify the **Criteria for passing first input** and **Threshold** values without having to open the Switch block dialog box.

Block Icon	Block Dialog Box
	Main / Signal Attributes Criteria for passing first input: `u2 ~= 0` Threshold: `0.5`
	Main / Signal Attributes Criteria for passing first input: `u2 >= Threshold` Threshold: `0.5`
	Main / Signal Attributes Criteria for passing first input: `u2 > Threshold` Threshold: `0.5`

Trigonometric Function Block Supports Expanded Input Range for CORDIC Algorithm

The Trigonometric Function block now supports an input range of [–2π, 2π) radians when you set **Function** to `sin`, `cos`, or `sincos` and set **Approximation method** to `CORDIC`. Previously, the input range allowed was [0, 2π) radians.

This enhancement enables you to use a wider range of input values that are natural for problems that involve trigonometric calculations.

Repeating Sequence Stair Block Supports Enumerated Data Types

The Repeating Sequence Stair block now supports enumerated data types. For more information, see Enumerations and Modeling in the Simulink User's Guide.

Abs Block Supports Specification of Minimum Output Value

The Abs block now supports specification of an **Output minimum** parameter. This enhancement enables you to specify both minimum and maximum values for block output. In previous releases, you could specify the maximum output value but not the minimum, which Simulink assumed to be 0 by default.

Saturation Block Supports Logging of Minimum and Maximum Values for the Fixed-Point Tool

When you set **Fixed-point instrumentation mode** to `Minimums, maximums and overflows` in the Fixed-Point Tool, the Saturation block logs minimum and maximum values. In previous releases, this block did not support min/max logging.

Vector Concatenate Block Now Appears in the Commonly Used and Signal Routing Libraries

In the Simulink Library Browser, the Vector Concatenate block now appears in the Commonly Used and Signal Routing libraries. This block continues to appear in the Math Operations library.

Model Discretizer Support for Second-Order Integrator Block

You can now discretize a model containing a Second-Order Integrator block using the Model Discretizer. Based on your block parameter settings, the tool replaces the continuous Second-Order Integrator block with one of the four discrete subsystems in the z-domain.

Integer Delay and Unit Delay Blocks Now Have Input Processing Parameter

The Integer Delay and Unit Delay blocks now have an **Input processing** parameter. This parameter enables you to specify whether the block performs sample- or frame-based processing on the input. To perform frame-based processing, you must have a Signal Processing Blockset™ license.

Compatibility Considerations

Beginning this release, MathWorks is changing how our products control frame-based processing. Previously, signals themselves were sample or frame based. Our blocks inherited that information from the signal, and processed the input accordingly, either as individual samples or as frames of data. Beginning this release, signals are no longer responsible for carrying information about their frame status. The blocks themselves now control whether they perform sample- or frame-based processing on the input.

Some blocks can do only one type of processing and thus require no changes. Other blocks can do both sample- and frame-based processing and thus require a new parameter. The Integer Delay and Unit Delay blocks fall into the latter category.

If you have any Integer Delay or Unit Delay blocks in an R2010a or earlier model, those blocks will continue to produce the same results in R2010b. When you open an existing model with an Integer Delay or Unit Delay block in R2010b, the **Input processing** parameter of those blocks will be set to `Inherited`. Your models will continue to run in this mode, but it is recommended that you run the slupdate function to set the **Input processing** parameter to the equivalent non-inherited mode. The non-inherited modes are `Elements as channels (sample based)` and `Columns as channels (frame based)`.

If you do not run the slupdate function on your model before the `Inherited` option is removed, any **Input processing** parameter set to `Inherited` on an Integer Delay or Unit Delay block will be set automatically to `Elements as channels (sample based)`.

Data Store Read Block Sample Time Default Changed to -1

In R2010b, the Data Store Read block uses a default value of `-1` for the **Sample time**, for consistency with the Data Store Write block and most other blocks. In previous releases, the default sample time was `0`.

Compatibility Considerations

The **Sample time** default for the Data Store Read block has changed from `0` in previous releases to `-1` in R2010b.

Support of Frame-Based Signals Being Removed From the Bias Block

Starting in R2010b, frame-based signal support is being removed from the Bias block. In a future release, the block will no longer support frame-based processing. To offset a frame-based signal in R2010b or later releases, you can use the Signal Processing Blockset Array-Vector Add block.

Compatibility Considerations

If you have any R2010a or earlier models that use the Bias block to offset a frame-based signal, you can use the slupdate function to upgrade your model. For each instance where you use a Bias block with a frame-based input signal, the `slupdate` function replaces the Bias block with an Array-Vector Add block.

Relaxation of Limitations for Function-Call Split Block

Two limitations of the Function-Call Split block have been relaxed for R2010b.

- Previously, the direct children of a branched function-call had to have periodic or asynchronous sample time. Now the direct children can also be triggered. Therefore, the branched function-call can trigger a Stateflow chart directly.

- Previously, if a branched function-call initiator was a Stateflow event, then the Stateflow function-call output event had to be bound to a particular state. Now the event can be bound or unbound to a state when invoking a branched function-call.

User Interface Enhancements

Model Explorer and Command-Line Support for Saving and Loading Configuration Sets

Previously, you could save and load a configuration set from the command line only, requiring many steps. Now you can save and load a configuration set using the Model Explorer. You can also save or load the active configuration set using one function, the Simulink.BlockDiagram.saveActiveConfigSet or Simulink.BlockDiagram.loadActiveConfigSet function.

For details, see the following sections in the *Simulink User's Guide*:

• Save a Configuration Set
• Load a Saved Configuration Set

Model Explorer: Grouping by a Property

In the Contents pane, you can group data based on a property values. For example, you can group by the `BlockType` property by right-clicking that column heading and selecting the **Group by This Column** menu item. The result looks similar to this:
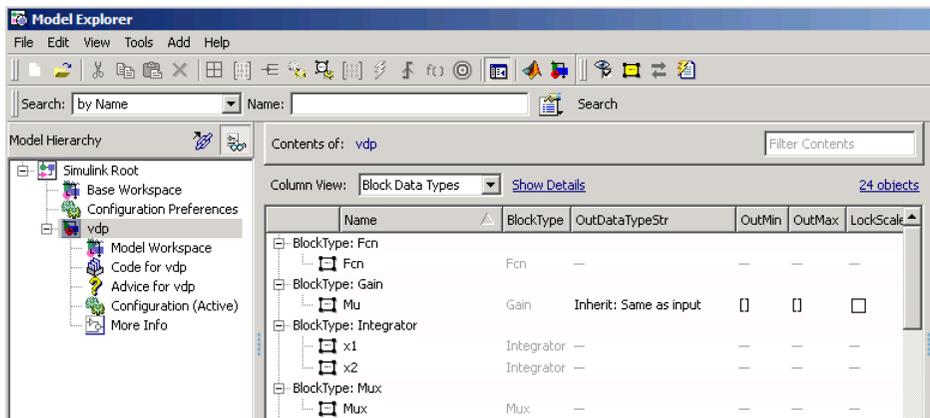
For details, see Grouping by a Property.

Model Explorer: Filtering Contents

In the **Contents** pane, you can specify a text string that the Model Explorer uses to filter the displayed objects. Use the **Filter Contents** text box at the top of the **Contents** pane to specify the text for filtering.
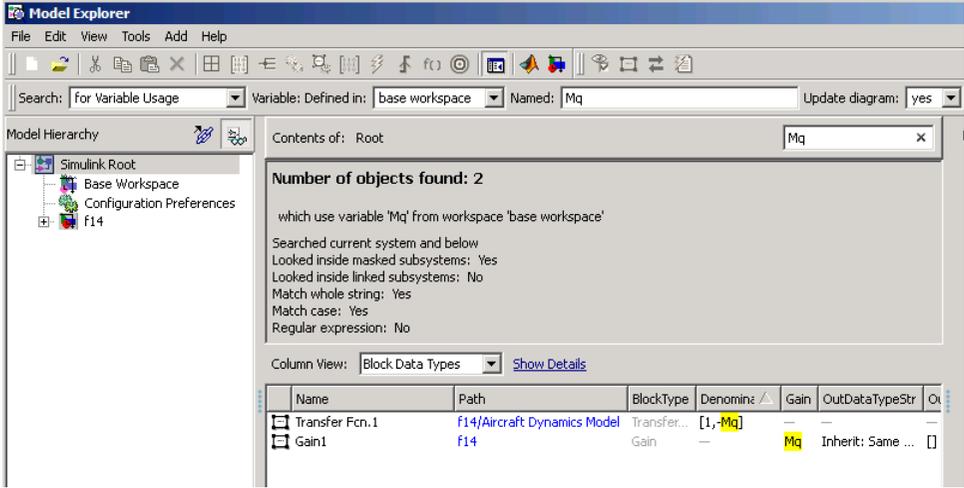
For details, see Filtering Contents.

Model Explorer: Finding Variables That Are Used by a Model or Block

In the Model Explorer, you can get a list of variables that a model or block uses. For example, one way to get that list of variables is:

1   In the **Contents** pane, right-click the block for which you want to find what variables it uses.

2   Select the **Find Referenced Variables** menu item.

You can also use the following approaches to find variables that are used by a model or block:

·   In the Model Explorer, in the **Model Hierarchy** pane, right-click a model or block and select the **Find Referenced Variables** menu item.

·   In the Model Explorer, in the search bar, use the `for Referenced Variables` search type option.

17-23

- In the Model Editor, right-click a block, subsystem, or the canvas and select the **Find Referenced Variables** option.

For details, see Finding Variables That Are Used by a Model or Block.

Model Explorer: Finding Blocks That Use a Variable

You can use the Model Explorer to get a list of blocks that use a workspace variable. One way to get that list of blocks is to right-click a variable in the **Contents** pane and select the **Find Where Used** menu item.

You can also find blocks that use a variable using one of these approaches:

- In the **Search** bar, select the `for Variable Usage` search type option.
- In the **Search Results** tab, right-click a variable and select the **Find Where Used** menu item.

For details, see Finding Blocks That Use a Specific Variable.

Model Explorer: Exporting and Importing Workspace Variables

You can export workspace variables from the Model Explorer to a MATLAB file or MAT-file.

One way to select the variables to export is by right-clicking the workspace node (for example, `Base Workspace`) and selecting the **Export** menu item.

Another way to select variables to export is to:

1. In the **Contents** pane, select the variables that you want to export.
2. Right-click on one of the highlighted variables and select the **Export Selected** menu item.

Also, you can import variables into a workspace in the Model Explorer:

1. In the **Model Hierarchy** pane, right-click the workspace into which you want to import variables.
2. Select the **Import** menu item.

For details, see Exporting Workspace Variables and Importing Workspace Variables.

Model Explorer: Link to System

The **Contents of** link at the top left side of the **Contents** pane links to the currently selected node in the **Model Hierarchy** pane.

Lookup Table Editor Can Now Propagate Changes in Table Data to Workspace Variables with Nonstandard Data Format

In R2010b, the Lookup Table Editor can propagate changes in table data to workspace variables with nonstandard data format when you:

- Use `sl_customization.m` to register a customization function for the Lookup Table Editor.
- Store this customization function on the MATLAB search path.

For more information, see Lookup Table Editor in the Simulink User's Guide.

Enhanced Designation of Hybrid Sample Time

Because of a new sample time enhancement, a block or signal with a continuous and a fixed in minor step sample time is no longer designated as hybrid. Instead, the block or signal is continuous and colored black. This enhancement assists in identifying hybrid subsystems that require attention.

Inspect Solver Jacobian Pattern

You can now inspect the solver Jacobian pattern in MATLAB and thereby determine if the pattern for your model is sparse. If so, the Sparse Perturbation Method and the Sparse Analytical Method may be able to take advantage of this sparsity pattern to reduce the number of computations necessary and thereby improve performance. For a demonstration that explains how to inspect and assess the sparsity pattern, see Exploring the Solver Jacobian Structure of a Model.

Inspection of Values of Elements in Checksum

You can now use Simulink.BlockDiagram.getChecksum to inspect the individual values of the elements making up the ConfigSet checksum.

Conversion of Error and Warning Messages Identifiers

In R2010b, all error and warning message identifiers that Simulink issues have a converted format. As part of this conversion, error and warning identifiers changed from a two-part format to a three-part format. For example, the message identifier 'Simulink:SL_SetParamWriteOnly' is now 'Simulink:Command:SetParamWriteOnly'.

Compatibility Considerations

Scripts that search for specific message identifiers or that turn off warning messages using an identifier must be updated with the new error and warning message identifiers. For an example script and a complete mapping of the new identifiers to the original identifiers, see http://www.mathworks.com/support/solutions/en/data/1-CNY5F6/index.html.

View and Compare Logged Signal Data from Multiple Simulations Using New Simulation Data Inspector Tool

This release introduces the new Simulation Data Inspector tool for quickly viewing and comparing logged signal data. You can use the tool to:

- View signal data in a graph
- View a comparison of specified signal data in a graph, including a plot of their differences
- Store signal data for multiple simulations so that you can specify and compare signal data between multiple simulations

For more information, see Inspecting and Comparing Logged Signal Data and Basic Simulation Workflow.

Viewing Requirements Linked to Model Objects

If your model, or blocks in your model, has links to requirements in external documents, you can now perform the following tasks without a Simulink Verification and Validation license:

- Highlight objects in a model that have links to requirements
- View information about a requirement

- Navigate from a model object to associated requirements
- Filter requirements highlighting based on keywords

S-Functions

Legacy Code Tool Support for Arrays of Simulink.Bus

The Legacy Code Tool now supports arrays of Simulink.Bus objects as valid data types in function specifications. For more information see Supported Data Types under Declaring Legacy Code Tool Function Specifications.

S-Functions Generated with legacy_code function and singleCPPMexFile S-Function Option Must Be Regenerated

Due to an infrastructure change, if you have generated an S-function with a call to legacy_code that defines the S-function option singleCPPMexFile, you must regenerate the S-function to use it with this release of Simulink.

For more information, see the description of legacy_code and Integrating Existing C Functions into Simulink Models with the Legacy Code Tool.

Compatibility Considerations

If you have generated an S-function with a call to legacy_code that defines the S-function option singleCPPMexFile, regenerate the S-function to use it with this release of Simulink.

Level-2 M-File S-Function Block Name Changed to Level-2 MATLAB S-Function

Level-2 MATLAB S-Function is the new name for the Simulink block previously called Level-2 M-File S-Function. In the Function Block Parameters dialog box, **S-function name** is the new name for the parameter previously called **M-file name**. The block type M-S-Function remains unchanged.

Compatibility Considerations

If you have a MATLAB script that uses the add_block function with the old block name, you need to update your script with the new name.

Functions Removed

Function Being Removed in a Future Release

This function will be removed in a future release of Simulink software.

Function Name	What Happens When You Use This Function?	Compatibility Considerations
simplot	Still works in R2010b	Use the Simulation Data Inspector to plot simulation data.

R2010a

Version: 7.5

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Computation of Sparse and Analytical Jacobian for Implicit Simulink Solvers

The implicit Simulink solvers now support numerical and analytical methods for computing the Jacobian matrix in one of the following representations: sparse perturbation, full perturbation, sparse analytical, and full analytical. The sparse methods attempt to improve performance by taking advantage of sparsity information associated with the Jacobian matrix. Similarly, the analytical methods attempt to improve performance by computing the Jacobian using analytical equations rather than the perturbation equations.

Since the applicability of these representations is highly model dependent, an `auto` option directs Simulink to use a heuristic to choose an appropriate representation for your model. In the case of a model that has a large number of states and for which the Jacobian is computed in sparse analytical form, the performance improvement may be substantial. In general, the performance improvement achieved varies from model to model.

Sparse Perturbation Support for RSim and Rapid Accelerator Mode

For implicit Simulink solvers, the numerical sparse perturbation method for solving the Jacobian supports both RSim and Rapid Accelerator mode.

Increased Accuracy in Detecting Zero-Crossing Events

The zero-crossing bracketing algorithm now uses a smaller tolerance for defining the interval in which an event occurs. The resulting increased accuracy of locating an event means that existing models may exhibit slightly different numerical results.

Saving Code Generated by Accelerating Models to slprj Folder

In Accelerator mode and in Rapid Accelerator mode, a build has historically resulted in the creation of generated code, respectfully, in the *modelname*_accel_rtw and the *modelname*_raccel_rtw folders in the current working folder. However, in order to be more consistent with other builds, in R2010a and future releases, these files will be created in the slprj/accel/*modelname* and the slprj/raccel/*modelname* folders.

Component-Based Modeling

Defining Mask Icon Variables

For model efficiency, use the **Icon & Ports** pane to run MATLAB code and to define variables used by the mask icon drawing commands. In releases earlier than R2010a, you had to use the **Initialization** pane to define variables used for icon drawing.

Simulink executes the MATLAB code in the **Icon & Ports** pane only when the block icon needs to be drawn. If you include variables used by mask icon drawing commands in the **Initialization** pane, Simulink evaluates the variables as part of simulation and code generation.

For more information, see Defining a Mask Icon.

Compatibility Considerations

Starting in R2010a, you can execute any MATLAB function in the **Ports & Icons** pane of the Mask Editor. If a variable in the mask workspace has the same name as a function in the **Ports & Icons** pane, Simulink returns an error.

For Each Subsystem Block

The For Each Subsystem block is very useful for modeling scenarios where you need to repeat the same algorithm on individual elements (or submatrices) of an input signal. The set of blocks within the subsystem represent the algorithm that is to be applied to a single element (or submatrix) of the original signal. You can configure the inputs of the subsystem to decompose the corresponding inputs into elements (or submatrices), and configure the outputs to suitably concatenate the processed results. Additionally, each block that has states inside this subsystem maintains separate sets of states for each element or submatrix it processes. Consequently, the operation of this subsystem is akin to copying the contents of the subsystem as many times as the number of elements in the original input signal, and then processing each element through its respective subsystem copy.

An additional benefit of this subsystem is that it may be utilized to improve code reuse in Real-Time Workshop generated code for certain models. Consider a model containing two reusable Atomic Subsystems with the same scalar algorithm applied to each element

of the signal. If the input signal dimensions for these subsystems are different, you will find that two distinct functions are produced in the code generated by Real-Time Workshop for this model. Now, if you were to convert the two subsystems to For Each Subsystems such that the contents of each processes a single scalar element, then you will find that the two subsystems produce a single function in the code generated by Real-Time Workshop. This function is parameterized by the number of elements to be processed.

New Function-Call Split Block

A new Function-Call Split block allows you to branch periodic and asynchronous function-call signals and connect them to multiple function-call subsystems (or models). These subsystems (or models) are guaranteed to execute in the order determined by their data dependencies. If a deterministic order cannot be computed, the model produces an error.

To test the validity of your function-call connections, use the Model Advisor diagnostic, **Check usage of function-call connections**. This diagnostic determines if:

- **Configurations > Diagnostics > Connectivity > Invalid function-call connection** is set to `error`
- **Configuration Parameters > Diagnostics > Connectivity > Context-dependent inputs** is set to `Enable All`

Trigger Port Enhancements

You can use trigger ports, which you define with a Trigger block, in new ways:

- Place edge-based (rising, falling, or either), as well as function-call, trigger ports at the root level of a model. Before R2010a, to place a trigger port in a root-level model, you had to set the trigger type to `function-call`.
- Place triggered ports in models referenced by a Model block. See Triggered Models.
- Lock down the data type, port dimension, and trigger signal sample time. To specify these values, use the new **Signal Attributes** pane of the Block Parameters dialog box of the Trigger block. Specifying these attributes is useful for unit testing and running standalone simulation of a subsystem or referenced model that has an edge-based trigger port. See Triggered Models.

Compatibility Considerations

When you add a trigger port to a root-level model, if you use the **File > Save As** option to specify a release before R2010a, Simulink replaces the trigger port with an empty subsystem.

Embedded MATLAB Function Blocks

New Ability to Use Global Data

Embedded MATLAB Function blocks are now able to use global data within a Simulink model and across multiple models.

This feature provides these benefits:

- Allows you to share data between Embedded MATLAB Function blocks and other Simulink blocks without introducing additional input and output wires in your model. This reduces unnecessary clutter and improves the readability of your model.
- Provides a means of scoping the visibility of data within your model.

For more information, see Using Global Data with the MATLAB Function Block in the Simulink documentation.

Support for Logical Indexing

Embedded MATLAB Function blocks now support logical indexing when variable sizing is enabled. Embedded MATLAB supports variable-size data by default for MEX and C/C++ code generation.

For more information about logical indexing, see Using Logicals in Array Indexing in the MATLAB documentation.

Support for Variable-Size Matrices in Buses

Embedded MATLAB Function blocks now support Simulink buses containing variable-size matrices as inputs and outputs.

Support for Tunable Structure Parameters

Embedded MATLAB Function blocks now support tunable structure parameters. See Working with Structure Parameters in MATLAB Function Blocks.

Check Box for 'Treat as atomic unit' Now Always Selected

In existing models, simulation and code generation for Embedded MATLAB Function blocks always behave as if the **Treat as atomic unit** check box in the Subsystem

Parameters dialog box is selected. Starting in R2010a, this check box is always selected for consistency with existing behavior.

Simulink Data Management

New Function Finds Variables Used by Models and Blocks

The new Simulink.findVars function returns information about workspace variables and their usage. For example, you can use `Simulink.findVars`, sometimes in conjunction with other Simulink functions, to:

- Identify all workspace variables used by a model or block
- Identify any workspace variables unused by a model or block
- Search a model for all places where a specified variable is referenced
- Subdivide a model, including only necessary variables with each model

See Simulink.findVars and the other Simulink functions referenced on that page for more information.

MATLAB Structures as Tunable Structure Parameters

You can create a MATLAB structure that groups base workspace variables into a hierarchy, and dereference the structure fields to provide values in Simulink block parameter expressions. This technique reduces base workspace clutter and allows related workspace variables to be conveniently grouped. However, in previous releases you could not use a MATLAB structure as a masked subsystem or a model reference argument, and no value given by a MATLAB structure field could be tuned. These restrictions limited the usefulness of MATLAB structures for grouping variables used in block parameter expressions.

In R2010a, these restrictions no longer apply to MATLAB structures that contain only numeric data. You can use a numeric structure, or any substructure within it, as a masked subsystem or a model reference argument, thereby passing all values in the structure with a single argument. You can also control MATLAB structure tunability using the same techniques that control MATLAB variable tunability. In R2010a, all values in a given structure must be either tunable or nontunable. See Using Structure Parameters for more information.

Simulink.saveVars Documentation Added

The Simulink.saveVars function was added in R2009b but was incompletely documented. See New Function Exports Workspace Variables and Values for more information.

Custom Floating-Point Types No Longer Supported

Custom floating-point types, `float(TotalBits, ExpBits)`, are no longer supported.

Compatibility Considerations

If you have code that uses custom floating-point types, modify this code using one of these methods:

- Replace calls to `float(TotalBits, ExpBits)` with calls to `fixdt('double')` or `fixdt('single')` as appropriate.

- Create your own custom float replacement function.

  Write a MATLAB function `custom_float_user_replacement` and place the file on your MATLAB path. This function must take `TotalBits` and `ExpBits` as input arguments and return a supported `numerictype` object, such as `fixdt('double')` or `fixdt('single')`.

  For example,

```
function DataType = custom_float_user_replacement(TotalBits,ExpBits)

if (TotalBits <= 32) && (ExpBits <= 8)
    DataType = numerictype('single');
else
    DataType = numerictype('double');
end
```

  In R2010a and future releases, if the file `custom_float_user_replacement.m` is on your MATLAB path, calls to `float(TotalBits, ExpBits)` automatically call `custom_float_user_replacement(TotalBits, ExpBits)`.

Data Store Logging

You can log the values of a local or global data store data variable for all the steps in a simulation. Data store logging is useful for:

- Model debugging – view the order of all data store writes
- Confirming a model modification – use the logged data to establish a baseline for comparing results to identify the impact of a model modification

To log a local data store that you create with a Data Store Memory block:

- Use the new **Logging** pane of the Block Parameters dialog box for the Data Store Memory block.
- Enable data store logging with the new **Configuration Parameters** > **Data Import/Export** > **Data stores** parameter.

To log a data store defined by a `Simulink.Signal` object, from the MATLAB command line, set `DataLogging` (which is a property of the `LoggingInfo` property of `Simulink.Signal`) to `1`.

For details, see Logging Data Stores. To see an example of logging a global data store, run the `sldemo_mdlref_dsm` demo.

Models with No States Now Return Empty Variables

Simulink creates empty variables for state logging (`xout`) or final state logging (`xfinal`), if both of these conditions apply:

- A model has no states.
- In the **Configuration Parameters** > **Data Import/Export** pane, you enable the **States**, **Final States**, or both parameters (the default is `off`).

Compatibility Considerations

If you configure your model to return empty variables when it has no states, then a possible result is that Simulink creates more variables than it did in previous releases.

Using model variants, running different models in batch mode, tuning models, or reconfiguring models can produce unexpected results based on the state values. For example, if you simulate a model that produces a state value, and then run a model variant that produces no state, Simulink overwrites the state value with an empty variable. If your model depends on the first state value not being overwritten if no state is returned in a subsequent simulation (which was the case in previous releases), then you get unexpected results.

To File Block Enhancements

The To File block now supports:

- Saving very large data sets that may be too large to fit in RAM
- Saving logged data up until the point of a premature ending of simulation processing. Previously, if the simulation processing did not complete, then To File did not store any logged data for that simulation.
- A new **Save format** parameter to control whether the block uses `Timeseries` or array format for data.

    - Use `Timeseries` format for writing multidimensional, real, or complex inputs, with different data types, (for example, built-in data types, including `Boolean`; enumerated (`enum`) data and fixed-point data with a word length of up to 32 bits.
    - Use Array format only for one-dimensional, double, noncomplex inputs. Time values are saved in the first row. Additional rows correspond to input elements.

Compatibility Considerations

For data saved using MAT file versions prior to 7.3, the From File block can only load two-dimensional arrays consisting of one-dimensional, double, noncomplex samples. To load data of any other type, complexity, or dimension, use a `Timeseries` object and save the file using MAT file version 7.3 or later. For example, use `'save file_name -v7.3 timeseries_object'`:

```
save file_name -v7.3 timeseries_object
```

From File Block Enhancements

The From File block now supports:

- Incremental loading of very large data sets that may be too large to fit in RAM
- Built-in data types, including `Boolean`
- Fixed-point data with a word length of up to 32 bits
- Complex data
- Multidimensional data

Root Inport Support for Fixed-Point Data Contained in a Structure

You can now use a root (top-level) Inport block to supply fixed-point data that is contained in a structure.

In releases before R2010a, you had to use a Simulink.Timeseries object instead of a structure.

Simulink Signal Management

Enhanced Support for Proper Use of Bus Signals

To improve model reliability and robustness, avoid mixing Mux blocks and bus signals. To help you use Mux blocks and bus signals properly, R2010a adds these enhancements:

- When Simulink detects Mux block and bus signal mixtures, the Mux blocks used to create bus signals diagnostic now generates:

  - A warning when all the following conditions apply:

    - You load a model created in a release before R2010a.
    - The diagnostic is set to `'None'`.
    - Simulink detects improper Mux block usage.

  - An error for new models

- Two new diagnostics in the **Configuration Parameters** > **Diagnostics** > **Connectivity** pane:

  - The Non-bus signals treated as bus signals diagnostic detects when Simulink implicitly converts a non-bus signal to a bus signal to support connecting the signal to a Bus Assignment or Bus Selector block.

  - The Repair bus selections diagnostic repairs broken selections in the Bus Selector and Bus Assignment block parameters dialog boxes that are due to upstream bus hierarchy changes.

Compatibility Considerations

In R2010a, if you load a model created in a prior release, you might get warning messages that you did not get before. To avoid getting Mux block-related warnings for existing models that you want to load in R2010a, use the `slreplace_mux` function to substitute Bus Creator blocks for any Mux blocks used to create buses signals.

Bus Initialization

In releases before R2010a:

- For *virtual* buses, you could specify a non-zero scalar or vector initial condition (IC) value that applies to all elements of the bus. You could use a vector value only if all bus elements use the same data type.

- For *nonvirtual* buses, the only value you could specify was zero.

In R2010a, you can create a MATLAB structure for an IC. You can:

- Specify ICs for all or a subset of the bus elements.
- Use the new `Simulink.Bus.createMATLABStruct` helper method to create a full IC structure.
- Use the new Model Advisor Simulink check, **Check for partial structure parameter usage with bus signals**, to detect when structure parameters are not consistent in shape with the associated bus signal.

Using IC structures helps you to:

- Specify nonzero initial conditions
- Specify initial conditions for mixed-dimension signals
- Apply a different IC for each signal in the bus
- Specify ICs for a subset of signals in a bus without specifying ICs for all the signals
- Use the same ICs for multiple blocks, signals, or models

For information about creating and using initial condition structures, see Specifying Initial Conditions for Bus Signals.

S-Functions for Working with Buses

The following S-functions provide a programmatic interface for working with buses:

S-function	Description
ssGetBusElementComplexSignal	Get the signal complexity for a bus element.
ssGetBusElementDataType	Get the data type identifier for a bus element.
ssGetBusElementDimensions	Get the dimensions of a bus element.
ssGetBusElementName	Get the name of a bus element.

S-function	Description
ssGetBusElementNumDimensions	Get the number of dimensions for a bus element.
ssGetBusElementOffset	Get the offset from the start of the bus data type to a bus element.
ssGetNumBusElements	Get the number of elements in a bus signal.
ssGetSFcnParamName	Get the value of a block parameter for an S-function block.
ssIsDataTypeABus	Determine whether a data type identifier represents a bus signal.
ssRegisterTypeFromParameter	Register a data type that a parameter in the Simulink data type table specifies.
ssSetBusInputAsStruct	Specify whether to convert the input bus signal for an S-function from virtual to nonvirtual.
ssSetBusOutputAsStruct	Specify whether the output bus signal from an S-function must be virtual or nonvirtual.
ssSetBusOutputObjectName	Specify the name of the bus object that defines the structure and type of the output bus signal.

Command Line API for Accessing Information About Bus Signals

You can use two new signal property parameters to get information about the type and hierarchy of a signal programmatically:

- `CompiledBusType`

  - Returns information about whether the signal connected to a port is a bus, and if so, whether it is a virtual or nonvirtual bus

- `SignalHierarchy`

  - Returns the signal name of the signal. If the signal is a bus, the parameter also returns the hierarchy and names of the bus signal.

See Model Parameters and View Information about Buses.

Signal Name Propagation for Bus Selector Block

The new `SignalNameFromLabel` port parameter supports signal name propagation for Bus Creator block input signals whenever you change the name of an input signal programmatically. You can set this parameter with the `set_param` command, specifying either a port or line handle and the signal name to propagate.

See Model Parameters.

Block Enhancements

New Square Root Block

You can use the new Sqrt block to perform square-root calculations. This block includes the following functions:

Function	Icon
sqrt	
signedSqrt	
rSqrt	

Compatibility Considerations

The sqrt and 1/sqrt functions no longer appear in the Math Function block. For backward compatibility, models with a Math Function block that uses one of these two functions continue to work. However, consider running the slupdate function on your model. slupdate replaces any Math Function block that uses sqrt or 1/sqrt with an equivalent Sqrt block that ensures the same behavior.

New Second-Order Integrator Block

You can use the new Second-Order Integrator block to model second-order systems that have bounds on their states. This block is useful for modeling physical systems, for example, systems that use Newton's Second Law and have constraints on their motion.

Benefits of using this block include:

- Highly accurate results
- Efficient detection of zero crossings

- Prevention of direct feedthrough and algebraic loops

New Find Nonzero Elements Block

You can use the new Find block to locate all nonzero elements of an input signal. This block outputs the indices of nonzero elements in linear indexing or subscript form and provides these benefits:

When you use the block to...	You can...
Convert logical indexing to linear indexing	Use the linear indices you get from processing a logical indexing signal as the input to a Selector or Assignment block
Extract subscripts of nonzero values	Use the subscript of matrices for 2-D or higher-dimensional signal arrays to aid with image processing
Represent sparse signals	Use indices and values as a compact representation of sparse signals

PauseFcn and ContinueFcn Callback Support for Blocks and Block Diagrams

The new PauseFcn and ContinueFcn callbacks detect clicking of the **Pause** and **Continue** buttons during simulation. You can set these callbacks using the set_param command or the **Callbacks** tab of the Model Properties dialog box. Both the PauseFcn and ContinueFcn callbacks support Normal and Accelerator simulation modes.

Gain Block Can Inherit Parameter Data Type from Gain Value

The Gain block now supports the **Parameter data type** setting of Inherit: Inherit from 'Gain'. This enhancement provides the benefit of inheriting the parameter data type directly from the **Gain** parameter. For example:

If you set Gain to...	The parameter data type inherits...
2	double
single(2)	single

If you set Gain to...	The parameter data type inherits...
`int8(2)`	`int8`

Direct Lookup Table (n-D) Block Enhancements

The Direct Lookup Table (n-D) block now supports:

- Multidimensional signals for the table input port
- Fixed-point data types for the table input port
- Explicit specification of the table data type in the block dialog box

Multiport Switch Block Allows Explicit Specification of Data Port Indices

The icon for the Multiport Switch block now shows the values of indices on data port labels. This enhancement helps you identify the data inputs without having to open the block dialog box:

Block Parameter Settings	Block Icon
Data port order: Specify indices Data port indices (e.g. {1,[2,3]}): `{[1,4],10,5}`	
Data port order: Specify indices Data port indices (e.g. {1,[2,3]}): `{myColors.RED, myColors.YELLOW, [myColors.GREEN, myColors.BLACK]}`	
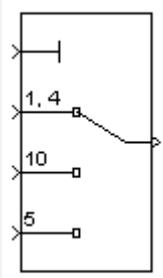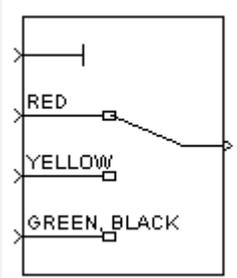
When you load existing models that contain the Multiport Switch block, the following parameter mapping occurs:

Block Parameter Settings of a Model from R2009b or Earlier	Block Parameter Settings When You Load the Model in R2010a
Number of inputs: 3 ☐ Use zero-based indexing	Data port order: One-based contiguous Number of data ports: 3
Number of inputs: 3 ☑ Use zero-based indexing	Data port order: Zero-based contiguous Number of data ports: 3

The following command-line parameter mapping applies:

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of inputs	Number of data ports	Inputs	Same
Use zero-based indexing	Data port order	zeroidx	DataPortOrder

The parameter mapping in R2010a ensures that you get the same block behavior as in previous releases.

Compatibility Considerations

In R2010a, a warning appears at compile time when your model contains a Multiport Switch block with the following configuration:

- The control port uses an enumerated data type.
- The data port order is contiguous.

During edit time, the block icon cannot show the mapping of each data port to an enumerated value. This configuration can also lead to unused ports during simulation and unused code during Real-Time Workshop code generation.

Run the slupdate function on your model to replace each Multiport Switch block of this configuration with a block that explicitly specifies data port indices. Otherwise, your model might not work in a future release.

In R2010a, the following Multiport Switch block configuration also produces a warning at compile time:

- The control port uses a fixed-point or built-in data type.
- The data port order is contiguous.
- At least one of the contiguous data port indices is not representable with the data type of the control port.

The warning alerts you to unused ports during simulation and unused code during Real-Time Workshop code generation.

Trigonometric Function Block Supports CORDIC Algorithm and Fixed-Point Data Types

When you select `sin`, `cos`, or `sincos` for the Trigonometric Function block, additional parameters are available.

New Block Parameter	Purpose	Benefit
Approximation method	Specify the type of approximation the block uses to compute output: `None` or `CORDIC`.	Enables you to use a faster method of computing block output for fixed-point and HDL applications.
Number of iterations	For the `CORDIC` algorithm, specify how many iterations to use for computing block output.	Enables you to adjust the precision of your block output.

This block now supports fixed-point data types when you select `sin`, `cos`, or `sincos` and set **Approximation method** to `CORDIC`.

Enhanced Block Support for Enumerated Data Types

The following Simulink blocks now support enumerated data types:

- Data Type Conversion Inherited
- Data Type Duplicate
- Interval Test
- Interval Test Dynamic
- Probe (input only)
- Relay (output only)
- Unit Delay Enabled
- Unit Delay Enabled Resettable
- Unit Delay Resettable
- Unit Delay With Preview Enabled
- Unit Delay With Preview Enabled Resettable
- Unit Delay With Preview Enabled Resettable External RV
- Unit Delay With Preview Resettable
- Unit Delay With Preview Resettable External RV

For more information, see Enumerations and Modeling in the Simulink User's Guide.

Lookup Table Dynamic Block Supports Direct Selection of Built-In Data Types for Outputs

In R2010a, you can select the following data types directly for the **Output data type** parameter of the Lookup Table Dynamic block:

- `double`
- `single`
- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean`

Previously, you had to enter an expression for **Output data type** to specify a built-in data type.

Compare To Zero and Wrap To Zero Blocks Now Support Parameter Overflow Diagnostic

If the input data type to a Compare To Zero or Wrap To Zero block cannot represent zero, detection of this parameter overflow occurs. In the **Diagnostics > Data Validity** pane of the Configuration Parameters dialog box, set **Parameters > Detect overflow** to `warning` or `error`.

Data Type Duplicate Block Enhancement

The Data Type Duplicate block is now a built-in block. Previously, this block was a masked S-Function. The read-only **BlockType** parameter has changed from `S-Function` to `DataTypeDuplicate`.

Compatibility Considerations

In R2010a, signal propagation might behave differently from previous releases. As a result, your model might not compile under these conditions:

- Your model contains a Data Type Duplicate block in a source loop.
- Your model has underspecified signal data types.

If your model does not compile, set data types for signals that are not fully specified.

Lookup Table and Lookup Table (2-D) Blocks To Be Deprecated in a Future Release

In a future release, the Lookup Table and Lookup Table (2-D) blocks will no longer appear in the Simulink Library Browser. Consider replacing instances of those two blocks by using 1-D and 2-D versions of the Lookup Table (n-D) block. Among other enhancements, the Lookup Table (n-D) block supports the following features that the other two blocks do not:

- Specification of parameter data types different from input or output signal types

- Reduced memory use and faster code execution for evenly spaced breakpoints that are nontunable
- Fixed-point data types with word lengths up to 128 bits
- Specification of index search method
- Specification of action for out-of-range inputs

To upgrade your model:

Step	Description	Reason
1	Run the Simulink Model Advisor check for Check model, local libraries, and referenced models for known upgrade issues.	Identify blocks that do not have compatible settings with the Lookup Table (n-D) block.
2	For each block that does not have compatible settings with the Lookup Table (n-D) block: • Decide how to address each warning. • Adjust block parameters as needed.	Modify each Lookup Table or Lookup Table (2-D) block to make them compatible.
3	Repeat steps 1 and 2 until you are satisfied with the results of the Model Advisor check.	Ensure that block replacement works for the entire model.
4	Run the slupdate function on your model.	Perform block replacement with the Lookup Table (n-D) block.

Compatibility Considerations

The Model Advisor check groups all Lookup Table and Lookup Table (2-D) blocks into three categories:

- Blocks that have compatible settings with the Lookup Table (n-D) block
- Blocks that have incompatible settings with the Lookup Table (n-D) block
- Blocks that have repeated breakpoints

Blocks with Compatible Settings

When a block has compatible parameter settings with the Lookup Table (n-D) block, automatic block replacement can occur without backward incompatibilities.

Lookup Method in the Lookup Table or Lookup Table (2-D) Block	Parameter Settings in the Lookup Table (n-D) Block After Block Replacement	
	Interpolation	Extrapolation
`Interpolation-Extrapolation`	`Linear`	`Linear`
`Interpolation-Use End Values`	`Linear`	`None-Clip`
`Use Input Below`	`None-Flat`	Not applicable

Depending on breakpoint characteristics, the Lookup Table (n-D) block uses one of two index search methods.

Breakpoint Characteristics in the Lookup Table or Lookup Table (2-D) Block	Index Search Method in the Lookup Table (n-D) Block After Block Replacement
Not evenly spaced	`Binary search`
Evenly spaced and tunable	A prompt appears, asking you to select `Binary search` or `Evenly spaced points`.
Evenly spaced and nontunable	

The Lookup Table (n-D) block also adopts other parameter settings from the Lookup Table or Lookup Table (2-D) block. For parameters that exist only in the Lookup Table (n-D) block, the following default settings apply after block replacement:

Lookup Table (n-D) Block Parameter	Default Setting After Block Replacement
Breakpoint data type	`Inherit: Same as corresponding input`
Action for out-of-range input	`None`

Blocks with Incompatible Settings

When a block has incompatible parameter settings with the Lookup Table (n-D) block, the Model Advisor shows a warning and a recommended action, if applicable.

- If you perform the recommended action, you can avoid incompatibility during block replacement.

- If you use automatic block replacement without performing the recommended action, you might see numerical differences in your results.

Incompatibility Warning	Recommended Action	What Happens for Automatic Block Replacement
The **Lookup Method** is `Use Input Nearest` or `Use Input Above`. The Lookup Table (n-D) block does not support these lookup methods.	Change the lookup method to one of the following: • `Interpolation - Extrapolation` • `Interpolation - Use End Values` • `Use Input Below`	The **Lookup Method** changes to `Interpolation - Use End Values`. In the Lookup Table (n-D) block, this setting corresponds to: • **Interpolation** set to `Linear` • **Extrapolation** set to `None-Clip`
The **Lookup Method** is `Interpolation - Extrapolation`, but the input and output are not the same floating-point type. The Lookup Table (n-D) block supports linear extrapolation only when all inputs and outputs are the same floating-point type.	Change the extrapolation method or the port data types of the block.	You also see a message that explains possible numerical differences.
The block uses small fixed-point word lengths, so that interpolation uses only one rounding operation. The Lookup Table (n-D) block uses two rounding operations for interpolation.	None	You see a message that explains possible numerical differences.

Blocks with Repeated Breakpoints

When a block has repeated breakpoints, the Model Advisor recommends that you change the breakpoint data and rerun the check. You cannot perform automatic block replacement for blocks with repeated breakpoints.

Elementary Math Block Now Obsolete

The Elementary Math block is now obsolete. You can replace any instance of this obsolete block in your model by using one of these blocks in the Math Operations library:

• Math Function

- Rounding Function
- Trigonometric Function

Compatibility Considerations

If you open a model that contains an Elementary Math block, a warning message appears. This message suggests running slupdate on your model to replace each instance of the obsolete block with an appropriate substitute.

If you try to start simulation or generate code for a model that contains this obsolete block, an error message appears.

DocBlock Block RTF File Compression

In R2010a, when you add or modify a DocBlock block that uses Microsoft RTF format and you save the model, Simulink compresses the RTF file. The saved RTF files with images are much smaller than in previous releases.

Compatibility Considerations

In R2010a, if you use slupdate or save a model that includes a DocBlock block that uses RTF format, you cannot run the model in an earlier version of Simulink.

To run a model that has a compressed RTF file in an earlier version of Simulink, use **Save As** to save the model in the format of the earlier release.

Simulink Extras PID Controller Blocks Deprecated

In R2010a, the PID Controller (with Approximate Derivative) and PID Controller blocks of the Simulink Extras library no longer appear in the Simulink Library Browser. For models created using R2009b or earlier, consider using the slupdate function to replace these blocks with the new PID Controller block of the Simulink/Continuous or Simulink/Discrete library. Among other enhancements, the new PID Controller block supports:

- Continuous-time and discrete-time modeling
- Ideal and Parallel controller forms
- Automatic PID tuning (requires a Simulink Control Design™ license)

For more information, see the PID Controller and PID Controller (2 DOF) block reference pages.

Compatibility Considerations

For backward compatibility, simulation and code generation of models that contain the deprecated PID Controller (with Approximate Derivative) or PID Controller block continue to work.

User Interface Enhancements

Model Explorer Column Views

The Model Explorer now supports column views, which specify sets of property columns to display in the **Contents** pane. The Model Explorer displays only the properties that are defined for the current column view. The Model Explorer does not add new properties dynamically as you add objects to the **Contents** pane. Using a defined subset of properties to display streamlines the task of exploring and editing model object properties and increases the density of the data displayed.

Model Explorer provides several standard column views with common property sets. You can:

- Select the column view based on the task you are performing
- Customize the standard column views
- Create your own column views
- Export and import column views saved in MAT-files, which you can share with other users

See The Model Explorer: Controlling Contents Using Views.

Compatibility Considerations

Column views replace the **Customize Contents** option provided in previous releases.

In R2010a, the Model Explorer provides a different interface for performing some of the tasks that you previously performed using **View** menu items. The following table summarizes differences between R2009b and R2010a.

R2009b View Menu Item	R2010a Model Explorer Interface Change
Dialog View	Replaced by **Show Dialog Pane**
Customize Contents	Replaced by **Column View > Show Details**
Show Properties	Eliminated; select **Column View > Show Details** to specify properties to display

R2009b View Menu Item	R2010a Model Explorer Interface Change
Mark Nonexistent Properties	Replaced by **Show Nonexistent Properties as ' - '**
Library Browser	Eliminated (you can access the Library Browser from the Simulink Editor **View** menu)
List View Options	Replaced by **Row Filter**

Model Explorer Display of Masked Subsystems and Linked Library Subsystems

The Model Explorer now contains global options for specifying whether the Model Explorer displays the contents of library links and masked subsystems. These options also control whether the **Model Hierarchy** pane displays linked or masked subsystems. See Displaying Masked Subsystems and Displaying Linked Library Subsystems.

Compatibility Considerations

In R2010a, when you select a masked subsystem node in the **Model Hierarchy** pane, the **Contents** pane displays the objects of the subsystem, reflecting the global setting to display masked subsystems. In prior releases, if you selected a masked subsystem node, you needed to right-click the node and select **Look Under Mask** to view the subsystem objects in the **Contents** pane.

In R2010a, the search results reflect the **Show Library Links** and **Show Masked Subsystems** settings. In previous releases, you specified the **Look Inside Masked Subsystems** and **Look Inside Linked Subsystems** options as part of the search options. R2010a does not include those search options.

Model Explorer Object Count

The top-right section of the **Contents** pane displays a count of objects found for the currently selected nodes in the **Model Hierarchy** pane. The count indicates the number of objects displayed in the **Contents** pane, compared to the total number of objects in the currently selected nodes. The number of displayed objects is less than the total number of objects in scope when you filter some objects by using **View** > **Row Filter** options. See Object Count.

Model Explorer Search Option for Variable Usage

You can use the new `for Variable Usage` search type to search for blocks that use a variable that is defined in the base or model workspaces. See Search Bar Controls.

Model Explorer Display of Signal Logging and Storage Class Properties

The Model Explorer **Contents** pane displays the following additional properties for signal lines:

- Signal logging-related properties (such as `DataLogging`)
- Storage class properties, including properties associated with custom storage classes for signals

Displaying these properties in the **Contents** pane enables batch editing. Prior to R2010a, you could edit these properties only in the Signal Properties dialog box.

Model Explorer Column Insertion Options

In R2010a, right-clicking on a column heading in the Contents pane provides two new column insertion options:

- **Insert Path** – adds the `Path` property column to the right of the selected column.
- **Insert Recently Hidden Columns** – selects a property from a list of columns you recently hid, to add that property column to the right of the selected column

See Adding Property Columns.

Diagnostics for Data Store Memory Blocks

The Model Advisor 'By Task' folder now contains a Data Store Memory Blocks subfolder. This subfolder contains checks relating to Data Store Memory blocks that examine your model for:

- Multitasking, strong typing, and shadowing issues
- An enabled status of the read/write diagnostics
- Read/write issues

New Command-Line Option for RSim Targets

A new −h command-line option allows you to print a summary of the available options for RSim executable targets.

Simulink.SimulationOutput.get Method for Obtaining Simulation Results

The Simulink.SimulationOutput class now has a get method. After simulating your model, you can use this method to access simulation results from the Simulink.SimulationOutput object.

Simulink.SimState.ModelSimState Class has New snapshotTime Property

The Simulink.SimState.ModelSimState class has a new snapshotTime property. You can use this property to access the exact time at which Simulink took a "snapshot" of the simulation state (SimState) of your model.

Simulink.ConfigSet.saveAs to Save Configuration Sets

The saveAs method is added to the Simulink.ConfigSet class to allow you to easily save the settings of configuration sets as MATLAB functions or scripts. Using the MATLAB function or script, you can share and archive model configuration sets. You can also compare the settings in different configuration sets by comparing the MATLAB functions or scripts of the configuration sets.

For details, see Save a Configuration Set in the *Simulink User's Guide*.

S-Functions

Building C MEX-Files from Ada and an Example Ada Wrapper

In an R2008b release note, MathWorks announced that support for Ada S-functions in Simulink would be removed in a future release and a migration strategy would be forthcoming.

In this release, the addition of Technical Note 1821 facilitates your incorporating Ada code into Simulink without using Ada S-function support. This note, "Developing and Building Ada S-Functions for Simulink", is available at Technical Note 1821 and demonstrates:

- How to build a C MEX S-function from Ada code without using the `mex -ada` command
- An example of an Ada wrapper around a C MEX S-Function API

New S-Function API Checks for Branched Function-Calls

A new S-function API, `ssGetCallSystemNumFcnCallDestinations`, allows you to determine the number of function-call blocks that your S-function calls. Based on this returned number, you can then deduce whether or not your S-function calls a branched function-call.

You can call this SimStruct function from `mdlSetWorkWidths` or later in your S-function.

New C MEX S-Function API and M-File S-Function Flag for Compliance with For Each Subsystem

To allow a C MEX S-function to reside inside of a For Each Subsystem block, you must call the new `ssSupportsMultipleExecInstances` API and set the flag to true in the `mdlSetWorkWidths` method.

As for M-file S-functions, you must set the new flag `block.SupportsMultipleExecInstances` to true in the Setup section.

Legacy Code Tool Enhanced to Support Enumerated Data Types and Structured Tunable Parameters

The Legacy Code Tool has been enhanced to support

- Enumerated data types for input, output, parameters, and work vectors
- Structured tunable parameters

For more information about data types that the Legacy Code Tool supports, see Supported Data Types. For more information about the Legacy Code Tool, see

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Writing S-Functions documentation
- legacy_code function reference page

Compatibility Considerations

For enumerated data type support:

- If you upgrade from R2008b or later release, you can continue to compile the S-function source code and continue to use the compiled output from an earlier release without recompiling the code.
- If you upgrade from R2008a or earlier release, you cannot use enumerated types; the Simulink engine will display an error during simulation.

You cannot use tunable structured parameters with Legacy Code Tool in a release prior to R2010a.

Documentation Improvements

Modeling Guidelines for High-Integrity Systems

MathWorks intends the Modeling Guidelines for High-Integrity Systems document to be for engineers developing models and generating code for high-integrity systems using Model-Based Design with MathWorks products. This document describes creating Simulink models that are complete, unambiguous, statistically deterministic, robust, and verifiable. The document focus is on model settings, block usage, and block parameters that impact simulation behavior or code generated by the Real-Time Workshop Embedded Coder product.

These guidelines do not assume that you use a particular safety or certification standard. The guidelines reference some safety standards where applicable, including DO-178B, IEC 61508, and MISRA C.

You can use the Model Advisor to support adhering to these guidelines. Each guideline lists the checks that are applicable to that guideline.

For more information, see Modeling Guidelines for High-Integrity Systems in the Simulink documentation.

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Included in Help

MathWorks Automotive Advisory Board (MAAB) involves major automotive original equipment manufacturers (OEMs) and suppliers in the process of evolving MathWorks controls, simulation, and code generation products, including the Simulink, Stateflow, and Real-Time Workshop products. An important result of the MAAB has been the "MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow." Help for the Simulink product now includes these guidelines. The MAAB guidelines link to relevant Model Advisor MAAB check help and MAAB check help links to relevant MAAB guidelines.

For more information, see MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow in the Simulink documentation.

R2009bSP1

Version: 7.4.1

Bug Fixes

R2009b

Version: 7.4

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Single-Output `sim` Syntax

An enhanced `sim` command provides for greater ease of use and for greater compatibility with `parfor` loops. Since the command now saves all simulation results to a single object, the management of output variables is straightforward for all cases, including parallel computing.

Expanded Support by Rapid Accelerator

Simulink Rapid Accelerator mode now supports root inputs of enumerated data type and fixed-point parameters of any word length.

SimState Support in Accelerator Mode

Simulink Accelerator mode now supports the SimState feature. You can therefore save the simulation state and later resume the simulation from the exact save time.

Integer Arithmetic Applied to Sample Hit Computations

For fixed-step simulations, Simulink now computes sample time hits using integer arithmetic. This modification improves the timing resolution of sample hits of multirate models.

Compatibility Considerations

Previously, if an S-function had two rates, and if (ssIsSampleHit(S, idx1) == true && ssIsSampleHit(S,idx2) == true, then Simulink would adjust the task times to be evaluated as ssGetTaskTime(S, idx1) == ssGetTaskTime(S, idx2). Simulink no longer forces this equality; instead, Simulink now leaves the individual task times to be integer multiples of their corresponding periods. Consequently, existing code with logic that relies upon the equality of the task times needs to be updated.

In addition, the behavior of the command get_param(model, 'SimulationTime') is now different. Instead of returning the time of the next known sample hit at the bottom of the current step, this command now returns the current time.

Improved Accuracy of Variable-Step Discrete Solver

For variable-step discrete simulation of purely discrete models, where the fundamental step size is the same as the fastest discrete rate, Simulink now uses the specified start and stop times.

Compatibility Considerations

Previously, if the fundamental step size was equal to the fastest discrete rate, the Simulink simulation did not uniformly honor the user-specified start and stop times. Specifically, if the start and stop times were not exact multiples of the fundamental step size, then the start time was adjusted to the time of the first sample time hit and the simulation stopped at the sample time hit just before the specified stop time. However, if the simulation was required to hit certain time points (either by specifying `TSPAN` in the `sim` command such as 'sim('Model_A',[0 10])', or via the `OutputTimes` parameter), then the start and stop times were not adjusted

Now Simulink variable-step simulation of purely discrete models consistently honors the user-specified start and stop times, irrespective of whether the fastest discrete sample time is the GCD of all of the other sample times

Component-Based Modeling

Enhanced Library Link Management

In R2009b, improved library link management (Links Tool) facilitates visualizing and restoring edited library links. See Working with Library Links for more information.

Enhanced Mask Editor Provides Tabs and Signal Attributes

You can use the R2009b Mask Editor to create a mask that has tabbed panes, and define the same signal attribute specifications in a mask that built-in Simulink blocks provide. See Working with Block Masks , Simulink Mask Editor and Mask Icon Drawing Commands for more information.

Model Reference Variants

Model reference variants allow you to configure any Model block to select its referenced model from a set of candidate models. The selection occurs when you compile the model that contains the Model block, and depends on the values of one or more MATLAB variables or Simulink parameters in the base workspace. To configure a Model block to select the model that it references, you:

- Provide a set of Boolean expressions that reference base workspace values.
- Associate each expression with one of the models that the block could reference.

When you compile the model, Simulink evaluates all the expressions. Each Model block that uses model reference variants then selects the candidate model whose associated expression is `true`, and ignores all the other models. Compilation then proceeds exactly as if you had entered the name of the selected model literally in the Model block's **Model name** field.

You can nest Model blocks that use variants to any level, allowing you to define any number of arbitrarily complex customized models within a single framework. No matter how many simulation environments you define, selecting one requires only setting variable or parameter values appropriately in the base workspace. See Setting Up Model Variants for more information.

Protected Referenced Models

A *protected model* is a referenced model from which all block and line information has been eliminated. Protecting a model does not use encryption technology. A protected model can be distributed without revealing the intellectual property that it embodies. The model is said to run in Protected mode, and gives the same results that its source model does when run in Accelerator mode.

You can use a protected model much as you could any referenced model that executes in Accelerator mode. Simulink tools work with protected models to the extent possible given that the model's contents are obscured. For example, the Model Explorer and the Model Dependency Viewer show the hierarchy under an ordinary referenced model, but not under a protected model. Signals in a protected model cannot be logged, because the log could reveal information about the protected model's contents.

When a referenced model requires object definitions or tunable parameters that are defined in the MATLAB base workspace, the protected version of the model may need some or all of those same definitions when it executes as part of a third-party model. Simulink provides techniques for identifying and obtaining the needed data. You can use the Simulink Manifest Tools or other techniques to package the model and any data for delivery.

Protecting a model requires a Real-Time Workshop license, which makes code generation capabilities available for use internally when creating the protected version of the model. The receiver of a protected model does not need a Real-Time Workshop license to use the model, and cannot use Real-Time Workshop to generate code for the model or any model that references it.

To accommodate protected models, the Model block now accepts a suffix in the **Model name** field. This suffix can be `.mdl` for an unprotected model or `.mdlp` for a protected model. If the suffix is omitted, Model block first searches the MATLAB path for a block with the specified name and the suffix `.mdl`. If that search fails, the block searches the path for a model with the suffix `.mdlp`.

The Model block now has a field named `ProtectedModel`, a boolean that indicates whether the referenced model is protected, and three fields for representing the name of the referenced model in different formats: `ModelNameDialog`, `ModelName`, and `ModelFile`. See the Model block parameters in Ports & Subsystems Library Block Parameters for information about these parameters. For more information about protecting models, see Protecting Referenced Models.

Simulink Manifest Tools

Enhanced Simulink Manifest Tools now discover and analyze model variants, protected models, and Simscape files.

New manifest analysis options for controlling whether to report file dependency locations for user files, all files, or no files. For example, you may not want to view the file locations of all the dependencies on MathWorks products. This is typical if your main use of Simulink Manifest Tools is to discover and package all the required files for your model. By not analyzing file locations, you speed up report creation, and the report is smaller and easier to navigate. If you need to trace all dependencies to understand why a particular file or toolbox is required by a model, you can always regenerate the full report of all files.

The manifest report is enhanced with sortable columns, and now MATLAB Programs as well as P-files are reported in the manifest if both exist.

For more information, see Model Dependencies in the Simulink User's Guide.

S-Function Builder

The S-Function Builder has been enhanced to support bus signals for managing complex signal interfaces. See Developing S-Functions for more information.

Embedded MATLAB Function Blocks

Support for Variable-Size Arrays and Matrices

Embedded MATLAB Function blocks now support variable-size arrays and matrices with known upper bounds. With this feature, you can define inputs, outputs, and local variables to represent data that varies in size at runtime.

Change in Text and Visibility of Parameter Prompt for Easier Use with Fixed-Point Advisor and Fixed-Point Tool

The **Lock output scaling against changes by the autoscaling tool** check box is now **Lock data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This enhancement enables you to lock the current data type settings on the dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

New Compilation Report for Embedded MATLAB Function Blocks

The new compilation report provides compile-time type information for the variables and expressions in your Embedded MATLAB functions. This information helps you find the sources of error messages and understand type propagation issues, particularly for fixed-point data types. For more information, see Working with MATLAB Function Reports in the Simulink User's Guide.

Compatibility Considerations

The new compilation report is not supported by the MATLAB internal browser on Sun™ Solaris™ 64-bit platforms. To view the compilation report on Sun Solaris 64-bit platforms, you must configure your MATLAB Web preferences to use an external browser, for example, Mozilla Firefox. To learn how to configure your MATLAB Web preferences, see Web Preferences in the MATLAB documentation.

New Options for Controlling Run-time Checks for Faster Performance

In simulation, the code generated for Embedded MATLAB Function blocks includes various run-time checks. To reduce the size of the generated code, and potentially

improve simulation times, you can use new **Simulation Target** configuration parameters to control whether or not your generated code performs:

- Integrity checks to detect violations of memory integrity in the generated code. For more information, see Ensure memory integrity in the Simulink Graphical User Interface.
- Responsiveness checks to periodically check for Ctrl+C breaks and refresh graphics. For more information, see Ensure responsiveness in the Simulink Graphical User Interface.

Embedded MATLAB Function Blocks Improve Size Propagation Behavior

Heuristics for size propagation have improved for underspecified models. During size propagation, Embedded MATLAB Function blocks no longer provide default sizes. Instead, for underspecified models, Simulink gets defaults from other blocks that have more size information.

Compatibility Considerations

Certain underspecified models that previously ran without error may now generate size mismatch errors. Examples of underspecified models include:

- Models that contain a cycle in which no block specifies output size
- Models that do not specify the size of input ports

To eliminate size mismatch errors:

- Specify sizes for the input ports of your subsystem or model.
- Specify sizes of all ports on at least one block in any loop in your model.

Simulink Data Management

New Function Exports Workspace Variables and Values

The new Simulink.saveVars function can save workspace variables and their values into a MATLAB file. The file containing the data is human-readable and can be manually edited. If Simulink cannot generate MATLAB code for a workspace variable, `Simulink.saveVars` saves that variable into a companion MAT-file rather than a MATLAB file. Executing the MATLAB file (which also loads any companion MAT file) restores the saved variables and their values to the workspace. See Simulink.saveVars for more information.

New Enumerated Constant Block Outputs Enumerated Data

Although the Constant block can output enumerated values, it provides many block parameters that do not apply to enumerated types, such as **Output minimum** and **Output maximum**. In R2009b, the **Sources** library includes the Enumerated Constant block. When you need a block that outputs constant enumerated values, use Enumerated Constant rather than Constant to avoid seeing irrelevant block parameters.

Enhanced Switch Case Block Supports Enumerated Data

The Switch Case block now supports enumerated data types for the input signal and case conditions. For more information, see Enumerations and Modeling and the Switch Case block documentation.

Code for Multiport Switch Block Shows Enumerated Values

In previous releases, generated code for a Multiport Switch block that uses enumerated data contains the underlying integer for each enumerated value rather than its name. In R2009b, the code contains the name of each enumerated value rather than its underlying integer. This change adds readability and facilitates comparing the code with the model, but has no effect on the behavior of the code. For more information, see Enumerations and Modeling and Multiport Switch.

Data Class Infrastructure Partially Deprecated

Some classes and properties in the Simulink data class infrastructure have been deprecated in R2009b. See Working with Data for information about Simulink data classes.

Compatibility Considerations

If you use any of the deprecated constructs, Simulink posts a warning that identifies the construct and describes one or more techniques for eliminating it. The techniques differ depending on the construct. You can ignore these warnings in R2009b, but MathWorks recommends making the described changes now because the deprecated constructs may be removed from future releases, upgrading the warnings to errors.

Saving Simulation Results to a Single Object

Enhanced `sim` command that saves all simulation results to a single object for easier management of simulation results.

Simulation Restart in R2009b

In order to restart an R2009a simulation in R2009b, you should first regenerate the initial SimState in R2009b.

Compatibility Considerations

The SimState that Simulink saves from a R2009a simulation might be incompatible with the internal representation of the same model in R2009b. Simulink detects this incompatibility when the R2009a SimState is used to restart a R2009b simulation. If the mismatch resides in the model interface only, then Simulink issues a warning. (You can use the Simulink diagnostic 'SimState interface checksum mismatch' to turn off such warnings or to direct Simulink to report an error.) However, if the mismatch resides in the structural representation of the model, then Simulink reports an error. To avoid these errors and warnings, you need to regenerate the initial SimState in R2009b.

Removing Support for Custom Floating-Point Types in Future Release

Support for custom floating-point types, `float(TotalBits, ExpBits)`, will be removed in a future release.

In R2009b, Simulink continues to process these types.

For more information, see float.

Simulink File Management

Removal of Functions

The following functions are no longer available:

- adams.m
- euler.m
- gear.m
- linsim.m
- rk23.m
- rk45.m

Deprecation of SaveAs to R12 and R13

In R2009b, you will no longer be able to use the SaveAs feature to save a model to releases R12 or R13. You will, however, be able to save models to R12 and R13 using the command-line. In R2010a, the command-line capability will also be removed.

Improved Behavior of Save_System

When you use the save_system function to save a model to an earlier release, you will no longer receive a dialog box that indicates that the save was successful.

Simulink Signal Management

Variable-Size Signals

New capability that allows signal sizes to change during execution facilitates modeling of systems with varying environments, resources, and constraints. For Simulink models that demonstrate using variable-size signals, see Working with Variable-Size Signals

Simulink Support

- Referenced Model
- Simulink Accelerator and Rapid Accelerator
- Bus Signals
- C-mex S-function
- Level-2 M-file S-function
- Simulink Debugger
- Signal Logging and Loading
- Block Run-Time Object

Simulink Block Support

Support for variable-size signal inputs and outputs in over 40 Simulink blocks including many blocks from the Math Operations library. For a list of Simulink blocks, see Simulink Block Support for Variable-Size Signals

Block Enhancements

New Turnkey PID Controller Blocks for Convenient Controller Simulation and Tuning

You can implement a continuous- or discrete-time PID controller with just one block by using one of the new PID Controller and PID Controller (2DOF) blocks. With the new blocks, you can:

- Configure your controller in any common controller configuration, including PID, PI, PD, P, and I.
- Tune PID controller gains either manually in the block or automatically in the new PID Tuner. (PID Tuner requires a Simulink Control Design license.)
- Generate code to implement your controller using any Simulink data type, including fixed-point data types (requires a Real-Time Workshop license).

You can set many options in the PID Controller and PID Controller (2DOF) blocks, including:

- Ideal or parallel controller configurations
- Optional output saturation limit with anti-windup circuitry
- Optional signal-tracking mode for bumpless control transfer and multiloop controllers
- Setpoint weighting in the PID Controller (2DOF) block

The blocks are available in the Continuous and Discrete libraries. For more information on using the blocks, see the PID Controller and PID Controller (2DOF) reference pages. For more information on tuning the PID blocks, see Automatic PID Tuning in the Simulink Control Design reference pages.

New Enumerated Constant Block Outputs Enumerated Data

Although the Constant block can output enumerated values, it provides many block parameters that do not apply to enumerated types, such as **Output minimum** and **Output maximum**. In R2009b, the **Sources** library includes the Enumerated Constant block. When you need a block that outputs constant enumerated values, use Enumerated Constant rather than Constant to avoid seeing irrelevant block parameters.

Enhanced Switch Case Block Supports Enumerated Data

The Switch Case block now supports enumerated data types for the input signal and case conditions. For more information, see Enumerations and Modeling and the Switch Case block documentation.

Code for Multiport Switch Block Shows Enumerated Values

In previous releases, generated code for a Multiport Switch block that uses enumerated data contains the underlying integer for each enumerated value rather than its name. In R2009b, the code contains the name of each enumerated value rather than its underlying integer. This change adds readability and facilitates comparing the code with the model, but has no effect on the behavior of the code. For more information, see Enumerations and Modeling and Multiport Switch.

Discrete Transfer Fcn Block Has Performance, Data Type, Dimension, and Complexity Enhancements

The following enhancements apply to the Discrete Transfer Fcn block:

- Improved numerics and run-time performance of outputs and states by reducing the number of divide operations in the filter to one
- Support for signed fixed-point and signed integer data types
- Support for vector and matrix inputs
- Support for input and coefficients with mixed complexity
- A new **Initial states** parameter for entering nonzero initial states
- A new **Optimize by skipping divide by leading denominator coefficient (a0)** parameter that provides more efficient implementation by eliminating all divides when the leading denominator coefficient is one. This enhancement provides optimized block performance.

Compatibility Considerations

Due to these enhancements, you might encounter the following compatibility issues:

- **Realization parameter removed**

The Real-Time Workshop software `realization` parameter has been removed from this block. You can no longer use the `set_param` and `get_param` functions on this block parameter. The generated code for this block has been improved to be similar to the former `'sparse'` realization when the **Optimize by skipping divide by leading denominator coefficient (a0)** parameter is selected, while maintaining tunability as in the former `'general'` realization when the parameter is not selected.

- **State changes**

  Due to the reduction in the number of divide operations that the block performs, you might notice that your logged states have changed when the leading denominator coefficient is not one.

Lookup Table (n-D) Block Supports Parameter Data Types Different from Signal Data Types

The Lookup Table (n-D) block supports breakpoint data types that differ from input data types. This enhancement provides these benefits:

- Lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Sharing of prescaled breakpoint data between two Lookup Table (n-D) blocks with different input data types
- Sharing of custom storage breakpoint data in generated code for blocks with different input data types

The Lookup Table (n-D) block supports table data types that differ from output data types. This enhancement provides these benefits:

- Lower memory requirement for storing table data that uses a smaller type than the output signal
- Sharing of prescaled table data between two Lookup Table (n-D) blocks with different output data types
- Sharing of custom storage table data in generated code for blocks with different output data types

The Lookup Table (n-D) block also supports separate data type specification for intermediate results. This enhancement enables use of a higher precision for internal computations than for table data or output data.

For consistency with other lookup table blocks, the **Process out-of-range input** parameter prompt is now **Action for out-of-range input**. Similarly, the command-line parameter is now `ActionForOutOfRangeInput`. For backward compatibility, the old command-line parameter `ProcessOutOfRangeInput` continues to work. The parameter settings also remain the same: `None`, `Warning`, or `Error`.

Reduced Memory Use and More Efficient Code for Evenly Spaced Breakpoints in Prelookup and Lookup Table (n-D) Blocks

For the Prelookup and Lookup Table (n-D) blocks, the generated code now stores only the first breakpoint, spacing, and number of breakpoints when:

- The breakpoint data is nontunable.
- The index search method is `Evenly spaced points`.

This enhancement reduces memory use and provides faster code execution. Previously, the code stored all breakpoint values in a set, regardless of the tunability or spacing of the breakpoints.

The following enhancements also provide more efficient code for the two blocks:

Block	Enhancement for Code Efficiency
Lookup Table (n-D)	Removal of unnecessary bit shifts for calculating the fraction
Prelookup and Lookup Table (n-D)	Use of simple division instead of computation-expensive function calls for calculating the index and fraction

Math Function Block Computes Reciprocal of Square Root

The Math Function block now supports a new function for computing the reciprocal of a square root: `1/sqrt`. You can use one block instead of two separate blocks for this computation, resulting in smaller block diagrams.

You can select one of two methods for computing the reciprocal of a square root: `Exact` or `Newton-Raphson`. Both methods support real input and output signals. When you use the `Newton-Raphson` method, you can also specify the number of iterations to perform the algorithm.

Math Function Block Enhancements for Real-Time Workshop Code Generation

The Math Function block now supports Real-Time Workshop code generation in these cases:

- Complex input and output signals for the `pow` function, for use with floating-point data types
- Fixed-point data types with fractional slope and nonzero bias for the `magnitude^2`, `square`, and `reciprocal` functions

Relational Operator Block Detects Signals That Are Infinite, NaN, or Finite

The Relational Operator block now includes `isInf`, `isNaN`, and `isFinite` functions to detect signals that are infinite, NaN, or finite. These new functions support real and complex input signals. If you select one of these functions, the block changes automatically to one-input mode.

Changes in Text and Visibility of Dialog Box Prompts for Easier Use with Fixed-Point Advisor and Fixed-Point Tool

The **Lock output scaling against changes by the autoscaling tool** check box is now **Lock output data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type for the output, such as `fixdt(1,16,0)`. This check box is now visible for any output data type specification. This enhancement helps you lock the current data type settings on a dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

This enhancement applies to the following blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference

- Discrete Derivative
- Discrete-Time Integrator
- Divide
- Dot Product
- Fixed-Point State-Space
- Gain
- Inport
- Lookup Table
- Lookup Table (2-D)
- Lookup Table Dynamic
- Math Function
- MinMax
- Multiport Switch
- Outport
- Prelookup
- Product
- Product of Elements
- Relay
- Repeating Sequence Interpolated
- Repeating Sequence Stair
- Saturation
- Saturation Dynamic
- Signal Specification
- Switch

The **Lock scaling against changes by the autoscaling tool** check box is now **Lock data type settings against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This enhancement helps you lock the current data type settings on a dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

This enhancement applies to the following blocks:

- Discrete FIR Filter
- Interpolation Using Prelookup
- Lookup Table (n-D)
- Sum
- Sum of Elements

Direct Lookup Table (n-D) Block Enhancements

The Direct Lookup Table (n-D) block now supports:

- Direct entry of **Number of table dimensions**
- Entry of **Table data** using the Lookup Table Editor

Previously, entering an integer greater than 4 for the **Number of table dimensions** required editing **Explicit number of table dimensions**. This extra parameter no longer appears on the block dialog box. For backward compatibility, scripts that contain `explicitNumDims` continue to work.

The other parameters for the block have changed as follows. For backward compatibility, the old command-line parameters continue to work.

Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of table dimensions	`maskTabDims`	`NumberOfTableDimensions`
Inputs select this object from table	`outDims`	`InputsSelectThisObjectFromTable`
Make table an input	`tabIsInput`	`TableIsInput`
Table data	`mxTable`	`Table`
Action for out-of-range input	`clipFlag`	`ActionForOutOfRangeInput`
Sample time	`samptime`	`SampleTime`

The read-only **BlockType** parameter has also changed from `S-Function` to `LookupNDDirect`.

Compatibility Considerations

In R2009b, signal dimension propagation can behave differently from previous releases. Your model might not compile under these conditions:

- A Direct Lookup Table (n-D) block is in a source loop.
- Underspecified signal dimensions exist.

If your model does not compile, set dimensions explicitly for underspecified signals.

Unary Minus Block Enhancements

Conversion of the Unary Minus block from a masked S-Function to a core block enables more efficient simulation of the block.

You can now specify sample time for the block. The **Saturate to max or min when overflows occur** check box is now **Saturate on integer overflow**, and the command-line parameter is now `SaturateOnIntegerOverflow`. For backward compatibility, the old command-line parameter `DoSatur` continues to work.

The read-only **BlockType** parameter has also changed from `S-Function` to `UnaryMinus`.

Weighted Sample Time Block Enhancements

Conversions of the Weighted Sample Time and Weighted Sample Time Math blocks from masked S-Functions to core blocks enable more efficient simulation of the blocks.

The following parameter changes apply to both blocks. For backward compatibility, the old command-line parameters continue to work.

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Output data type mode	Output data type	`OutputDataType ScalingMode`	`OutDataTypeStr`
Saturate to max or min when overflows occur	Saturate on integer overflow	`DoSatur`	`SaturateOnIntegerOverflow`

The read-only **BlockType** parameter has also changed from `S-Function` to `SampleTimeMath`.

Switch Case Block Parameter Change

For the Switch Case block, the command-line parameter for the **Show default case** check box is now `ShowDefaultCase`. For backward compatibility, the old command-line parameter `CaseShowDefault` continues to work.

Signal Conversion Block Parameter Change

For the Signal Conversion block, the parameter prompt for the **Override optimizations and always copy signal** check box is now **Exclude this block from 'Block reduction' optimization**.

Compare To Constant and Compare To Zero Blocks Use New Default Setting for Zero-Crossing Detection

The **Enable zero-crossing detection** parameter is now `on` by default for the Compare To Constant and Compare To Zero blocks. This change provides consistency with other blocks that support zero-crossing detection.

Signal Builder Block Change

You can no longer see the system under the Signal Builder block mask. In previous releases, you could right-click this block and select **Look Under Mask**.

In the Model Explorer, the Signal Builder block no longer appears in the Model Hierarchy view. In previous releases, this view was visible.

User Interface Enhancements

Context-Sensitive Help for Simulink Blocks in the Continuous Library

R2009b introduces context-sensitive help for parameters that appear in Simulink blocks of the Continuous library. This feature provides quick access to a detailed description of the block parameters.

To use the context-sensitive help:

1 Place your pointer over the label of a parameter and right-click.

2 A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu that appears after right-clicking the **Enable zero-crossing detection** parameter for the PID Controller block.
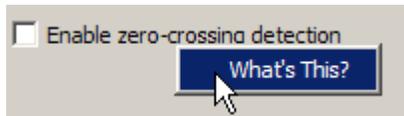
3 Click **What's This?** A window appears showing a description of the parameter.

Adding Blocks from a Most Frequently Used Blocks List

If you are using the same block repeatedly in a model, then you can save time by using the:

- **Most Frequently Used Blocks** tab in the Library Browser
- **Most Frequently Used Blocks** context menu option in the Model Editor

These features provide quick access to blocks you have added to models frequently. For details, see Adding Frequently Used Blocks.

Highlighting for Duplicate Inport Blocks

The **Highlight to Destination** option for a signal provides more information now for duplicate inport blocks. Applying this option to a signal of an inport block that has duplicate blocks highlights:

- The signal and destination block for that signal
- The signals and destination blocks of the duplicate blocks at the currently opened level in the model

Using the Model Explorer to Add a Simulink.NumericType Object

You can add a Simulink.NumericType object to the model workspace using the Model Explorer, provided you do not enable the **Is alias** option.

An example of when you might use this feature is when you:

- Want to define user-defined data types together in the model
- Do not need to preserve the data type name in the model or in the generated code

Block Output Display Dialog Has OK and Cancel Buttons

The **Block Output Display** dialog now includes **OK** and **Cancel** buttons to specify whether or not to apply your option settings.

Improved Definition of Hybrid Sample Time

Historically, you could not use the hybrid sample time to effectively identify a multirate subsystem or block. A subsystem was marked as "hybrid" and colored in yellow whether it contained two discrete sample times or one discrete sample time and one or more blocks with constant sample time [inf, 0]. Now, in R2009b, the check for the hybrid attribute no longer includes constant sample times, thereby improving the usefulness of the hybrid sample time color in identifying subsystems (and blocks) that are truly multirate.

Find Option in the Model Advisor

In R2009b, the Model Advisor includes a **Find** option to help you find checks. The find option, accessible through the **Edit** menu, allows you to find checks and folders more easily by searching names and analysis descriptions.

For more information, see Overview of the Model Advisor Window.

R2009a

Version: 7.3

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Saving and Restoring the Complete SimState

Use the new SimState feature to save the complete simulation state. Unlike the final states stored in earlier versions of Simulink, the SimState contains the complete simulation state of the model (including block states that are logged). You can then restore the state at a later time and continue simulation from the exact instant at which you stopped the simulation.

Save Simulink Profiler Results

Save the results of the Simulink Profiler and later regenerate reports for review or for comparison.

Component-Based Modeling

Port Value Displays in Referenced Models

In R2009a, port value displays can appear for blocks in a Normal mode referenced model. To control port value displays, choose **View > Port Values** in the model window. For complete information about port value displays, see Displaying Port Values.

Parallel Builds Enable Faster Diagram Updates for Large Model Reference Hierarchies In Accelerator Mode

R2009a provides potentially faster diagram updates for models containing large model reference hierarchies by building referenced models that are configured in Accelerator mode in parallel whenever possible. For example, updating of each model block can be distributed across the cores of a multicore host computer.

To take advantage of this feature, Parallel Computing Toolbox software must be licensed and installed in your development environment. If Parallel Computing Toolbox software is available, updating a model diagram rebuilds referenced models configured in Accelerator mode in parallel whenever possible.

For example, to use parallel building for updating a large model reference hierarchy on a desktop machine with four cores, you could perform the following steps:

1 Issue the MATLAB command `'matlabpool 4'` to set up a pool of four MATLAB workers, one for each core, in the Parallel Computing Toolbox environment.

2 Open your model and make sure that the referenced models are configured in Accelerator mode.

3 Optionally, inspect the model reference hierarchy. For example, you can use the Model Dependency Viewer from the **Tools** menu of Model Explorer to determine, based on model dependencies, which models will be built in parallel.

4 Update your model. Messages in the MATLAB command window record when each parallel or serial build starts and finishes.

The performance gain realized by using parallel builds for updating referenced models depends on several factors, including how many models can be built in parallel for a given model referencing hierarchy, the size of the referenced models, and host machine attributes such as amount of RAM and number of cores.

The following notes apply to using parallel builds for updating model reference hierarchies:

- Parallel builds of referenced models support only local MATLAB workers. They do not support remote workers in MATLAB Distributed Computing Server™ configurations.

- The host machine should have an appropriate amount of RAM available for supporting the number of local workers (MATLAB sessions) that you plan to use. For example, setting `matlabpool` to 4 results in five MATLAB sessions on your machine, each using approximately 120 MB of memory at startup.

- The same MATLAB environment must be set up in each MATLAB worker session as in the MATLAB client session — for example, the same base workspace variables, MATLAB path settings, and so forth. You can do this using the `PreLoadFcn` callback of the top model. Since the top model is loaded with each MATLAB worker session, its preload function can be used for any MATLAB worker session setup.

Embedded MATLAB Function Blocks

Support for Enumerated Types

Embedded MATLAB Function blocks now support Simulink enumerated types and generate C code for enumerated data. See Using Enumerated Data in MATLAB Function Blocks in the Simulink documentation.

Use of Basic Linear Algebra Subprograms (BLAS) Libraries for Speed

Embedded MATLAB Function blocks now use BLAS libraries to speed up low-level matrix operations during simulation. See Speeding Up Simulation with the Basic Linear Algebra Subprograms (BLAS) Library in the Simulink documentation.

Data Management

Signal Can Resolve to at Most One Signal Object

You can resolve a named signal to a signal object. The object can then specify or validate properties of the signal. For more information, see Simulink.Signal, Using Signal Objects to Initialize Signals and Discrete States, and Using Signal Objects to Tune Initial Values.

In previous releases, you could associate a signal with multiple signal objects, provided that the multiple objects specified compatible signal attributes. In R2009a, a signal can be associated with at most one signal object. The signal can reference the object more than once, but every reference must resolve to exactly the same object. A different signal object that has exactly the same properties will not meet the requirement. See Multiple Signal Objects for more information.

Compatibility Considerations

A compile-time error occurs in R2009a if a model associates more than one signal object with any signal. To prevent the error, decide which object the signal will use, and delete or reconfigure all references to any other signal objects so that all remaining references resolve to the chosen signal object. See Displaying Signal Sources and Destinations for a description of techniques that you can use to trace the full extent of a signal.

"Signed" Renamed to "Signedness" in the Simulink.NumericType class

In previous releases, the Property dialog of a `Simulink.NumericType` object whose **Data type mode** was any `Fixed-point` mode showed a property named **Signed**, which was a checkbox. Selecting the checkbox specified a signed type; clearing it specified an unsigned type. The API equivalent of **Signed** was `Signed`, a Boolean whose values could be `1` (signed) or `0` (unsigned).

In R2009a, a property named **Signedness** replaces **Signed** in the Property dialog of a `Simulink.NumericType` object. You can set **Signedness** to `Signed` (the default), `Unsigned`, or `Auto`, which specifies that the object inherits its **Signedness**. The API equivalent of **Signedness** is `Signedness`, which can be 1 (signed), 0 (unsigned), or `Auto`.

For compatibility with existing models, the property `Signed` remains available in R2009a. Setting `Signed` in R2009a sets `Signedness` accordingly. Accessing `Signed` in

R2009a returns the value of `Signedness` if that value is 0 or 1, or generates an error if the value of `Signedness` is `Auto`, because that is not a legal value for `Signed`.

Do not use the `Signed` with `Simulink.NumericType` in new models; use `Signedness` instead. See Simulink.NumericType for more information.

"Sign" Renamed to "Signedness" in the Data Type Assistant

For blocks and classes that support fixed-point data types, the property **Sign** previously appeared in the Data Type Assistant when the **Mode** was `Fixed point`. In R2009a, this property appears in the Data Type Assistant as **Signedness**. Only the GUI label of the property differs; its behavior and API are unchanged in all contexts.

Tab Completion for Enumerated Data Types

Tab completion now works for enumerated data types in the same way that it does for other MATLAB classes. See Instantiating Enumerations in MATLAB for details.

Simulink File Management

Model Dependencies Tools

Enhanced file dependency analysis has the following new features:

- Files in the Simulink manifest are now recorded relative to a project root folder making manifests easier to share, compare and read. See Generate Manifests and Edit Manifests.
- Command-line dependency analysis can now report toolbox dependencies, and when discovering file dependencies you can optionally generate a manifest file. See Command-Line Dependency Analysis

Block Enhancements

Prelookup and Interpolation Using Prelookup Blocks Support Parameter Data Types Different from Signal Data Types

The Prelookup block supports breakpoint data types that differ from input data types. This enhancement provides these benefits:

- Enables lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Enables sharing of prescaled breakpoint data between two Prelookup blocks with different input data types
- Enables sharing of custom storage breakpoint data in generated code for blocks with different input data types

The Interpolation Using Prelookup block supports table data types that differ from output data types. This enhancement provides these benefits:

- Enables lower memory requirement for storing table data that uses a smaller type than the output signal
- Enables sharing of prescaled table data between two Interpolation Using Prelookup blocks with different output data types
- Enables sharing of custom storage table data in generated code for blocks with different output data types

The Interpolation Using Prelookup block also supports separate data type specification for intermediate results. This enhancement enables use of a greater precision for internal computations than for table data or output data.

Lookup Table (n-D) and Interpolation Using Prelookup Blocks Perform Efficient Fixed-Point Interpolations

Whenever possible, Lookup Table (n-D) and Interpolation Using Prelookup blocks use a faster overflow-free subtraction algorithm for fixed-point interpolation. To achieve this efficiency, the blocks use a data type of larger container size to perform the overflow-free subtraction, instead of using control-flow branches as in previous releases. Also, the generated code for fixed-point interpolation is now smaller.

Compatibility Considerations

Due to the change in the overflow-free subtraction algorithm, fixed-point interpolation in Lookup Table (n-D) and Interpolation Using Prelookup blocks might, in a few cases, introduce different rounding results from previous releases. Both simulation and code generation use the new overflow-free algorithm, so they have the same rounding behavior and provide bit-true consistency.

Expanded Support for Simplest Rounding Mode to Maximize Block Efficiency

In R2009a, support for the `Simplest` rounding mode has been expanded to enable more blocks to handle mixed floating-point and fixed-point data types:

- Abs
- Data Type Conversion Inherited
- Difference
- Discrete Derivative
- Discrete FIR Filter
- Discrete-Time Integrator
- Dot Product
- Fixed-Point State-Space
- Gain
- Index Vector
- Lookup Table (n-D)
- Math Function (for the `magnitude^2`, `reciprocal`, `square`, and `sqrt` functions)
- MinMax
- Multiport Switch
- Saturation
- Saturation Dynamic
- Sum
- Switch
- Transfer Fcn Direct Form II

- Transfer Fcn Direct Form II Time Varying
- Transfer Fcn First Order
- Transfer Fcn Lead or Lag
- Transfer Fcn Real Zero
- Weighted Sample Time
- Weighted Sample Time Math

For more information, see Rounding Mode: Simplest.

New Rounding Modes Added to Multiple Blocks

For the following Simulink blocks, the dialog box now displays `Convergent` and `Round` as possible rounding modes. These modes enable numerical agreement with embedded hardware and MATLAB results.

- Abs
- Data Type Conversion
- Data Type Conversion Inherited
- Difference
- Discrete Derivative
- Discrete FIR Filter
- Discrete-Time Integrator
- Divide
- Dot Product
- Fixed-Point State-Space
- Gain
- Index Vector
- Interpolation Using Prelookup
- Lookup Table
- Lookup Table (2-D)
- Lookup Table (n-D)
- Lookup Table Dynamic
- Math Function (for the `magnitude^2`, `reciprocal`, `square`, and `sqrt` functions)

- MinMax
- Multiport Switch
- Prelookup
- Product
- Product of Elements
- Saturation
- Saturation Dynamic
- Sum
- Switch
- Transfer Fcn Direct Form II
- Transfer Fcn Direct Form II Time Varying
- Transfer Fcn First Order
- Transfer Fcn Lead or Lag
- Transfer Fcn Real Zero
- Weighted Sample Time
- Weighted Sample Time Math

In the dialog box for these blocks, the field **Round integer calculations toward** has been renamed **Integer rounding mode**. The command-line parameter remains the same.

For more information, see Rounding Mode: Convergent and Rounding Mode: Round in the Fixed-Point Toolbox™ documentation.

Compatibility Considerations

If you use an earlier version of Simulink software to open a model that uses the `Convergent` or `Round` rounding mode, the mode changes automatically to `Nearest`.

Lookup Table (n-D) Block Performs Faster Calculation of Index and Fraction for Power of 2 Evenly-Spaced Breakpoint Data

For power of 2 evenly-spaced breakpoint data, the Lookup Table (n-D) block uses bit shifts to calculate the index and fraction, instead of division. This enhancement provides these benefits:

- Faster calculation of index and fraction for power of 2 evenly-spaced breakpoint data
- Smaller size of generated code for the Lookup Table (n-D) block

Discrete FIR Filter Block Supports More Filter Structures

The following filter structures have been added to the Discrete FIR Filter block:

- `Direct form symmetric`
- `Direct form antisymmetric`
- `Direct form transposed`
- `Lattice MA`

Running a model with these filter structures requires a Signal Processing Blockset license.

Discrete Filter Block Performance, Data Type, Dimension, and Complexity Enhancements

The following enhancements have been made to the Discrete Filter block:

- Improved numerics and run-time performance of outputs and states by reducing the number of divide operations in the filter to at most one
- Support for signed fixed-point and integer data types
- Support for vector and matrix inputs
- Support for complex inputs and filter coefficients, where inputs and coefficients can each be real or complex, independently of the other
- A new **Initial states** parameter allows you to enter non-zero initial states
- A new **Leading denominator coefficient equals 1** parameter provides a more efficient implementation by eliminating all divides when the leading denominator coefficient is one

Compatibility Considerations

Due to these enhancements, you might encounter the compatibility issues in the following sections.

Realization parameter removed.   The Real-Time Workshop software `realization` parameter has been removed from this block. You can no longer use the `set_param` and `get_param` functions on this block parameter. The generated code for this block has been improved to be similar to the former `'sparse'` realization, while maintaining tunability as in the former `'general'` realization.

State changes.   Due to the reduction in the number of divide operations performed by the block, you might notice that your logged states have changed when the leading denominator coefficient is not one.

MinMax Block Performs More Efficient and Accurate Comparison Operations

For multiple inputs with mixed floating-point and fixed-point data types, the MinMax block selects an appropriate data type for performing comparison operations, instead of using the output data type for all comparisons, as in previous releases. This enhancement provides these benefits:

- Faster comparison operations, with fewer fixed-point overflows
- Smaller size of generated code for the MinMax block

Logical Operator Block Supports NXOR Boolean Operator

In R2009a, the Logical Operator block has been enhanced with a new `NXOR` Boolean operator. When you select this operator, the block returns TRUE when an even number of inputs are TRUE. Similarly, the block returns FALSE when an even number of inputs are FALSE.

Use `NXOR` to replace serial `XOR` and `NOT` operations in a model.

Discrete-Time Integrator Block Uses Efficient Integration-Limiting Algorithm for Forward Euler Method

When you select the **Limit output** check box for the Forward Euler method, the Discrete-Time Integrator block uses only one saturation when a second saturation is unnecessary. This change in the integration-limiting algorithm provides these benefits:

- Faster integration

- Smaller size of generated code for the Discrete-Time Integrator block

Dot Product Block Converted from S-Function to Core Block

Conversion of the Dot Product block from a masked S-Function to a core block enables more efficient simulation and better handling of the block in Simulink models.

Due to this conversion, you can specify sample time and values for the output minimum and maximum for the Dot Product block. The read-only **BlockType** parameter has also changed from S-Function to DotProduct.

Compatibility Considerations

In R2009a, signal dimension propagation might behave differently from previous releases. As a result, your model might not compile under these conditions:

- Your model contains a Dot Product block in a source loop.
- Your model has underspecified signal dimensions.

If your model does not compile, set dimensions for signals that are not fully specified.

For example, your model might not compile in this case:

- Your model contains a Transfer Fcn Direct Form II Time Varying block, which is a masked S-Function with a Dot Product block in a source loop.
- The second and third input ports of the Transfer Fcn Direct Form II Time Varying block are unconnected, which results in underspecified signal dimensions.

To ensure that your model compiles in this case, connect Constant blocks to the second and third input ports of the Transfer Fcn Direct Form II Time Varying block and specify the signal dimensions for both ports explicitly.

Pulse Generator Block Uses New Default Values for Period and Pulse Width

For the Pulse Generator block, the default **Period** value has changed from 2 to 10, and the default **Pulse Width** value has changed from 50 to 5. These changes enable easier transitions between time-based and sample-based mode for the pulse type.

Random Number, Uniform Random Number, and Unit Delay Blocks Use New Default Values for Sample Time

The default **Sample time** values for the Random Number, Uniform Random Number, and Unit Delay blocks have changed:

- The default **Sample time** value for the Random Number and Uniform Random Number blocks has changed from 0 to 0.1.
- The default **Sample time** value for the Unit Delay block has changed from 1 to –1.

Trigonometric Function Block Provides Better Support of Accelerator Mode

The Trigonometric Function block now supports Accelerator mode for all cases with real inputs and Normal mode support. For more information about simulation modes, see Accelerating Models in the Simulink User's Guide.

Reshape Block Enhanced with New Input Port

The Reshape block **Output dimensionality** parameter has a new option, `Derive from reference input port`. This option creates a second input port, Ref, on the block and derives the dimensions of the output signal from the dimensions of the signal input to the Ref input port. Similarly, the Reshape block command-line parameter, `OutputDimensionality`, has the new option, `Derive from reference input port`.

Multidimensional Signals in Simulink Blocks

The following blocks were updated to support multidimensional signals. For more information, see Signal Dimensions in the Simulink User's Guide.

- Assertion
- Extract Bits
- Check Discrete Gradient
- Check Dynamic Gap
- Check Dynamic Lower Bound
- Check Dynamic Range
- Check Dynamic Upper Bound

- Check Input Resolution
- Check Static Gap
- Check Static Lower Bound
- Check Static Range
- Check Static Upper Bound
- Data Type Scaling Strip
- Wrap to Zero

Subsystem Blocks Enhanced with Read-Only Property That Indicates Virtual Status

The following subsystem blocks now have the property, IsSubsystemVirtual. This read-only property returns a Boolean value, on or off, to indicate if a subsystem is virtual.

- Atomic Subsystem
- Code Reuse Subsystem
- Configurable Subsystem
- Enabled and Triggered Subsystem
- Enabled Subsystem
- For Iterator Subsystem
- Function-Call Subsystem
- If Action Subsystem
- Subsystem
- Switch Case Action Subsystem
- Triggered Subsystem
- While Iterator Subsystem

User Interface Enhancements

Port Value Displays in Referenced Models

In R2009a, port value displays can appear for blocks in a Normal mode referenced model. To control port value displays, choose **View > Port Values** in the model window. For complete information about port value displays, see Displaying Port Values.

Print Sample Time Legend

Print the Sample Time Legend either as an option of the block diagram print dialog box or directly from the legend. In either case, the legend will print on a separate sheet of paper. For more information, see Print Sample Time Legend.

M-API for Access to Compiled Sample Time Information

New MATLAB API provides access to the compiled sample time data, color, and annotations for a specific block or the entire block diagram directly from M code.

Model Advisor Report Enhancements

In R2009a, the Model Advisor report is enhanced with:

- The ability to save the report to a location that you specify.
- Improved readability, including the ability to:
  - Filter the report to view results according to the result status. For example, you can now filter the report to show errors and warnings only.
  - Collapse and expand the folder view in the report.
  - View a summary of results for each folder in the report.

See Consulting the Model Advisor in the Simulink User's Guide.

Counterclockwise Block Rotation

This release lets you rotate blocks counterclockwise as well as clockwise (see How to Rotate a Block for more information).

Physical Port Rotation for Masked Blocks

This release lets you specify that the ports of a masked block not be repositioned after a clockwise rotation to maintain a left-to-right and top-to-bottom numbering of the ports. This enhancement facilitates use of masked blocks in mechanical systems, hydraulic systems, and other modeling applications where block diagrams do not have a preferred orientation (see Port Rotation Type for more information.)

Smart Guides

In R2009a, when you drag a block, Simulink draws lines, called smart guides, that indicate when the block's ports, center, and edges align with the ports, centers, and edges of other blocks in the same diagram. This helps you create well-laid-out diagrams (see Smart Guides for more information).

Customizing the Library Browser's User Interface

Release 2009a lets you customize the Library Browser's user interface. You can change the order in which libraries appear in the Library Browser, disable or hide libraries, sublibraries, and blocks, and add, disable, or hide items on the Library Browser's menus. See Customizing the Library Browser for more information.

Subsystem Creation Command

This release adds a command, Simulink.BlockDiagram.createSubSystem, that creates a subsystem from a specified group of blocks.

Removal of Lookup Table Designer from the Lookup Table Editor

In R2009a, the Lookup Table Designer is no longer available in the Lookup Table Editor.

Compatibility Considerations

Previously, you could select **Edit** > **Design Table** in the Lookup Table Editor to launch the Lookup Table Designer. In R2009a, this menu item is no longer available.

S-Functions

Level-1 Fortran S-Functions

In this release, if you attempt to compile or simulate a model with a Level-1 Fortran S-function, you will receive an error due to the use of the newly deprecated function 'MXCREATEFULL' within the Fortran S-function wrapper 'simulink.F'. If your S-function does not explicitly use 'MXCREATEFULL', simply recompile the S-function. If your S-function uses 'MXCREATEFULL', replace each instance with 'MXCREATEDOUBLEMATRIX' and recompile the S-function.

R2008b

Version: 7.2

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Parallel Simulations in Rapid Accelerator Mode

Simulink now has the capability to run parallel simulations in Rapid Accelerator mode using `parfor` on prebuilt Simulink models.

You can now run parallel simulations in Rapid Accelerator mode with different external inputs and tunable parameters. The `sim` command can be called from a `parfor` loop if the model does not require a rebuild.

For more information, see Running a Simulation Programmatically.

Improved Rebuild Mechanism in Rapid Accelerator Mode

Simulink now has enhanced tuning of the solver and logging parameters in Rapid Accelerator mode without requiring a rebuild.

An improved rebuild mechanism ensures that the model does not rebuild when you change block diagram parameters (e.g., stop time, solver tolerances, etc.). This enhancement significantly decreases the time for simulation in Rapid Accelerator mode.

Data Type Size Limit on Accelerated Simulation Removed

In previous releases, accelerated simulation was not supported for models that use integer or fixed-point data types greater than 32 bits in length. In this release, the acceleration limit on integer and fixed-point data type size has increased to 128 bits, the same as the limit for normal-mode, i.e., unaccelerated simulation.

New Initialization Behavior in Conditional, Action, and Iterator Subsystems

For releases prior to 2008b, at the simulation start time, Simulink initializes all blocks unconditionally and subsystems cannot reset the states. Release 2008b introduces behavior that mirrors the behavior of Real-Time Workshop. For normal simulation mode, the Simulink block initialization method (mdlInitializeConditions) can be called more than once at the start time if:

- The block is contained within a Conditional, Action, or Iterator subsystem.

- The subsystem is configured to reset states when enabled (or triggered); and the subsystem is enabled (or triggered) at the start time.

This new initialization behavior has the following effect on S-functions:

- If you need to ensure that the initialization code in the mdlInitializeConditions function runs only once, then move this initialization code into the mdlStart method. MathWorks recommends this code change as a best practice.
- The change to the block initialization method, as described above, exposed a bug in the S-function macro ssIsFirstInitCond for applications involving an S-function within a Conditional, Action or Iterator subsystem. This bug has been fixed in R2008b.

  To determine if you consequently need to update your Simulink S-functions for compatibility, compare the simulation results from R2007b or an earlier release with those of R2008b. If they differ at the start time, ssIsFirstInitCond is running more than once and you must regenerate and recompile the appropriate Simulink S-functions.

  For Real-Time Workshop, you must regenerate and recompile all S-function targets and any Real-Time Workshop target for which the absolute time is turned on. (If a third-party vendor developed your S-functions, have the vendor regenerate and recompile them for you. The vendor can use the SLDiagnostics feature to identify all S-functions in a model.)

Component-Based Modeling

Processor-in-the-Loop Mode in Model Block

In R2008b, Simulink has a new Model block simulation mode for processor-in-the-loop (PIL) verification of generated code. This feature requires Real-Time Workshop Embedded Coder software. The feature lets you test the automatically generated and cross-compiled object code on your embedded processor by easily switching between Normal, Accelerator, and PIL simulation modes in your original model. You can reuse test suites, resulting in faster iteration between model development and generated code verification. For more information, see Referenced Model Simulation Modes.

Conditionally Executed Subsystem Initial Conditions

R2008b of Simulink includes enhanced handling of initial conditions for conditionally executed subsystems, Merge blocks, and Discrete-Time Integrator blocks, improving consistency of simulation results.

This feature allows you to select simplified initialization mode for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks. The simplified initialization improves the consistency of simulation results, especially for models that do not specify initial conditions for conditional subsystem output ports, and for models that have conditionally executed subsystem output ports connected to S-functions.

Note: To use the new simplified initialization mode, you must activate this feature.

Activating This Feature for New Models

For new models, you can activate this feature as follows:

1   In the model window, select **Simulation** > **Configuration Parameters**.

    The Configuration Parameters dialog box opens.

2   Select **Diagnostics** > **Data Validity**.

    The Data Validity Diagnostics pane opens.

3 In the Model Initialization section, set **Underspecified initialization detection** to `Simplified`.

4 Select **Diagnostics** > **Connectivity**.

The Connectivity Diagnostics pane opens.

5 Set **Mux blocks used to create bus signals** to `error`.

6 Set **Bus signal treated as vector** to `error`.

7 Click **OK**.

For more information, see Underspecified initialization detection.

Migrating Existing Models

For existing models, MathWorks recommends using the Model Advisor to migrate your model to the new simplified initialization mode settings.

To migrate an existing model:

1 In the model window, select **Simulation** > **Configuration Parameters**.

The Configuration Parameters dialog box opens.

2 Select **Diagnostics** > **Data Validity**.

The Data Validity Diagnostics pane opens.

3 In the Merge Block section, set **Detect multiple driving blocks executing at the same time step** to `error`.

4 Click **OK**.

5 Simulate the model and ensure that it runs without errors.

6 Select **Tools** > **Model Advisor**.

The Model Advisor opens.

7 In the Model Advisor Task Manager, select **By Product** > **Simulink**.

8 Run **Check bus usage** in the Model Advisor.

9 Run **Check consistency of initialization parameters for Outport and Merge blocks** in the Model Advisor.

10 After you have resolved any errors identified by this check, click **Proceed** to migrate your model to simplified initialization mode.

For information on using the Model Advisor, see Consulting the Model Advisor in the Simulink User's Guide.

For information on the Model Advisor checks, see Check consistency of initialization parameters for Outport and Merge blocks.

Compatibility Considerations

Activating this feature can cause differences in simulation results, when compared to previous versions. Since you must opt-in to this feature before any changes are made, there are no issues for existing models. However, MathWorks recommends that you backup existing models before you migrate them, in case you want to return to the original behavior.

Model Block Input Enhancement

Model block inputs can now be local and reusable. This capability reduces global data usage and data copying when interfacing with code from a referenced model, which can reduce memory usage during simulation and increase the efficiency of generated code. This enhancement is always relevant, so no configuration parameter is necessary or provided to control it.

One Parameter Controls Accelerator Mode Build Verbosity

In previous releases, the `ModelReferenceSimTargetVerbose` parameter controlled verbosity when a referenced model was built for execution in Accelerator mode, as specified by the Model block's Simulation mode parameter. The `ModelReferenceSimTargetVerbose` had no GUI equivalent. See Referenced Model Simulation Modes and the Model block documentation for more information.

A different parameter, `AccelVerboseBuild`, controls the verbosity when a model is built in Simulink Accelerator mode or Rapid Accelerator mode, as specified in the **Simulation** menu. See Accelerating Models for more information. The GUI equivalent of the `AccelVerboseBuild` parameter is **Configuration Parameters > Optimization > Verbose accelerator builds**. See Verbose accelerator builds for more information.

All types of accelerated simulation entail code generation (though the code is not visible to the user) and the two verbosity parameters control whether a detailed account of the code generation process appears in the MATLAB Command Window. However, providing separate verbosity parameters for the two cases was unnecessary.

In R2008b, the `ModelReferenceSimTargetVerbose` parameter is deprecated and has no effect. The `AccelVerboseBuild` parameter (**Configuration Parameters > Optimization > Verbose accelerator builds**) now controls the verbosity for Simulink Accelerator mode, referenced model Accelerator mode, and Rapid Accelerator mode.

Another parameter, `RTWVerbose` (**Configuration Parameters > Real-Time Workshop > Debug > Verbose build**) controls the verbosity of Real-Time Workshop code generation. This parameter is unaffected by the changes to `ModelReferenceSimTargetVerbose` and `AccelVerboseBuild`.

Compatibility Considerations

In R2008b, trying to set `ModelReferenceSimTargetVerbose` generates a warning message and has no effect on verbosity. The warning says to use `AccelVerboseBuild` instead. The default for `AccelVerboseBuild` is `'off'`.

A model saved in R2008b will not include the `ModelReferenceSimTargetVerbose` parameter. An R2008b model saved to an earlier Simulink version that supports `ModelReferenceSimTargetVerbose` will include that parameter, giving it the same value that `AccelVerboseBuild` has in the R2008b version.

The effect of loading a model from an earlier Simulink version into R2008b depends on the source version:

- **Prior to R14:** Neither parameter exists, so no compatibility consideration arises.
- **R14 – R2006b:** Only `ModelReferenceSimTargetVerbose` exists. Copy its value to `AccelVerboseBuild`.
- **R2007a:** Both parameters exist but neither has a GUI equivalent. Ignore the value of `ModelReferenceSimTargetVerbose` and post no warning.
- **R2007b – R2008a:** Both parameters exist and `AccelVerboseBuild` and has a GUI equivalent. If `ModelReferenceSimTargetVerbose` is `'on'`, post a warning to use `AccelVerboseBuild` instead.

Embedded MATLAB Function Blocks

Support for Fixed-Point Word Lengths Up to 128 Bits

Embedded MATLAB Function blocks now support up to 128 bits of fixed-point precision. This increase in maximum precision from 32 to 128 bits supports generating efficient code for targets with non-standard word sizes and allows Embedded MATLAB Function blocks to work with large fixed-point signals.

Enhanced Simulation and Code Generation Options for Embedded MATLAB Function Blocks

You can now specify embeddable code generation options from the Embedded MATLAB Editor using a new menu item: **Tools** > **Open RTW Target**. Simulation options continue to be available from **Tools** > **Open Simulation Target**.

In addition, simulation and embeddable code generation options now appear in a single dialog box. For details, see "Unified Simulation and Embeddable Code Generation Options" on page 22-20.

Data Type Override Now Works Consistently on Outputs

When you enable data type override for Embedded MATLAB Function blocks, outputs with explicit and inherited types are converted to the override type. For example, if you set data type override to `true singles`, the Embedded MATLAB Function block converts all outputs to `single` type and propagates the override type to downstream blocks.

In previous releases, Embedded MATLAB Function blocks did not apply data type override to outputs with inherited types. Instead, the inherited type was preserved even if it did not match the override type, sometimes causing errors during simulation.

Compatibility Considerations

Applying data type override rules to outputs with inherited types may introduce the following compatibility issues:

- Downstream Embedded MATLAB Function blocks must be able to accept the propagated override type. Therefore, you must allow data type override for

downstream blocks for which you set output type explicitly. Otherwise, you may not be able to simulate your model.

- You might get unexpected simulation results if the propagated type uses less precision than the original type.

Improperly-Scaled Fixed-Point Relational Operators Now Match MATLAB Results

When evaluating relational operators, Embedded MATLAB Function blocks compute a common type that encompasses both input operands. In previous releases, if the common type required more than 32 bits, Embedded MATLAB Function blocks may have given different answers from MATLAB. Now, Embedded MATLAB Function blocks give the same answers as MATLAB.

Compatibility Considerations

Some relational operators generate multi-word code even if one of the fixed-point operands is not a multi-word value. To work around this issue, cast both operands to the same Fixed-Point Toolbox type (using the same scaling method and properties).

Data Management

Support for Enumerated Data Types

Simulink models now support enumerated data types. For details, see:

- Enumerations and Modeling
- Using Enumerated Data in Stateflow Charts in the Stateflow documentation
- Enumerations in the Real-Time Workshop documentation

Simulink Bus Editor Enhancements

The Simulink Bus Editor can now filter displayed bus objects by either name or relationship. See Filtering Displayed Bus Objects for details.

You can now fully customize the export and import capabilities of the Simulink Bus Editor. See Customizing Bus Object Import and Export for details.

New Model Advisor Check for Data Store Memory Usage

A new Model Advisor check posts advice and warnings about the use of Data Store Memory, Data Store Read, and Data Store Write blocks. See Check Data Store Memory blocks for multitasking, strong typing, and shadowing issues for details.

Simulink File Management

Model Dependencies Tools

Enhanced file dependency analysis can now:

- Find system target files
- Analyze `STF_make_rtw_hook` functions
- Analyze all configuration sets, not just the active set.

See Scope of Dependency Analysis in the Simulink User's Guide.

Block Enhancements

Trigonometric Function Block

R2008b provides an enhanced Trigonometric Function block to:

- Support sincos
- Provide greater floating-point consistency

Math Function Block

In Simulink 2008b, an enhanced Math Function block provides greater floating-point consistency.

Merge Block

R2008b provides enhanced handling of initial conditions for the Merge block and thus improves the consistency of simulation results.

For more information, see "Conditionally Executed Subsystem Initial Conditions" on page 22-4.

Discrete-Time Integrator Block

R2008b provides an enhanced handling of initial conditions for the Discrete-Time Integrator block and thereby improves the consistency of simulation results.

For more information, see "Conditionally Executed Subsystem Initial Conditions" on page 22-4.

Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors

In this release, Simulink software can signal an error if a block callback function, e.g., `CopyFcn`, modifies a link to a library block. For example, an error occurs if you attempt to copy a library link to a self-modifying masked subsystem whose `CopyFcn` deletes a

block contained by the subsystem. This change means that you cannot use block callback functions to create self-modifying library blocks. Mask initialization code for a library block is the only code allowed to modify the block.

Compatibility Considerations

Previous releases allowed use of block callback functions to create self-modifying library blocks. Opening, editing, or running models that contain links to such blocks can cause illegal modification errors in the current release. As a temporary work around, you can break any links in your model to a library block that uses callback functions to modify itself. The best long-term solution is to move the self-modification code to the block's mask initialization section.

Random Number Block

In the dialog box for the Random Number block, the field **Initial Seed** has been renamed **Seed**. The command-line parameter remains the same.

Signal Generator Block

The Signal Generator block now supports multidimensional signals. For a list of blocks that support multidimensional signals, see Signal Dimensions in the Simulink User's Guide.

Sum Block

The accumulator of the Sum block now applies for all input signals of any data type (for example, double, single, integer, and fixed-point). In previous releases, the accumulator of this block was limited to inputs and outputs of only integer or fixed-point data types.

Switch Block

The Switch block now supports the immediate back propagation of a known output data type to the first and third input ports. This occurs when you set the **Output data type** parameter to `Inherit: Inherit via internal rule` and select the **Require all data port inputs to have the same data type** check box. In previous releases, this back propagation did not occur immediately.

Uniform Random Number Block

In the dialog box for the Uniform Random Number block, the field **Initial Seed** has been renamed **Seed**. The command-line parameter remains the same.

User Interface Enhancements

Sample Time

The display of sample time information has been expanded to include:

- Signal lines labeling with new color-independent **Annotations**
- A new **Sample Time Legend** maps the sample time **Colors** and **Annotations** to sample times.
- A distinct color for indicating that a block and signal are asynchronous.

The section "Modeling and Simulation of Discrete Systems" has been renamed "Working with Sample Times" and has been significantly expanded to provide a comprehensive review of sample times and a discussion on the new Sample Time Legend and Sample Time Display features. For more information, see Working with Sample Times.

Model Advisor

In R2008b, the Model Advisor is enhanced with:

- A model and data restore point that provides you with the ability to revert changes made in response to advice from the Model Advisor
- Context-sensitive help available for Model Advisor checks
- Tristate check boxes that visually indicate selected and cleared checks in folders
- A system selector for choosing the system level that the Model Advisor checks

See Consulting the Model Advisor in the Simulink User's Guide.

"What's This?" Context-Sensitive Help for Commonly Used Blocks

R2008b introduces context-sensitive help for parameters that appear in the following commonly used blocks in Simulink:
Bus Creator
Bus Selector
Constant
Data Type Conversion
Demux

Discrete-Time Integrator
Gain
Inport
Integrator
Logical Operator
Mux
Outport
Product
Relational Operator
Saturation
Subsystem
Sum
Switch
Terminator
Unit Delay

This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

1   Place your cursor over the label of a parameter.

2   Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after right-clicking the **Multiplication** parameter for the Gain block.

3   Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

Compact Icon Option Displays More Blocks in Library Browser

This release introduces a compact icon option that maximizes the number of blocks and libraries visible in the Library Browser's **Library** pane without scrolling (see Library Pane).

Signal Logging and Test Points Are Controlled Independently

In previous releases, a signal could be logged only if it was also a test point. Therefore, selecting **Log signal data** in the Signal Properties dialog box automatically selected **Test point**, and disabled it so that it could not be cleared. However, a signal can be a test point without being logged, so clearing **Log signal data** did not automatically clear **Test point**. The same asymmetric behavior occurred programmatically with the underlying DataLogging and TestPoint parameters.

In R2008b, no connection exists between enabling logging for a signal and making the signal a test point. Either, both, or neither capability can be enabled for any signal. Selecting and clearing **Log signal data** therefore has no effect on the setting of **Test point**, and similarly for the underlying parameters. See Exporting Signal Data Using Signal Logging and Working with Test Points for more information.

To reflect the independence of logging and test points, the command **Test Point Indicators** in the Simulink **Format > Port/Signal Displays** menu has been renamed **Testpoint/Logging Indicators**. The effect of the command, the graphical indicators displayed, and the meaning of the underlying parameter ShowTestPointIcons, are all unchanged.

Compatibility Considerations

Scripts and practices that relied on **Log signal data** to automatically set a test point must be changed to set the test point explicitly. The relevant set_param commands are:

```
set_param(PortHandle(n),'DataLogging','on')
set_param(PortHandle(n),'TestPoint','on')
```

To disable either capability, set the relevant parameter to 'off'. See Enabling Logging for a Signal for an example.

Signal Logging Consistently Retains Duplicate Signal Regions

A *virtual signal* is a signal that graphically represents other signals or parts of other signals. Virtual signals are purely graphical entities; they have no functional or mathematical significance. The nonvirtual components of a virtual signal are called *regions*. For example, if Mux block (which is a virtual block) inputs two nonvirtual signals, the block outputs a virtual signal that has two regions. See Virtual Signals and Mux Signals for more information.

In previous releases, when a virtual signal contains duplicate regions, signal logging excluded all but one of the duplicates in some contexts, but included all of the duplicates in other contexts, giving inconsistent results. For example, if the same nonvirtual signal is connected to two input ports of a Mux block, that one signal is the source of two regions in the Mux block output. Previously, if that output was being logged in Normal mode simulation, the log object would contain data for only one of the regions, because the other was eliminated as a duplicate.

In R2008a, Simulink no longer eliminates duplicate regions when logging the output of virtual blocks like Mux or Selector blocks. Simulink now logs all regions, which appear in a `Simulink.TsArray` object. The duplicate regions have unique names as follows:

```
<signal_name>_reg<#counter>
```

This change affects signal logs and all capabilities that depend on signal logging, such as scopes and signal viewers.

Compatibility Considerations

In cases where signal logging previously omitted duplicate regions, signal logs will now be larger, and scopes and signal viewers will now show more data. This change could give the impression that the results of simulation have changed, but actually only the logging of those results has changed. No action is needed unless:

- A dependency exists on the exact size of a log or the details of its contents.
- The size and details have changed due to the inclusion of previously omitted signals.

In such a case, make changes as needed to accept the changed logging behavior. See Exporting Signal Data Using Signal Logging for more information.

Simulink Configuration Parameters

In R2008b, the following Simulink configuration parameters are updated:

Note: The command-line parameter name is not changing for these parameters.

Location	Previous Parameter	New Parameter
Solver	**States shape preservation** / `ShapePreserveControl`	**Shape preservation** / `ShapePreserveControl`

Location	Previous Parameter	New Parameter
Solver	**Consecutive min step size violations** / `MaxConsecutiveMinStep`	**Number of consecutive min steps** / `MaxConsecutiveMinStep`
Solver	**Consecutive zero crossings relative tolerance** / `ConsecutiveZCsStepRelTol`	**Time tolerance** / `ConsecutiveZCsStepRelTol`
Solver	**Zero crossing location algorithm** / `ZeroCrosAlgorithm`	**Algorithm** / `ZeroCrosAlgorithm`
Solver	**Zero crossing location threshold** / `ZCThreshold`	**Signal threshold**/ `ZCThreshold`
Solver	**Number of consecutive zero crossings allowed** / `MaxConsecutiveZCs`	**Number of consecutive zero crossings** / `MaxConsecutiveZCs`
Optimization	**Eliminate superfluous temporary variables (Expression folding)** / `ExpressionFolding`	**Eliminate superfluous local variables (Expression folding)** / `ExpressionFolding`
Optimization	**Remove internal state zero initialization** / `ZeroInternalMemoryAtStart`	**Remove internal data zero initialization** / `ZeroInternalMemoryAtStartup`

In R2008b, the following Simulink configuration parameters have moved:

Note: The command-line parameter name is not changing for these parameters.

Parameter	Old Location	New Location
Check undefined subsystem initial output	**Diagnostics > Compatibility**	**Diagnostics > Data Validity**
Check preactivation output of execution context	**Diagnostics > Compatibility**	**Diagnostics > Data Validity**
Check runtime output of execution context	**Diagnostics > Compatibility**	**Diagnostics > Data Validity**

In R2008b, the **Optimization** > **Minimize array reads using temporary variables** parameter has been obsoleted.

Model Help Menu Update

The Simulink model **Help** menu now includes links to block support tables for the following products, if they are installed.

- Simulink
- Communications Blockset™
- Signal Processing Blockset
- Video and Image Processing Blockset™

To obtain the block support tables for all of these products that are installed, select **Help** > **Block Support Table** > **All Tables**.

In previous releases, **Help** > **Block Support Table** provided such tables only for the main Simulink library.

Unified Simulation and Embeddable Code Generation Options

You can now specify both simulation and embeddable code generation options in the Configuration Parameters dialog box. The simulation options apply only to Embedded MATLAB Function blocks, Stateflow charts, and Truth Table blocks.

The following table summarizes changes that apply for Embedded MATLAB Function blocks:

Type of Model	Simulation Options	Embeddable Code Generation Options
Nonlibrary	Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box. See: • "Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box" on page 22-21	New menu item in the Embedded MATLAB Editor for specifying code generation options for nonlibrary models: **Tools** > **Open RTW Target** New options in the **Real-Time Workshop** pane of the Configuration Parameters dialog box. See:

Type of Model	Simulation Options	Embeddable Code Generation Options
	• "Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box" on page 22-23 • "Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box" on page 22-24	• "Nonlibrary Models: Enhancement for the Real-Time Workshop: Symbols Pane of the Configuration Parameters Dialog Box" on page 22-32 • "Nonlibrary Models: Enhancement for the Real-Time Workshop: Custom Code Pane of the Configuration Parameters Dialog Box" on page 22-33
Library	Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box. See: • "Library Models: Changes for the General Pane of the Simulation Target Dialog Box" on page 22-27 • "Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box" on page 22-28 • "Library Models: Changes for the Description Pane of the Simulation Target Dialog Box" on page 22-30	New menu item in Embedded MATLAB Editor for specifying custom code generation options for library models: **Tools > Open RTW Target** For a description of these options, see "Library Models: Support for Specifying Custom Code Options in the Real-Time Workshop Pane of the Configuration Parameters Dialog Box" on page 22-33.

For details about the new options, see Configuration Parameters Dialog Box in the Simulink Graphical User Interface documentation. For compatibility information, see ???.

For changes specific to Stateflow, see Unified Simulation and Embeddable Code Generation Options for Stateflow Charts and Truth Table Blocks in the Stateflow and Stateflow Coder release notes.

Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box

The following sections describe changes in the panes of the Simulation Target dialog box for nonlibrary models.

Release	Appearance
Previous	**General** pane of the Simulation Target dialog box
New	**Simulation Target** pane of the Configuration Parameters dialog box

For details, see "Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box" on page 22-25.

Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	**Custom Code** pane of the Simulation Target dialog box
New	**Simulation Target > Symbols** pane of the Configuration Parameters dialog box
New	**Simulation Target > Custom Code** pane of the Configuration Parameters dialog box

Release	Appearance

For details, see "Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box" on page 22-25.

Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box

In previous releases, the **Description** pane of the Simulation Target dialog box appeared as follows.

In R2008b, these options are no longer available. For older models where the
Description pane contained information, the text is now accessible only in the Model
Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model
Hierarchy** pane, the text appears in the **Description** field for that model.

Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box

For nonlibrary models, the following table maps each GUI option in the Simulation
Target dialog box to the equivalent in the Configuration Parameters dialog box. The
options are listed in order of appearance in the Simulation Target dialog box.

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	**Simulation Target > Enable debugging / animation**	on
General > Enable overflow detection (with debugging)	**Simulation Target > Enable overflow detection (with debugging)**	on

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Echo expressions without semicolons	**Simulation Target > Echo expressions without semicolons**	`on`
General > Build Actions	**Simulation Target > Simulation target build mode**	`Incremental build`
None	**Simulation Target > Custom Code > Source file**	`''`
Custom Code > Include Code	**Simulation Target > Custom Code > Header file**	`''`
Custom Code > Include Paths	**Simulation Target > Custom Code > Include directories**	`''`
Custom Code > Source Files	**Simulation Target > Custom Code > Source files**	`''`
Custom Code > Libraries	**Simulation Target > Custom Code > Libraries**	`''`
Custom Code > Initialization Code	**Simulation Target > Custom Code > Initialize function**	`''`
Custom Code > Termination Code	**Simulation Target > Custom Code > Terminate function**	`''`
Custom Code > Reserved Names	**Simulation Target > Symbols > Reserved names**	`{}`
Custom Code > Use these custom code settings for all libraries	None	Not applicable

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
Description > Description	None<hr>**Note:** If you load an older model that contained user-specified text in the **Description** field, that text now appears in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.	Not applicable
Description > Document Link	None	Not applicable

Note: For nonlibrary models, **Simulation Target** options in the Configuration Parameters dialog box are also available in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, you can select **Simulation Target** in the **Contents** pane to access the options.

Library Models: Changes for the General Pane of the Simulation Target Dialog Box

In previous releases, the **General** pane of the Simulation Target dialog box for library models appeared as follows.
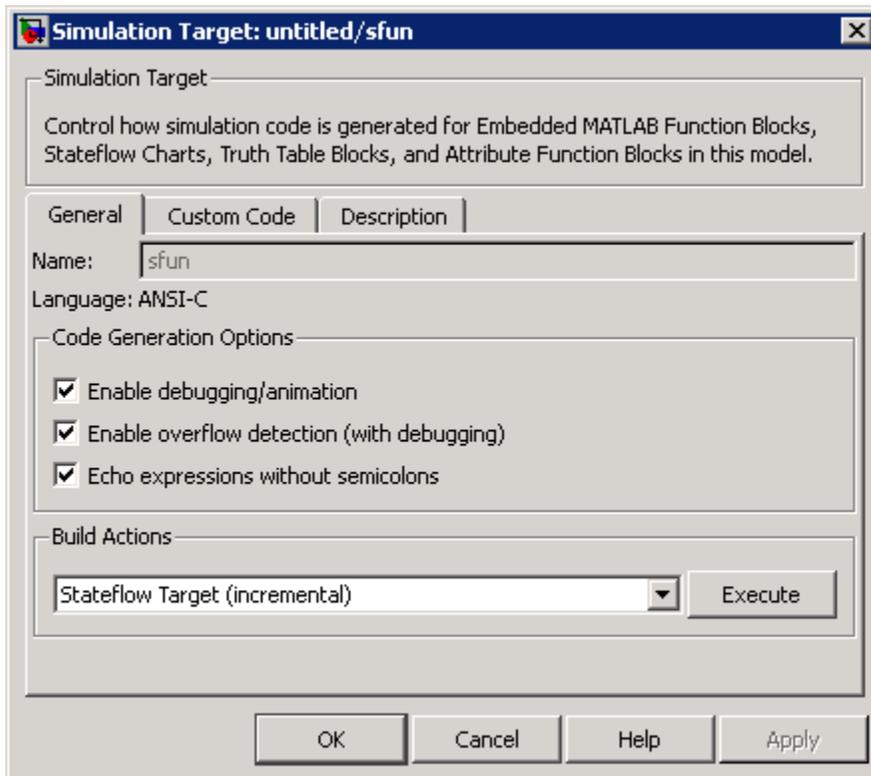
In R2008b, these options are no longer available. All library models inherit these option settings from the main model to which the libraries are linked.

Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	**Custom Code** pane of the Simulation Target dialog box
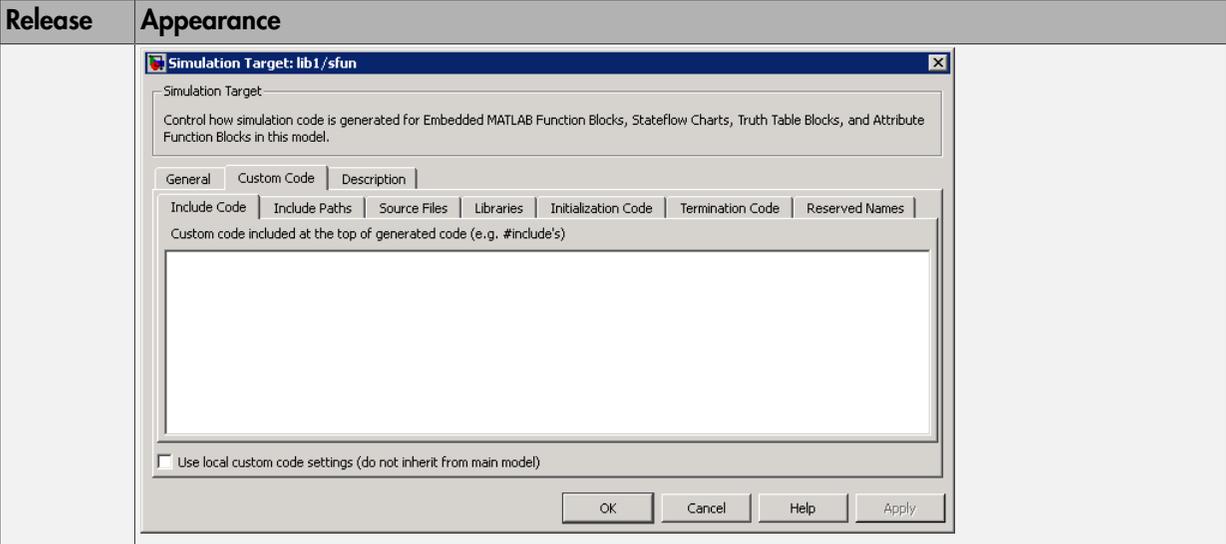
Release	Appearance
New	**Simulation Target** pane of the Configuration Parameters dialog box
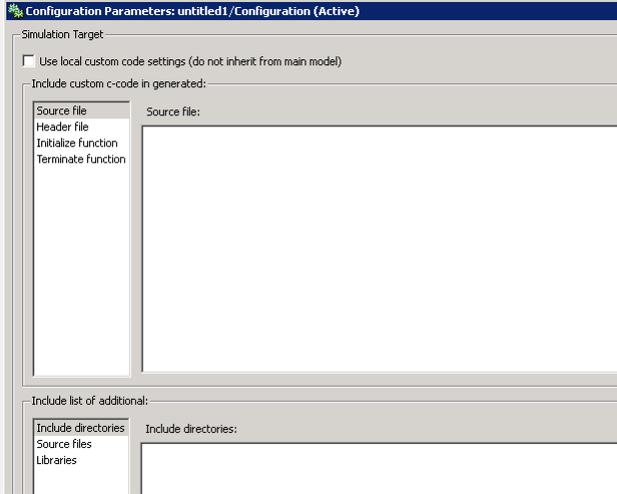
For details, see "Library Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box" on page 22-30.

Library Models: Changes for the Description Pane of the Simulation Target Dialog Box

In previous releases, the **Description** pane of the Simulation Target dialog box appeared as follows.
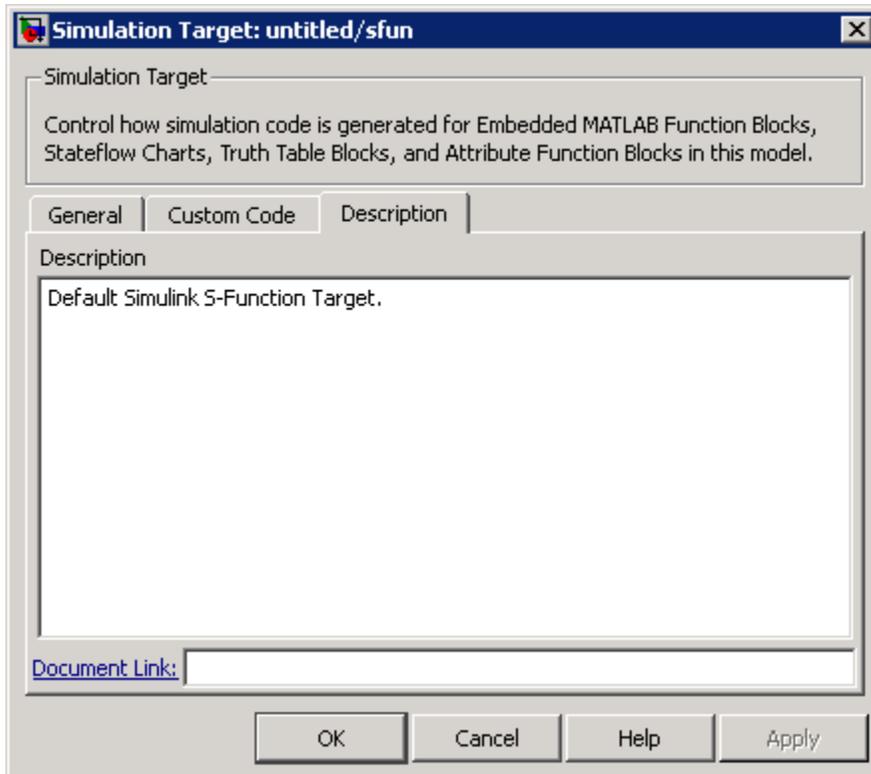
In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is discarded.

Library Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box

For library models, the following table maps each GUI option in the Simulation Target dialog box to the equivalent in the Configuration Parameters dialog box. The options are listed in order of appearance in the Simulation Target dialog box.

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	None	Not applicable
General > Enable overflow detection (with debugging)	None	Not applicable
General > Echo expressions without semicolons	None	Not applicable
General > Build Actions	None	Not applicable
None	**Simulation Target > Source file**	' '
Custom Code > Include Code	**Simulation Target > Header file**	' '
Custom Code > Include Paths	**Simulation Target > Include directories**	' '
Custom Code > Source Files	**Simulation Target > Source files**	' '
Custom Code > Libraries	**Simulation Target > Libraries**	' '
Custom Code > Initialization Code	**Simulation Target > Initialize function**	' '
Custom Code > Termination Code	**Simulation Target > Terminate function**	' '
Custom Code > Reserved Names	None	Not applicable
Custom Code > Use local custom code settings (do not inherit from main model)	**Simulation Target > Use local custom code settings (do not inherit from main model)**	off
Description > Description	None	Not applicable
Description > Document Link	None	Not applicable

> **Note:** For library models, **Simulation Target** options in the Configuration Parameters dialog box are not available in the Model Explorer.

Nonlibrary Models: Enhancement for the Real-Time Workshop: Symbols Pane of the Configuration Parameters Dialog Box

In previous releases, the **Real-Time Workshop** > **Symbols** pane of the Configuration Parameters dialog box appeared as follows.
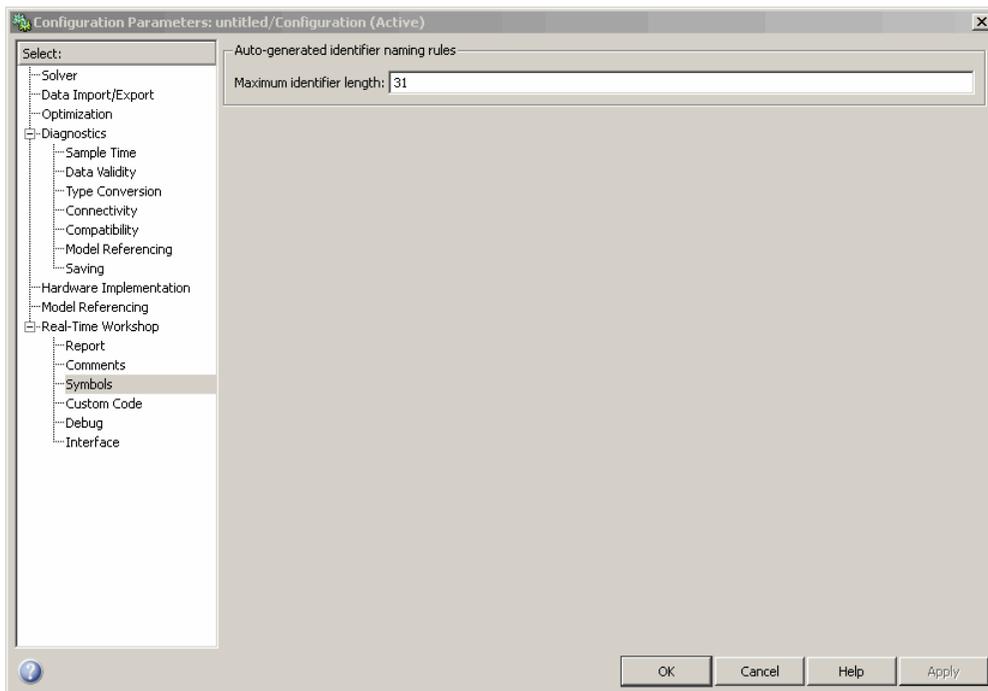
In R2008b, a new option is available in this pane: **Reserved names**. You can use this option to specify a set of keywords that the Real-Time Workshop build process should not use. This action prevents naming conflicts between functions and variables from external environments and identifiers in the generated code.

You can also choose to use the reserved names specified in the **Simulation Target** > **Symbols** pane to avoid entering the same information twice for the nonlibrary model. Select the option **Use the same reserved names as Simulation Target**.

Nonlibrary Models: Enhancement for the Real-Time Workshop: Custom Code Pane of the Configuration Parameters Dialog Box

In previous releases, the **Real-Time Workshop** > **Custom Code** pane of the Configuration Parameters dialog box appeared as follows.
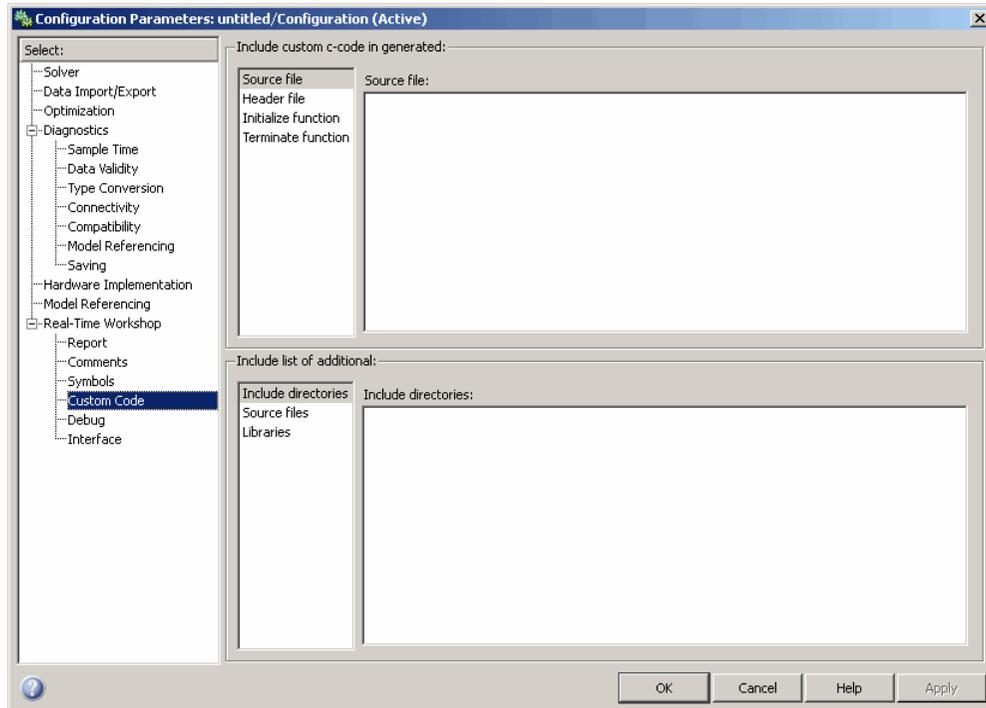
In R2008b, a new option is available in this pane: **Use the same custom code settings as Simulation Target**. You can use this option to copy the custom code settings from the **Simulation Target** > **Custom Code** pane to avoid entering the same information twice for the nonlibrary model.

Library Models: Support for Specifying Custom Code Options in the Real-Time Workshop Pane of the Configuration Parameters Dialog Box

In R2008b, you can specify custom code options in the Configuration Parameters dialog box, as shown:
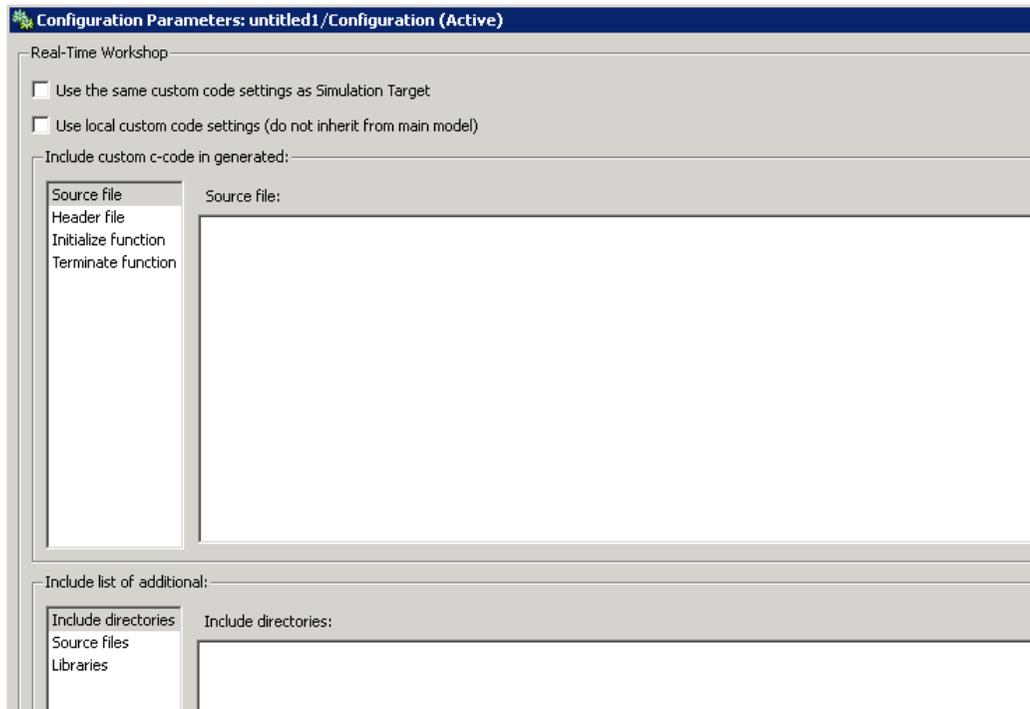
For more information, see Code Generation Pane: Custom Code in the Real-Time Workshop documentation.

Mapping of Target Object Properties to Parameters in the Configuration Parameters Dialog Box

Previously, you could programmatically set options for simulation and embeddable code generation of models containing Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks by accessing the API properties of Target objects sfun and rtw, respectively. In R2008b, the API properties of Target objects sfun and rtw are replaced by parameters that you configure using the commands get_param and set_param.

For compatibility details, see ???.

Mapping of Object Properties to Simulation Parameters for Nonlibrary Models

The following table maps API properties of the Target object `sfun` for nonlibrary models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old `sfun` Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
`CodeFlagsInfo ('debug')`	**General > Enable debugging / animation**	`SFSimEnableDebug` string - `off`, **on**	**Simulation Target > Enable debugging / animation**
`CodeFlagsInfo ('overflow')`	**General > Enable overflow detection (with debugging)**	`SFSimOverflowDetectio` string - `off`, **on**	**Simulation Target > Enable overflow detection (with debugging)**
`CodeFlagsInfo ('echo')`	**General > Echo expressions without semicolons**	`SFSimEcho` string - `off`, **on**	**Simulation Target > Echo expressions without semicolons**
`CustomCode`	**Custom Code > Include Code**	`SimCustomHeaderCode` *string* - `''`	**Simulation Target > Custom Code > Header file**
`CustomInitializer`	**Custom Code > Initialization Code**	`SimCustomInitializer` *string* - `''`	**Simulation Target > Custom Code > Initialize function**
`CustomTerminator`	**Custom Code > Termination Code**	`SimCustomTerminator` *string* - `''`	**Simulation Target > Custom Code > Terminate function**

Old `sfun` Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
ReservedNames	**Custom Code > Reserved Names**	SimReservedNameArray *string array* - **{}**	**Simulation Target > Symbols > Reserved names**
UserIncludeDirs	**Custom Code > Include Paths**	SimUserIncludeDirs *string* - **''**	**Simulation Target > Custom Code > Include directories**
UserLibraries	**Custom Code > Libraries**	SimUserLibraries *string* - **''**	**Simulation Target > Custom Code > Libraries**
UserSources	**Custom Code > Source Files**	SimUserSources *string* - **''**	**Simulation Target > Custom Code > Source files**

Mapping of Object Properties to Simulation Parameters for Library Models

The following table maps API properties of the Target object `sfun` for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old `sfun` Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	**Custom Code > Include Code**	SimCustomHeaderCode *string* - **''**	**Simulation Target > Header file**
CustomInitializer	**Custom Code > Initialization Code**	SimCustomInitializer *string* - **''**	**Simulation Target > Initialize function**

Old `sfun` Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomTerminator	**Custom Code > Termination Code**	SimCustomTerminator *string* - ''	**Simulation Target > Terminate function**
UseLocalCustomCodeSetting	**Custom Code > Use local custom code settings (do not inherit from main model)**	SimUseLocalCustomCode string - **off**, on	**Simulation Target > Use local custom code settings (do not inherit from main model)**
UserIncludeDirs	**Custom Code > Include Paths**	SimUserIncludeDirs *string* - ''	**Simulation Target > Include directories**
UserLibraries	**Custom Code > Libraries**	SimUserLibraries *string* - ''	**Simulation Target > Libraries**
UserSources	**Custom Code > Source Files**	SimUserSources *string* - ''	**Simulation Target > Source files**

Mapping of Object Properties to Code Generation Parameters for Library Models

The following table maps API properties of the Target object `rtw` for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old `rtw` Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	**Custom Code > Include Code**	CustomHeaderCode *string* - ''	**Real-Time Workshop > Header file**

Old rtw Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomInitializer	**Custom Code > Initialization Code**	CustomInitializer *string* - ''	**Real-Time Workshop > Initialize function**
CustomTerminator	**Custom Code > Termination Code**	CustomTerminator *string* - ''	**Real-Time Workshop > Terminate function**
UseLocalCustomCodeSettings	**Custom Code > Use local custom code settings (do not inherit from main model)**	RTWUseLocalCustomCode string - **off**, on	**Real-Time Workshop > Use local custom code settings (do not inherit from main model)**
UserIncludeDirs	**Custom Code > Include Paths**	CustomInclude *string* - ''	**Real-Time Workshop > Include directories**
UserLibraries	**Custom Code > Libraries**	CustomLibrary *string* - ''	**Real-Time Workshop > Libraries**
UserSources	**Custom Code > Source Files**	CustomSource *string* - ''	**Real-Time Workshop > Source files**

Compatibility Considerations

When you load and save older models in R2008b, not all target property settings are preserved.

What Happens When You Load an Older Model in R2008b

When you use R2008b to load a model created in an earlier version, dialog box options and the equivalent object properties for simulation and embeddable code generation targets migrate automatically to the Configuration Parameters dialog box, except in the cases that follow.

For the simulation target (sfun) of a nonlibrary model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the Simulation Target Dialog Box of a Nonlibrary Model	Equivalent Object Property
Custom Code > Use these custom code settings for all libraries	`ApplyToAllLibs`
Description > Description	`Description` **Note:** If you load an older model that contained user-specified text in the **Description** field, that text now appears in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.
Description > Document Link	`Document`

For the simulation target (sfun) of a library model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the Simulation Target Dialog Box of a Library Model	Equivalent Object Property
General > Enable debugging / animation	`CodeFlagsInfo('debug')`
General > Enable overflow detection (with debugging)	`CodeFlagsInfo('overflow')`
General > Echo expressions without semicolons	`CodeFlagsInfo('echo')`

Option in the Simulation Target Dialog Box of a Library Model	Equivalent Object Property
General > Build Actions	None
Custom Code > Reserved Names	`ReservedNames`
Description > Description	`Description`
Description > Document Link	`Document`

For the embeddable code generation target (`rtw`) of a library model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the RTW Target Dialog Box of a Library Model	Equivalent Object Property
General > Comments in generated code	`CodeFlagsInfo('comments')`
General > Use bitsets for storing state configuration	`CodeFlagsInfo('statebitsets')`
General > Use bitsets for storing boolean data	`CodeFlagsInfo('databitsets')`
General > Compact nested if-else using logical AND/OR operators	`CodeFlagsInfo('emitlogicalops')`
General > Recognize if-elseif-else in nested if-else statements	`CodeFlagsInfo('elseifdetection')`
General > Replace constant expressions by a single constant	`CodeFlagsInfo('constantfolding')`
General > Minimize array reads using temporary variables	`CodeFlagsInfo('redundantloadelimination')`
General > Preserve symbol names	`CodeFlagsInfo('preservenames')`
General > Append symbol names with parent names	`CodeFlagsInfo('preservenameswithparent')`
General > Use chart names with no mangling	`CodeFlagsInfo('exportcharts')`
General > Build Actions	None
Custom Code > Reserved Names	`ReservedNames`
Description > Description	`Description`

Option in the RTW Target Dialog Box of a Library Model	Equivalent Object Property
Description > Document Link	`Document`

What Happens When You Save an Older Model in R2008b

When you use R2008b to save a model created in an earlier version, parameters for simulation and embeddable code generation from the Configuration Parameters dialog box are saved. However, properties of API Target objects `sfun` and `rtw` are not saved if those properties do not have an equivalent parameter in the Configuration Parameters dialog box. In R2008b, this behavior applies even if you choose to save the model as an older version (for example, R2007a).

New Parameters in the Configuration Parameters Dialog Box for Simulation and Embeddable Code Generation

In R2008b, new parameters are added to the Configuration Parameters dialog box for simulation and embeddable code generation of models that contain Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

New Simulation Parameters for Nonlibrary Models

The following table lists the new simulation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description

`SimBuildMode`
 	
string –
`sf_incremental_build`,
`sf_nonincremental_build`,
`sf_make`, `sf_make_clean`,
`sf_make_clean_objects`
	Simulation Target > Simulation target build mode	Specifies how you build the simulation target for a model.

`SimCustomSourceCode` *string* - `''`	**Simulation Target > Custom Code > Source file**	Enter code lines to appear near the top of a generated source code file.

New Simulation Parameter for Library Models

The following table lists the new simulation parameter that applies to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimCustomSourceCode *string* - ''	**Simulation Target > Source file**	Enter code lines to appear near the top of a generated source code file.

New Code Generation Parameters for Nonlibrary Models

The following table lists the new code generation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
ReservedNameArray *string array* - **{}**	**Real-Time Workshop > Symbols > Reserved names**	Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code.
RTWUseSimCustomCode string – **off**, on	**Real-Time Workshop > Custom Code > Use the same custom code settings as Simulation Target**	Specify whether to use the same custom code settings as those specified for simulation.
UseSimReservedNames string – **off**, on	**Real-Time Workshop > Symbols > Use the same reserved names as Simulation Target**	Specify whether to use the same reserved names as those specified for simulation.

New Code Generation Parameters for Library Models

The following table lists the new code generation parameters that apply to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
CustomSourceCode *string* – ''	**Real-Time Workshop > Source file**	Enter code lines to appear near the top of a generated source code file.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
RTWUseSimCustomCode string – **off**, on	**Real-Time Workshop > Use the same custom code settings as Simulation Target**	Specify whether to use the same custom code settings as those specified for simulation.

S-Functions

Ada S-Functions

In future releases, Simulink will not have a built-in Ada S-function capability. As a mitigation strategy, call Ada code from Simulink using standard Ada 95 language features and the Simulink C-MEX S-function API. For details of this process, please contact Technical Support at MathWorks.

Legacy Code Tool Enhancement

The Legacy Code Tool data structure has been enhanced with a new S-function options field, `singleCPPMexFile`, which when set to `true`

- Requires you to generate and manage an inlined S-function as only one file (.cpp) instead of two (`.c` and `.tlc`)
- Maintains model code style—level of parentheses usage and preservation of operand order in expressions and condition expressions in `if` statements—as specified by model configuration parameters.

When you choose not to use this option, code generated by the Legacy Code Tool does not reflect code style configuration settings and requires you to manage C-MEX and TLC files.

For more information, see:

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Writing S-Functions documentation
- Integrate External Code Using Legacy Code Tool in the Real-Time Workshop documentation
- legacy_code function reference page

Compatibility Considerations

- If you upgrade from an earlier release, you can continue to use S-functions generated from the Legacy Code Tool available in earlier releases. You can continue to compile the S-function source code and you can continue to use the compiled output from an earlier release without recompiling the code.

- If you set the new `singleCPPMexFile` options field to `true`, when creating an S-function, you cannot use that S-function, in source or compiled form, with versions of Simulink earlier than Version 7.2 (R2008b).

MATLAB Changes Affecting Simulink

Changes to MATLAB Startup Options

The `matlab` command line arguments `-memmgr` and `-check_malloc` are deprecated and will be removed in a future release.

For more information, see Changes to matlab Memory Manager Startup Options in the MATLAB Release Notes.

MATLAB Graphics Tools Not Supported Under `-nojvm` Startup Option

If you start MATLAB using the command `matlab -nojvm` (which disables Java), you will receive warnings when using many graphical tools, for example, when you create figures, print Simulink models, or view Simulink scopes.

For more information, see "Changes to -nojvm Startup Option" (MATLAB) in the Desktop Tools and Development Environment release notes.

R2008a

Version: 7.1

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Rapid Accelerator

Improved Rapid Accelerator sim-command performance when running long simulations of small models on Microsoft Windows platforms.

Long Rapid Accelerator mode simulations of small models invoked by the `sim` command under the Microsoft Windows operating system now run faster.

Additional Zero Crossing Algorithm

A second zero crossing algorithm that is especially useful in systems exhibiting strong chattering behavior has been added for use with variable step solvers.

The new algorithm is selected by choosing `Adaptive` from the **Zero crossing location algorithm** option in the Solver pane of the Configuration Parameter dialog. The default algorithm is `Non-Adaptive`, which is the algorithm used prior to this release.

For more information, see Zero-Crossing Algorithms.

Component-Based Modeling

Efficient Parent Model Rebuilds

In previous releases, changing a referenced model that executed in Accelerator mode or was used for code generation triggered rebuilding every model that directly or indirectly referenced the changed model. The rebuilding occurred even if the change to the referenced model had no effect on its interface to its parent(s).

In R2008a, changing a referenced model that executes in Accelerator mode or is used for code generation triggers rebuilding a parent model only when the change directly affects the referenced model's interface to the parent model. This behavior eliminates unnecessary code regeneration, which can significantly reduce the time needed to update a diagram.

The faster diagram update has no effect on simulation behavior or performance, but may change the messages that appear in the MATLAB Command Window. See Referencing a Model for information about model referencing.

Scalar Root Inputs Passed Only by Reference

The **Configuration Parameters > Model Referencing > Pass scalar root inputs by value** option is `Off` by default, indicating that scalar root inputs are passed by reference. In previous releases, setting the option to `On` affected both simulation and generated code, and caused scalar root inputs to be passed by value. In R2008a, the option has no effect on simulation: scalar root inputs are now always passed by reference, regardless of the setting of **Pass scalar root inputs by value**. The effect of the option on code generation is the same as in previous releases. See Pass fixed-size scalar root inputs by value for code generation for more information.

Unlimited Referenced Models

In previous releases, Microsoft Windows imposed a limit on the number of models that could be referenced in Accelerator mode in a model hierarchy. This limitation is removed in R2008a. Under Microsoft Windows, as on all other platforms, the number of referenced models that can appear in a model hierarchy is effectively unlimited. See Referencing a Model for information about model referencing.

Embedded MATLAB Function Blocks

Nontunable Structure Parameters

Embedded MATLAB Function blocks now support nontunable MATLAB structure parameters. For more information, see Working with Structure Parameters in MATLAB Function Blocks.

Bidirectional Traceability

You can navigate between a line of generated code and its corresponding line of source code in Embedded MATLAB Function blocks. For more information, see Using Traceability in MATLAB Function Blocks.

Specify Scaling Explicitly for Fixed-Point Data

When you define data of fixed-point type in Embedded MATLAB Function blocks, you must specify the scaling explicitly in the General pane of the Data properties dialog box. For example, you cannot enter an incomplete specification such as `fixdt(1,16)` in the Type field. If you do not specify scaling explicitly, you will see an error message when you try to simulate your model.

To ensure that the data type definition is valid for fixed-point data, perform one of these steps in the General pane of the Data properties dialog box:

- Use a predefined option in the Type drop-down menu.
- Use the Data Type Assistant to specify the Mode as fixed-point.

Compatibility Considerations

Previously, you could omit scaling in data type definitions for fixed-point data. Such data types were treated as integers with the specified sign and word length. This behavior has changed. Embedded MATLAB Function blocks created in earlier versions may now generate errors if they contain fixed-point data with no scaling specified.

Data Management

Array Format Cannot Be Used to Export Multiple Matrix Signals

When you export signals to a workspace in `Array` format from more than one outport, none of the signals can be a matrix signal. In previous releases, violating this rule did not always cause an error, but the matrix data was not exported correctly. In R2008a, violating the rule always causes an error, and no data export occurs.

When exporting data to a workspace in `Array` format from multiple outports, use a Reshape block to convert any matrix signal to a one-dimensional (1-D) array. This restriction applies only to `Array` format. If you specify either `Structure` or `Structure with time` format, you can export matrix signals to a workspace from multiple outports without first converting the signals to vectors.

Compatibility Considerations

The more stringent error checking in R2008a can cause models that export data in `Array` format from multiple outports to generate errors rather than silently exporting matrix data incorrectly. To eliminate such errors, use a Reshape block to convert any matrix signal to a vector, or switch to `Structure` or `Structure with time` format. See Exporting Simulation Data for information about data export.

Bus Editor Upgraded

The Simulink Bus Editor has been reimplemented to provide a GUI interface similar to that of the Model Explorer, and to provide several new capabilities, including importing/exporting data from MAT-files and M-files, defining bus objects and elements with the Data Type Assistant, and creating and viewing bus hierarchies (nested bus objects). See Using the Bus Editor for details.

Changing Nontunable Values Does Not Affect the Current Simulation

In previous releases, changing the value of any variable or parameter during simulation took effect immediately. In R2008a, only changes to tunable variables and parameters take effect immediately. Other changes have no effect until the next simulation begins. This modification causes simulation behavior to match generated code behavior when the values of nontunable variables and parameters change, and it improves efficiency

by eliminating unnecessary re-evaluation. For information about parameter tuning, see Tunable Parameters and Using Tunable Parameters.

Compatibility Considerations

In R2008a, simulation behavior will differ if the behavior in a previous release depended on changing any nontunable variable or parameter during simulation. To regain the previous behavior, define as tunable any nontunable variable or parameter that you want to change during simulation for the purpose of affecting simulation immediately.

Detection of Illegal Rate Transitions

Illegal rate transitions between a block and a triggered subsystems or function call subsystems are now detected when the block is connected to a Unit Delay or Zero Hold block inside a triggered subsystem.

Compatibility Considerations

In this release, Simulink detects illegal rate transition errors when the block sample time is different from the triggered subsystem sample time in those models where the block is connected to a Unit Delay or Zero Hold block inside the triggered subsystem.

Explicit Scaling Required for Fixed-Point Data

In R2008a, when you define a fixed-point data type in a Simulink model, you must explicitly specify the scaling unless the block supports either integer scaling mode or best-precision scaling mode. If a block supports neither of these modes, you cannot define an incomplete fixed-point data type like `fixdt(1,16)`, which specifies no scaling. See Specifying a Fixed-Point Data Type and Showing Fixed-Point Details for information about defining fixed-point data types.

Compatibility Considerations

In previous releases, you could define a fixed-point data type that specified no scaling in a block that supported neither integer scaling mode nor best-precision scaling mode. The Simulink software posted no warning, and treated fixed-point data type as an integer data type with the specified word length. For example, `fixdt(1,16)` was treated as `fixdt(1,16,0)`.

In R2008a, a fixed-point data type definition that specifies no scaling generates an error unless the block supports either integer scaling mode or best-precision scaling mode. If such an error occurs when you compile a model from an earlier Simulink version, redefine the incomplete fixed-point data type to be an integer type if nothing more is needed, or to be scaled appropriately for its value range.

Fixed-Point Details Display Available

The Data Type Assistant can now display the status and details of fixed-point data types. See Specifying a Fixed-Point Data Type and Showing Fixed-Point Details for more information.

More than 2GB of Simulation Data Can be Logged on 64-Bit Platforms

When you log time, states, final states, and signals on a 64-bit platform, you can now save more simulation data in the MATLAB base workspace than was previously possible.

- When you log data using the `Structure`, `Structure with time`, or `Timeseries` format, you can now save up to $2^{48}-1$ bytes in each field that contains logged data.
- When you log data using `Array` format, you can now save up to $2^{48}-1$ bytes in each array that contains logged data.

Previously the limit was $2^{31}-1$ bytes in each field or array containing logged data. See Exporting Signal Data Using Signal Logging and Data Import/Export Pane for information about logging data.

Order of Simulink and MPT Parameter and Signal Fields Changed

The order of the fields in the `Simulink.Parameter` and `Simulink.Signal` classes, and in their respective subclasses `mpt.Parameter` and `mpt.Signal`, has changed in R2008a.

The order for `Simulink.Parameter` (and `mpt.Parameter`) is now:

```
Simulink.Parameter (handle)
        Value: []
      RTWInfo: [1x1 Simulink.ParamRTWInfo]
  Description: ''
     DataType: 'auto'
          Min: -Inf
          Max: Inf
     DocUnits: ''
   Complexity: 'real'
   Dimensions: [0 0]
```

The order for `Simulink.Signal` (and `mpt.Signal`) is now:

```
Simulink.Signal (handle)
      RTWInfo: [1x1 Simulink.SignalRTWInfo]
  Description: ''
     DataType: 'auto'
          Min: -Inf
          Max: Inf
     DocUnits: ''
   Dimensions: -1
   Complexity: 'auto'
   SampleTime: -1
 SamplingMode: 'auto'
 InitialValue: ''
```

Loading a model that uses any `Simulink.Parameter` or `mpt.Parameter` objects, and was saved in a release prior to R2008a, may post an Inconsistent Data warning in the MATLAB Command Window. This message does not indicate a problem with the model, which need not be changed. Resave the model in R2008a to update it to use the new parameter class definitions. The warning will not appear when you reopen the model.

Range Checking for Complex Numbers

Previous releases did not provide range checking for complex numbers, and attempting it generated an error. In R2008a, you can specify a minimum and/or maximum value for a complex number wherever range checking is available and a complex number is a legal value.

The specified minimum and maximum values apply separately to the real part and to the imaginary part of the complex number. If the value of either part of the number is less than the minimum, or greater than the maximum, the complex number is outside the specified range.

No range checking occurs against any combination of the real and imaginary parts, such as `(sqrt(a^2+b^2))`. See Checking Parameter Values and Signal Ranges for information about range checking.

Rate Transition Blocks Needed on Virtual Buses

In this release, Simulink never automatically inserts a Rate Transition block into a virtual bus, even if `Automatically handle rate transfer` is selected. Instead, an error is displayed indicating that a Rate Transition block must be manually inserted.

Compatibility Considerations

Some models that worked in previous releases, but were dependent on automatic Rate Transition block insertion, will now report an error and will no longer run. An error will be reported if all of these apply:

- The `Automatically handle rate transfer` option is enabled
- The model is multirate
- The model has a virtual bus, all of the elements on the bus have the same data type, and the sample time changes
- A bus selector block is not present on the virtual bus at a point after the sample time changes
- The only way to address the rate transition problem is to insert a rate transition block

Sample Times for Virtual Blocks

In models with asynchronous function calls, some virtual blocks now correctly assign generic sample times instead of triggered sample times.

Compatibility Considerations

The `CompiledSampleTime` parameter now reports the compiled sample time as generic sample time (that is, [-1, -inf]) rather than triggered sample time ([-1,-1]) for virtual blocks for which all of the flowing is true:

- The virtual block is downstream from an asynchronous source
- The virtual block is not inside a triggered subsystem
- The virtual block had a triggered ([-1,-1]) sample time in previous releases

The simulation results, code generation, and sample time colors are not affected by this change.

Signals Needing Resolution Are Graphically Indicated

In R2008a, the Simulink Editor by default graphically indicates signals that must resolve to signal objects. For any labeled signal whose **Signal name must resolve to signal object** property is enabled, a signal resolution icon appears to the left of the signal name. The icon looks like this:

A signal resolution icon indicates only that a signal's **Signal name must resolve to signal object** property is enabled. The icon does not indicate whether the signal is actually resolved, and does not appear on a signal that is implicitly resolved without its **Signal name must resolve to signal object** property being enabled. See Signal Resolution Indicators for more information.

Simulink File Management

Autosave

New Autosave option automatically creates a backup copy of models before updating or simulating. If you open or load a model which has a more recent autosave copy available, a dialog appears where you can choose to overwrite the original model file with the autosave copy.

You can set the Autosave option in the Simulink Preferences Window. See Autosave in the Simulink Graphical User Interface documentation.

Old Version Notification

New option to notify when loading a model saved in a previous version of Simulink software.

You can set this option in the Simulink Preferences Window. See Simulink Preferences Window: Main Pane in the Simulink Graphical User Interface documentation.

Model Dependencies Tools

Enhanced file dependency analysis now also detects:

- TLC files required by S-functions.
- `.fig` files created by GUIDE.
- Files referenced by common data loading functions. File names passed to `xlsread`, `importdata`, `dlmread`, `csvread`, `wk1read`, and `textread` are now identified, in addition to the existing capability for `load`, `fopen` and `imread`.

See Scope of Dependency Analysis in the Using Simulink documentation.

Block Enhancements

New Discrete FIR Filter Block Replaces Weighted Moving Average Block

The Discrete FIR Filter block in the Discrete library is new for this release. This block independently filters each channel of the input signal with the specified digital FIR filter. The Discrete FIR Filter block replaces the Weighted Moving Average block.

Compatibility Considerations

You should replace Weighted Moving Average blocks in your existing models with the Discrete FIR Filter block. To do this, run the slupdate command on your models.

Rate Transition Block Enhancements

Support for Rate Transition blocks has been enhanced in the following ways:

- Rate Transition block output port sample time now can be specified as a multiple of input port sample time, using the new Rate Transition block parameters **Output port sample time options** and **Sample time multiple (>0)**. See the Rate Transition block documentation for more information.

- In previous releases, auto-insertion of Rate Transition blocks was selected for a model using the option **Automatically handle data transfers between tasks** on the **Solver** pane of the Configuration Parameters dialog box. When selected, auto-insertion always ensured data transfer determinism for periodic tasks.

  This release allows you to control the level of data transfer determinism when auto-insertion of Rate Transition blocks is selected for your model. The **Solver** pane option for selecting auto-insertion has been renamed to **Automatically handle rate transition for data transfer**. Selecting auto-insertion now enables a new option, **Deterministic data transfer**. Selecting Never (minimum delay) or Whenever possible for this option can provide reduced latency for models that do not require determinism. See the Solver Pane section in the Simulink Graphical User Interface documentation for more information.

- Auto-insertion of Rate Transition blocks is now supported for additional rate transitions, such as sample times with nonzero offset, and between non-integer-multiple sample times.

Enhanced Lookup Table (n-D) Block

The Lookup Table (n-D) block now supports all data types, complex table data, and nonscalar inputs. See the Lookup Table (n-D) block documentation for more information.

New Accumulator Parameter on Sum Block

The Sum block dialog box displays a new parameter for specifying the data type of its accumulator. See the Sum block documentation for more information.

User Interface Enhancements

Simulink Library Browser

A new version of the Simulink Library browser has the following enhancements:

- Now available on all platforms supported by Simulink software.
- Improved performance for browsing and searching of libraries, by allowing these operations to proceed without actually loading the libraries.
- Enhanced search finds all blocks and displays search results in a separate tab.
- New option to display library blocks in a compact grid layout that conserves screen space.

Simulink Preferences Window

New unified Simulink Preferences window for configuring default settings. The new Preferences window allows you to configure file change notifications, autosave options, fonts, display options, and model configuration defaults.

See Simulink Preferences Window: Main Pane in the Simulink Graphical User Interface documentation.

Model Advisor

In R2008a, the Model Advisor tool is enhanced with improved GUI navigation, check analysis, and reports including:

- Reset option that reverts the status of all checks to **Not Run** while keeping the current check selection.
- Model Advisor Result Explorer to make changes to your model.
- **Input Parameters** to provide inputs to checks.
- Check results reported in the same order as the Model Advisor tree.
- The ability to generate reports for any folder.
- Timestamps in reports indicating when checks run at different times.

See Consulting the Model Advisor in the Simulink User's Guide.

Solver Controls

Enhanced controls in the Solver pane of the Configuration Parameters dialog. The Solver pane of the Configuration Parameters dialog has been changed as follows:

- The **Solver diagnostic controls** pane has been removed and two new panes have been added (**Tasking and sample time options**, and **Zero crossing options**)
- The `Automatically handle data transfers between tasks` control has been moved to the **Tasking and sample time options** pane, and has been renamed `Automatically handle rate transition for data transfer`
- The `Higher priority value indicates higher task priority` control has been moved to the **Tasking and sample time options** pane
- The `Number of consecutive min step size violations allowed` control has been moved to the **Solver options** pane, and has been renamed `Consecutive min step size violations allowed`
- The `States shape preservation` control has been added to the **Solver options** pane
- The `Consecutive zero crossings relative tolerance` control has been moved to the **Zero crossing options** pane
- The `Number of consecutive zero crossings allowed` control has been moved to the **Zero crossing options** pane
- The `Zero crossing control` control has been moved to the **Zero crossing options** pane
- The `Zero crossing location algorithm` control has been added to the **Zero crossing options** pane
- The `Zero crossing location threshold` control has been added to the **Zero crossing options** pane
- Options that in previous releases were only visible when enabled are now always visible. They are grayed when not enabled.

For more information on the Configuration parameters solver pane, see Solver Pane.

Compatibility Considerations

The Solver pane of the Configuration Parameter dialog has been restructured, and many parameters have moved or been renamed. Please refer to the list of changes above for information on specific parameters.

"What's This?" Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

R2008a introduces "What's This?" context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

1 Place your cursor over the label of a parameter.

2 Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.

3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

S-Functions

Simplified Level-2 M-File S-Function Template

New basic version of the Level-2 M-file S-function template `msfuntmpl_basic.m` simplifies creating Level-2 M-file S-functions. See Writing Level-2 MATLAB S-Functions in *Writing S-Functions* for more information.

Compatibility Considerations

MATLAB V7.6 (R2008a) on Linus Torvalds' Linux® platforms is now built with a compiler that utilizes glibc version 2.3.6. To work with MATLAB V7.6 (R2008a), MEX-file S-functions compiled on a Linux platform must be rebuilt.

R2007b

Version: 7.0

New Features

Bug Fixes

Compatibility Considerations

Simulation Performance

Simulink Accelerator

Simulink Accelerator™ has been incorporated into Simulink software, and a new Rapid Accelerator mode has been added for faster simulation through code generation technology. See Accelerating Models in Simulink User's Guide.

Note: When using From File blocks in Rapid Accelerator mode, the corresponding MAT file must be in the current directory.

Compatibility Considerations

A license is no longer required to use the Accelerator or Rapid Accelerator modes.

Simulink Profiler

Simulink Profiler has been incorporated into Simulink software for the identification of simulation performance bottlenecks. See Capturing Performance Data in Simulink User's Guide.

Compiler Optimization Level

Simulink Accelerator mode, Rapid Accelerator mode, and model reference simulation targets can now specify the compiler optimization level used (choose between minimizing compilation time or simulation time). See Customizing the Build Process in Simulink User's Guide.

Compatibility Considerations

The new model configuration parameter **Compiler optimization level** defaults to `Optimizations off (faster builds)`. As a result, you might notice shorter build times, but longer execution times, compared to previous releases. However, any previously defined custom compiler optimization options using `OPT_OPTS` will be honored, and model behavior should be unchanged.

Variable-Step Discrete Solver

Simulink software has been enhanced to no longer take unnecessary time steps at multiples of the maximum step size when using a variable-step discrete solver.

Referenced Models Can Execute in Normal or Accelerator Mode

In previous releases, Simulink software executed all referenced models by generating code for them and executing the generated code. In this release, Simulink software can execute appropriately configured referenced models interpretively. Such execution is called Normal mode execution, and execution via generated code is now called Accelerator mode execution. The technique of executing a referenced model via generated code has not changed, but it did not previously need a separate name because it was the only alternative.

Many restrictions that previously applied to all referenced model execution now apply only to Accelerator mode execution, and are relaxed in Normal mode. For example, some Simulink tools that did not work with referenced models because they are incompatible with generated code can now be used by executing the referenced model in Normal mode.

Normal mode also has some restrictions that do not apply to Accelerator mode. For example, at most, one instance of a given model in a referenced model hierarchy can execute in Normal mode. See Referencing a Model in Simulink User's Guide for information about using referenced models in Normal and Accelerator mode.

Accelerator and Model Reference Targets Now Use Standard Internal Functions

For more consistent simulation results, Simulink Accelerator mode, Rapid Accelerator mode, and the model reference simulation target now perform mathematical operations with the same internal functions that MATLAB and Simulink products use.

Component-Based Modeling

New Instance View Option for the Model Dependency Viewer

The Model Dependency viewer has a new option to display each reference to a model and indicate whether the reference is simulated in Accelerator or Normal mode. See Referencing a Model and Model Dependency Viewer in Simulink User's Guide.

Mask Editor Now Requires Java

The Mask Editor now requires that the MATLAB product start with Java enabled. See Simulink Mask Editor in Simulink User's Guide.

Compatibility Considerations

You can no longer use the Mask Editor if you start MATLAB with the `-nojvm` option.

Embedded MATLAB Function Blocks

Complex and Fixed-Point Parameters

Embedded MATLAB Function blocks now support complex and fixed-point parameters.

Support for Algorithms That Span Multiple M-Files

You can now generate embeddable code for external M-functions from Embedded MATLAB function blocks. This feature allows you to call external functions from multiple locations in an M-file or model and include these functions in the generated code.

Compatibility Considerations

In previous releases, Embedded MATLAB function blocks did not compile external M-functions, but instead dispatched them to the MATLAB product for execution (after warning). Now, the default behavior is to compile and generate code for external M-functions called from Embedded MATLAB function blocks. If you do not want Embedded MATLAB function blocks to compile external M-functions, you must explicitly declare them to be extrinsic, as described in Calling MATLAB Functions in the Embedded MATLAB documentation.

Loading R2007b Embedded MATLAB Function Blocks in Earlier Versions of Simulink Software

If you save Embedded MATLAB Function blocks in R2007b, you will not be able to load the corresponding model in earlier versions of Simulink software. To work around this issue, save your model in the earlier version before loading it, as follows:

1   In the Simulink Editor, select **File** > **Save As**.

2   In the **Save as type** field, select the version in which you want to load the model.

    For example, if you want to load the model in Simulink R2007a, select `Simulink 6.6/R2007a Models (*.mdl)`.

Data Management

New Diagnostic for Continuous Sample Time on Non-Floating-Point Signals

A new diagnostic detects continuous sample time on non-floating-point signals.

New Standardized User Interface for Specifying Data Types

This release introduces a new standardized user interface, the **Data Type Assistant**, for specifying data types associated with Simulink blocks and data objects, as well as Stateflow data. See Using the Data Type Assistant in Simulink User's Guide for more information.

The **Data Type Assistant** appears on the dialogs of the following Simulink blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference
- Discrete Derivative
- Discrete-Time Integrator
- Dot Product
- MATLAB Function (formally called Embedded MATLAB Function)
- Gain
- Inport
- Interpolation Using Prelookup
- Logical Operator
- Lookup Table
- Lookup Table (2-D)
- Lookup Table Dynamic
- Math Function
- MinMax

- Multiport Switch
- Outport
- Prelookup
- Product, Divide, Product of Elements
- Relational Operator
- Relay
- Repeating Sequence Interpolated
- Repeating Sequence Stair
- Saturation
- Saturation Dynamic
- Signal Specification
- Sum, Add, Subtract, Sum of Elements
- Switch
- Weighted Moving Average (obsolete — replaced by the Discrete FIR Filter block)

The **Data Type Assistant** appears on the dialogs of the following Simulink data objects:

- Simulink.BusElement
- Simulink.Parameter
- Simulink.Signal

New Block Parameters for Specifying Minimum and Maximum Values

The following new block parameters are available for specifying the minimum and maximum values of signals and other block parameters.

- **Output minimum**, **Minimum**
- **Output maximum**, **Maximum**
- **Parameter minimum**
- **Parameter maximum**

These new parameters selectively appear on the dialogs of the following Simulink blocks:

- Abs
- Constant

- Data Store Memory
- Data Type Conversion
- Difference
- Discrete Derivative
- Discrete-Time Integrator
- Gain
- Inport
- Interpolation Using Prelookup
- Lookup Table
- Lookup Table (2-D)
- Math Function
- MinMax
- Multiport Switch
- Outport
- Product, Divide, Product of Elements
- Relay
- Repeating Sequence Interpolated
- Repeating Sequence Stair
- Saturation
- Saturation Dynamic
- Signal Specification
- Sum, Add, Subtract, Sum of Elements
- Switch

New Range Checking of Block Parameters

In this release, Simulink software performs range checking of parameters associated with blocks that specify minimum and maximum values (see "New Block Parameters for Specifying Minimum and Maximum Values" on page 24-7). Simulink software alerts you when values of block parameters lie outside a range that corresponds to its specified minimum and maximum values and data type. See Checking Parameter Values in Simulink User's Guide for more information.

New Diagnostic for Checking Signal Ranges During Simulation

In the Configuration Parameters dialog, the **Diagnostics > Data Validity** pane contains a new diagnostic, **Simulation range checking**, which alerts you during simulation when blocks output signals that exceed specified minimum or maximum values (see "New Block Parameters for Specifying Minimum and Maximum Values" on page 24-7). For more information about using this diagnostic, see Signal Ranges in Simulink User's Guide.

Configuration Management

Disabled Library Link Management

The following new features help manage disabled library links and protect against accidental loss of work:

- "Disabled Link" appears in the title bar of a Model Editor window that displays a subsystem connected to a library by a disabled link.
- ToolTips for library-linked blocks include the link status as well as the destination block for the link.
- New diagnostics warn when saving a model that contains disabled or parameterized library links.
- New Model Advisor checks let you search for disabled or parameterized library links in a model.

See Disabling Links to Library Blocks in Simulink User's Guide for more information.

Model Dependencies Tools

The model dependencies manifest tools have these new capabilities:

- Enhanced analysis to detect file dependencies from Stateflow transitions, Embedded MATLAB functions, and requirements documents. See Scope of Dependency Analysis in Simulink User's Guide.
- Model dependencies tools now save user manifest edits for reuse the next time a manifest is generated. See Edit Manifests in Simulink User's Guide.

Embedded Software Design

Legacy Code Tool Enhancement

The Legacy Code Tool has been enhanced to allow the use of `void*` and `void**` to declare variables that represent memory allocated for specific instances of items such as file descriptors, device drivers, and memory managed externally.

For more information, see:

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Developing S-Functions
- legacy_code function documentation

Block Enhancements

Product Block Reorders Inputs Internally

In previous releases, a Product block whose

- **Number of inputs** parameter begins with a divide character (/)
- **Multiplication** parameter specifies `Element-wise(.*)`

computes the reciprocal of its first input before multiplying or dividing by subsequent inputs. For example, if a Product block specifies division for its first input, *u1*, and multiplication for its second input, *u2*, previous versions of Simulink software compute

```
(1 / u1) * u2
```

In this release, the Product block internally reorders its first two inputs if particular conditions apply, such that Simulink software now computes

```
u2 / u1
```

See the Product block documentation for more information.

Block Data Tips Now Work on All Platforms

In previous releases, block data tips worked only on Microsoft Windows platforms. In this release, the data tips work on all platforms. Also, the data tip for a library link, even if disabled, now includes the name of the library block it references.

Enhanced Data Type Support for Blocks

The following blocks now allow you to specify the data type of their outputs:

- Abs
- Multiport Switch
- Saturation
- Saturation Dynamic
- Switch

The following blocks now support single-precision floating-point inputs, outputs, and parameter values:

- Discrete Filter
- Discrete State-Space
- Discrete Transfer Fcn

New Simulink Data Class Block Object Properties

The following properties have been added to the Simulink.BlockData class:

- **AliasedThroughDataType**
- **AliasedThroughDataTypeID**

New Break Link Options for save_system Command

The save_system command's `BreakLink` option has been replaced by two options: `BreakAllLinks` and `BreakUserLinks`. The first option duplicates the behavior of the obsolete `BreakLink` option, i.e., it replaces all library links, including links to Simulink block libraries with copies of the referenced library blocks. The `BreakUserLinks` option replaces only links to user-defined libraries.

Compatibility Considerations

The `save_system` command continues to honor the `BreakLink` option but displays a warning message at the MATLAB command line that the option is deprecated.

Simulink Software Checks Data Type of the Initial Condition Signal of the Integrator Block

When the output port of the Constant or IC block is connected to the Initial Condition port of the Integrator block, Simulink software now compares the data type of the Initial Condition input signal of the Integrator block with the **Constant value** parameter or **Initial value** parameter of the Constant block or IC block, respectively.

Compatibility Considerations

If the data type for the output port of the Constant or IC blocks does not match the data type of the Initial Condition input signal for the Integrator block, Simulink software returns an error at compile time.

Usability Enhancements

Model Advisor

Model Advisor has been enhanced to navigate checks, display status, and report results. Also, this release contains a new Model Advisor Checks reference.

Alignment Commands

This release contains new block alignment, distribution, and resize commands to align groups of blocks along their edges, equalize interblock spacing, and resize blocks to be all the same size. See Aligning, Distributing, and Resizing Groups of Blocks Automatically in Simulink User's Guide for more information.

S-Functions

New S-Function APIs to Support Singleton Dimension Handling

The following functions have been added:

- ssPruneNDMatrixSingletonDims
- ssGetInputPortDimensionSize
- ssGetOutputPortDimensionSize

See S-Function SimStruct Functions — Alphabetical List in Developing S-Functions for more information.

New Level-2 M-File S-Function Example

This release includes a new Level-2 M-file S-function example in `sfundemos.mdl`. The Simulink model `msfcndemo_varpulse.mdl` uses the S-function `msfcn_varpulse.m` to create a variable-width pulse generator.

R2007a+

Version: 6.6.1

Bug Fixes

R2007a

Version: 6.6

New Features

Bug Fixes

Compatibility Considerations

Multidimensional Signals

This release includes support for multidimensional signals, including:

- Sourcing of multidimensional signals
- Logging or displaying of multidimensional signals
- Large-scale modeling applications, such as those from model referencing
- Buses and nonvirtual buses
- Code generation with Real-Time Workshop software
- S-functions, including Level-2 M-File S-functions
- Stateflow charts

For further details, see:

- "Multidimensional Signals in Simulink Blocks" on page 26-2
- "Multidimensional Signals in S-Functions" on page 26-4

Simulink software supports signals with up to 32 dimensions. Do not use signals with more than 32 dimensions.

Multidimensional Signals in Simulink Blocks

The following blocks were updated to support multidimensional signals. See Signal Dimensions in the Simulink documentation for further details.

- Abs
- Assignment
- Bitwise Operator
- Bus Assignment
- Bus Creator
- Bus Selector
- Compare to Constant
- Compare to Zero
- Complex to Magnitude-Angle
- Complex to Real-Imag
- Concatenate

- Constant
- Data Store Memory
- Data Store Read
- Data Store Write
- Data Type Conversion
- MATLAB Function (formally called Embedded MATLAB Function)
- Environment Controller
- From
- From Workspace
- Gain (only if the **Multiplication** parameter specifies `Element-wise(K*u)`)
- Goto
- Ground
- IC
- Inport
- Level-2 MATLAB S-Function
- Logical Operator
- Magnitude-Angle to Complex
- Manual Switch
- Math Function (no multidimensional signal support for the `transpose` and `hermitian` functions)
- Memory
- Merge
- MinMax
- Model
- Multiport Switch
- Outport
- Product, Product of Elements — only if the **Multiplication** parameter specifies `Element-wise`
- Probe
- Random Number
- Rate Transition

- Real-Imag to Complex
- Relational Operator
- Reshape
- Scope, Floating Scope
- Selector
- S-Function
- Signal Conversion
- Signal Specification
- Slider Gain
- Squeeze
- Subsystem, Atomic Subsystem, CodeReuse Subsystem
- Add, Subtract, Sum, Sum of Elements — along specified dimension
- Switch
- Terminator
- To Workspace
- Trigonometric Function
- Unary Minus
- Uniform Random Number
- Unit Delay
- Width

The Block Support Table does not list which blocks support multidimensional signals. To see if a block supports multidimensional signals, check for the entry `Multidimensionalized` in the **Characteristics** table of a block.

Multidimensional Signals in S-Functions

To use multidimensional signals in S-functions, you must use the new SimStruct function, ssAllowSignalsWithMoreThan2D.

Multidimensional Signals in Level-2 M-File S-Functions

To use multidimensional signals in Level-2 M-file S-functions, you must set the new Simulink.MSFcnRunTimeBlock property, `AllowSignalsWithMoreThan2D`.

New Block Parameters

This release introduces the following common block parameters.

- `PreCopyFcn`: Allows you to assign a function to call before the block is copied. See Block Callback Parameters in the Simulink documentation for details.
- `PreDeleteFcn`: Allows you to assign a function to call before the block is deleted. See Block Callback Parameters in the Simulink documentation for details.
- `StaticLinkStatus`: Allows you to obtain the link status of a block without updating out-of-date reference blocks See Checking and Setting Link Status Programmatically in the Simulink documentation for details.

GNU Compiler Upgrade

This release upgrades the GNU® compiler to GCC 4.0.3 on Mac platforms and GCC 4.1.1 on Linux platforms. The Fortran runtime libraries for the previous GCC 3.x versions are no longer included with MATLAB.

Compatibility Considerations

C, C++, or Fortran MEX-files built with the previous 3.x version of the GCC compiler are not guaranteed to load in this release. Rebuild the source code for these S-functions using the new version of the GCC compiler.

Changes to Concatenate Block

This release includes the following changes to the Concatenate block:

- Its **Mode** parameter provides two settings, namely, `Vector` and `Multidimensional array`.
- Its parameter dialog box contains a new option, **Concatenate dimension**, specifying the output dimension along which to concatenate the input arrays.
- The block displays a new icon when its **Mode** parameter is set to `Multidimensional array`.

This release updates Concatenate blocks when loading models created in previous releases.

Changes to Assignment Block

This release includes the following changes to the Assignment block:

- Enter the number of dimensions in the **Number of output dimensions** parameter, then configure the input and output with the **Index Option**, **Index**, and **Output Size** parameters.
- The parameter dialog box has the following new parameters:

  - **Number of output dimensions**
  - **Index Option**
  - **Index**
  - **Output Size**

- The **Initialize output (Y)** parameter replaces **Output (Y)** and has renamed options.
- The **Action if any output element is not assigned** parameter replaces **Diagnostic if not all required dimensions populated**.
- The block displays a new icon depending on the value of **Number of input dimensions** and the **Index Option** settings.

The following parameters are obsolete:

- **Input type**
- **Use index as start value**
- **Source of element indices**
- **Elements**
- **Source of row indices**
- **Rows**
- **Source of column indices**
- **Columns**
- **Output dimensions**

This release updates Assignment blocks when loading models created in previous releases.

Changes to Selector Block

This release includes the following changes to the Selector block:

- Enter the number of dimensions in the **Number of input dimensions** parameter, then configure the input and output with the **Index Option**, **Index**, and **Output Size** parameters.
- The parameter dialog box has the following new parameters:
  - **Number of input dimensions**
  - **Index Option**
  - **Index**
  - **Output Size**
- The behavior of the **Sample time** parameter has changed. See the Selector block **Sample time** parameter for details.
- The block displays a new icon depending on the value of **Number of input dimensions** and the **Index Option** settings.

The following parameters are obsolete:

- **Input type**
- **Use index as starting value**
- **Source of row indices**
- **Rows**
- **Source of column indices**
- **Columns**
- **Output port dimensions**

This release updates Selector blocks when loading models created in previous releases.

Improved Model Advisor Navigation and Display

This release improves the Model Advisor graphical user interface (GUI) for navigating lists of checks and viewing the status of completed checks. While Model Advisor functionality and content are largely unchanged from R2006b, the Model Advisor checks display and are navigated differently than in previous versions, and the generated Model Advisor report, if requested, displays in a MATLAB web browser window that is separate from the Model Advisor GUI.

To exercise the new features, open Model Advisor for a model (for example, enter `modeladvisor('vdp')` at the MATLAB command line) and then follow the instructions

in the Model Advisor window. For more information about Model Advisor navigation and display, see Consulting the Model Advisor in the Simulink documentation.

Change to Simulink.ModelAdvisor.getModelAdvisor Method

In this release, when using the `getModelAdvisor` method defined by the Simulink.ModelAdvisor class to change Model Advisor working scope to a different model, you must either close the previous model or invoke the `getModelAdvisor` method with `'new'` as the second argument. For example, if you previously set scope to modelX with

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
```

and you want to change scope to modelY, you must either close modelX or use

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY', 'new');
```

If you try to change scope between models without the `'new'` argument, an error message is displayed.

Compatibility Considerations

In previous releases, you could change Model Advisor working scope without closing the current session. This is no longer allowed.

If your code contains a code pattern such as the following,

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
...
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY');
```

you must add the `'new'` argument to the second and subsequent invocations of `getModeladvisor`. For example:

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
...
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY', 'new');
```

Alternatively, you can close ModelX before issuing
`Simulink.ModelAdvisor.getModelAdvisor('modelY')`.

New Simulink Blocks

This release introduces the following blocks:

- The Permute Dimensions block enables you to rearrange the dimensions of a multidimensional signal.
- The Squeeze block enables you to remove singleton dimensions from a multidimensional signal.

Change to Level-2 MATLAB S-Function Block

If a model includes a Level-2 MATLAB S-Function block, and an error occurs in the S-function, the Level-2 M-File S-Function block will display M-file stack trace information for the error in a dialog box. Click **OK** to remove the dialog box. In previous releases, this block did not display the stack trace information.

Model Dependency Analysis

The model dependencies manifest tools identify files required by your model. You can list required files in a 'manifest' file, package the model with required files into a ZIP file, or compare two file manifests.

See Model Dependencies for more information.

Model File Monitoring

- Warnings if a model file is changed on disk by another user or application while the model is loaded in Simulink software. (see Model File Change Notification in Managing Model Versions).
- Warning to notify the user if multiple models or libraries with the same name exist, as Simulink software may not be using the one the user expects. (see Shadowed Files).

Legacy Code Tool Enhancements

- New fields in the Legacy Code Tool data structure:
  `InitializeConditionsFcnSpec` and `SampleTime`.
  `InitializeConditionsFcnSpec` defines a function specification for a reentrant function that the S-function calls to initialize and reset states. `SampleTime` allows

you to specify whether sample time is inherited from the source block, represented as a tunable parameter, or fixed.

- Support for state (persistent memory) arguments in registered function specifications.

- Support for complex numbers specified for input, output, and parameter arguments in function specifications. This support is limited to use with Simulink built-in data types.

- Support for multidimensional arrays specified for input and output arguments in function specifications. Previously, multidimensional array support applied to parameters only.

- Ability to apply the size function to any dimension of function input data—input, output, parameter, or state. The data type of the `size` function's return value can be any type except complex, bus, or fixed-point.

- A new Legacy Code Tool option, `'backward_compatibility'`, which you can specify with the `legacy_code` function. This option enables Legacy Code Tool syntax, as made available from MATLAB Central in releases prior to R2006b, for a given MATLAB session.

- The following new demos:
  sldemo_lct_sampletime
  sldemo_lct_work
  sldemo_lct_cplxgain
  sldemo_lct_ndarray

For more information, see

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Writing S-Functions documentation

- Integrate External Code Using Legacy Code Tool in the Real-Time Workshop documentation

- legacy_code function reference page

Compatibility Considerations

If you are using a version of the Legacy Code Tool that was accessible from MATLAB Central before R2006b, the syntax for using the tool differs from the syntax currently supported by Simulink software. To continue using the old style syntax, for example, `legacy_code_initialize.m`, issue the following call to `legacy_code` for a given MATLAB session:

```
legacy_code('backward_compatibility');
```

Continuous State Names

State names can now be assigned in those blocks that employ continuous states. The names are assigned with the `ContinuousStateAttributes` Block-Specific Parameters parameter, or in the Blocks Parameter dialog box.

The following blocks support continuous state names:

- Integrator
- State-Space
- Transfer Fcn
- Variable Transport Delay
- Zero-Pole

Logging of continuous states is illustrated in the sldemo_hydrod demo.

Changes to Embedded MATLAB Function Block

This release introduces the following changes to the Embedded MATLAB Function block:

- "New Function Checks M-Code for Compliance with Embedded MATLAB Subset" on page 26-11
- "Support for Multidimensional Arrays" on page 26-12
- "Support for Function Handles" on page 26-12
- "Enhanced Support for Frames" on page 26-12
- "New Embedded MATLAB Runtime Library Functions" on page 26-12
- "Using & and | Operators in Embedded MATLAB Function Blocks" on page 26-14
- "Calling get Function from Embedded MATLAB Function Blocks" on page 26-15
- "Documentation on Embedded MATLAB Subset has Moved" on page 26-15

New Function Checks M-Code for Compliance with Embedded MATLAB Subset

Embedded MATLAB function blocks introduce a new function, Embedded MATLAB MEX (emlmex), that checks M-code for compliance with the syntax and semantics of the Embedded MATLAB subset. You can add Embedded MATLAB-compliant code to

Embedded MATLAB Function blocks and Truth Table blocks in Simulink models. For more information, see "Working with Embedded MATLAB MEX" in the Embedded MATLAB documentation.

Support for Multidimensional Arrays

Embedded MATLAB Function blocks now support multidimensional signals and parameter data, where the number of dimensions can be greater than 2. This feature is fully integrated with support for multidimensional signals in Simulink software. Supported functions in the Embedded MATLAB Run-Time Function Library have been enhanced to handle multidimensional data.

Support for Function Handles

Embedded MATLAB Function blocks now support function handles for invoking functions indirectly and parameterizing operations that you repeat frequently in your code. For more information, see the section on using function handles in About Code Generation from MATLAB Algorithms in the Embedded MATLAB documentation.

Enhanced Support for Frames

Embedded MATLAB Function blocks can now input and output frame-based signals directly in Simulink models. You no longer need to attach Frame Conversion blocks to inputs and outputs to achieve this functionality. See Working with Frame-Based Signals in the Simulink documentation.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide 31 new runtime library functions in the following categories:

- "Casting Functions" on page 26-13
- "Derivative and Integral Functions" on page 26-13
- "Discrete Math Functions" on page 26-13
- "Exponential Functions" on page 26-13
- "Filtering and Convolution Functions" on page 26-13
- "Logical Operator Functions" on page 26-13
- "Matrix and Array Functions" on page 26-13
- "Polynomial Functions" on page 26-14

See the Embedded MATLAB Run-Time Function Library for a list of all supported functions.

Casting Functions

- typecast

Derivative and Integral Functions

- cumtrapz
- trapz

Discrete Math Functions

- nchoosek

Exponential Functions

- expm

Filtering and Convolution Functions

- conv2
- deconv
- detrend
- filter2

Logical Operator Functions

- xor

Matrix and Array Functions

- cat
- flipdim
- normest
- rcond

- sortrows

Polynomial Functions

- poly

Set Functions

- issorted

Specialized Math

- beta
- betainc
- betaln
- ellipke
- erf
- erfc
- erfcinv
- erfcx
- erfinv
- expint
- gamma
- gammainc
- gammaln

Statistical Functions

- mode

Using & and | Operators in Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks no longer support & and | operators in `if` and `while` conditional statements.

Compatibility Considerations

In prior releases, these operators compiled without error, but their short-circuiting behavior was not implemented correctly. Substitute && and || operators instead.

Calling get Function from Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now support the Simulink Fixed Point get function for returning the properties of `fi` objects.

Compatibility Considerations

To get properties of *non-fixed-point* objects in Embedded MATLAB Function blocks, you must first declare `get` to be an extrinsic function; otherwise, your code will error. For more information refer to Calling MATLAB Functions in the Embedded MATLAB documentation.

Documentation on Embedded MATLAB Subset has Moved

Documentation on the Embedded MATLAB subset and its syntax, semantics, and supported functions has moved out of the Simulink Reference. See Code Generation from MATLAB User's Guide for the new Embedded MATLAB documentation.

Referenced Models Support Non-Zero Start Time

The simulation start time of all models in a model reference hierarchy was previously required to be 0. Now the simulation start time can be nonzero. The start time of all models in a model reference hierarchy must be the same. See Referencing a Model and Specifying a Simulation Start and Stop Time for information about these capabilities. See "Referencing Configuration Sets" on page 26-18 for information about a convenient way to give all models in a hierarchy the same configuration parameters, including simulation start time.

New Functions Copy a Model to a Subsystem or Subsystem to Model

Two new functions exist that you can use to copy contents between a block diagram and a subsystem.

Simulink.BlockDiagram.copyContentsToSubSystem

Copies the contents of a block diagram to an empty subsystem.

Simulink.SubSystem.copyContentsToBlockDiagram

Copies the contents of a subsystem to an empty block diagram.

For details, see the reference documentation for each function.

New Functions Empty a Model or Subsystem

Two new functions exist that you can use to delete the contents of a block diagram or subsystem.

Simulink.BlockDiagram.deleteContents

  Deletes the contents of a block diagram.

Simulink.SubSystem.deleteContents

  Deletes the contents of a subsystem.

For details, see the reference documentation for each function.

Default for Signal Resolution Parameter Has Changed

In the Configuration Parameters dialog, **Diagnostics > Data Validity** pane, the default setting for **Signal resolution** is now **Explicit only**. Previously, the default was **Explicit and warn implicit**. Equivalently, the default value of the `SignalResolutionControl` parameter is now `UseLocalSettings` (previously `TryResolveAllWithWarnings`). See Diagnostics Pane: Data Validity for more information.

Compatibility Considerations

Due to this change, labeling a signal is no longer enough to cause it to resolve by default to a signal object. You must also do one of the following:

- In the signal's Signal Properties dialog, select **Signal name must resolve to Simulink data object** and specify a Simulink.Signal object in the **Signal name** field. Simulink software then resolves that signal to that signal object.

- In the Configuration Parameters dialog, set **Diagnostics > Data Validity > Signal resolution** to `Explicit and warn implicit` (to post warnings) or `Explicit and implicit` (to suppress warnings). Simulink software then resolves all labeled signals to signal objects by matching their names, posting a warning of each resolution if so directed.

Models built in R2007a will default to **Explicit only**. Models created in previous versions will retain the **Signal resolution** value with which they were saved, and will run as they did before. New models may therefore behave differently from existing models that retain the previous default behavior. To specify the previous default behavior in a new model, change **Signal resolution** to **Explicit and warn implicit**.

Conversion Function

MathWorks discourages using implicit signal resolution except for fast prototyping, because implicit resolution slows performance, complicates model validation, and can have nondeterministic effects. Simulink software provides the disableimplicitsignalresolution function, which you can use to change settings throughout a model so that it does not use implicit signal resolution. See the function's reference documentation, or type:

```
help disableimplicitsignalresolution
```

in the MATLAB Command Window.

Referencing Configuration Sets

This release provides *configuration references* (`Simulink.ConfigSetRef` class), which you can use to link multiple models to a configuration set stored on the base workspace. All of those models then share the same configuration set, and therefore have the same configuration parameter values. Changing a parameter value in a shared configuration set changes that value for every model that uses the set. With configuration references, you can:

- Assign the same configuration set to any number of models
- Replace the configuration sets of any number of models without changing the model files
- Use different configuration sets for a referenced model in different contexts without changing the model file

See Manage a Configuration Set and Manage a Configuration Reference for more information.

Compatibility Considerations

You cannot change configuration parameter values by operating directly on a configuration reference as you can a configuration set. Instead, you must use the configuration reference to retrieve the configuration set and operate on the set. If you reconfigure a model to access configuration parameters using a configuration reference, you must update any scripts that change parameter values to incorporate the extra step of obtaining the configuration set from the reference before changing the values. See Create a Freestanding Configuration Set at the Command Line for details.

New Block, Model Advisor Check, and Utility Function for Bus to Vector Conversion

When the diagnostic **Configuration Parameters > Connectivity > Buses > Bus signal treated as vector** is disabled or **none**, you can input a homogeneous virtual bus to many blocks that accept vectors but are not formally defined as accepting buses. Simulink software transparently converts the bus to a vector, allowing the block to accept the bus.

However, MathWorks discourages intermixing buses and vectors, because such mixtures cause ambiguities that preclude strong type checking. The practice may become unsupported at some future time, and should not be used in new applications.

Simulink software provides diagnostics that report cases where buses are mixed with vectors, and includes capabilities that you can use to upgrade a model to eliminate such mixtures, as described in the following sections of the Simulink documentation:

- Using Composite Signals — A new chapter in R2007a that describes the specification and use of composite signals.

- Avoiding Mux/Bus Mixtures — Ambiguities that arise when composite signal types are intermixed, and the tools available for eliminating such mixtures.

- Using Diagnostics for Mux/Bus Mixtures — Two diagnostic options for detecting mixed composite signals: Mux blocks used to create bus signals and Bus signal treated as vector.

- Using the Model Advisor for Mux/Bus Mixtures — Model Advisor checks that detect mixed composite signals and recommend alternatives.

- Bus to Vector — A block that you can insert into a bus used implicitly as a vector to explicitly convert the bus to a vector.

- Simulink.BlockDiagram.addBusToVector — A function that creates a report of every bus used implicitly as a vector, and optionally inserts a Bus to Vector block into every such bus, replacing the implicit use with an explicit conversion.

Enhanced Support for Tunable Parameters in Expressions

Expressions that index into tunable parameters, such as `P(1)+P(2)/P(i)`, retain their tunability in generated code, including simulation code that is generated for a referenced model. Both the indexed parameter and the index itself can be tuned.

Parameter expressions of the form P(i) retain their tunability if all of the following are true:

- The index `i` is a constant or variable of `double` datatype
- `P` is a 1D array, or a 2D array with one row or one column, of `double` datatype
- `P` does not resolve to a mask parameter, but to a variable in the model or the base workspace

New Loss of Tunability Diagnostic

Previously, any loss of tunability generated a warning. In R2007a, you can use the **Loss of Tunability** diagnostic to control whether loss of tunability is ignored or generates a warning or error. See Detect loss of tunability for details.

Port Parameter Evaluation Has Changed

Previously, resolution of port parameters of a masked subsystem began within the subsystem, which could violate the integrity of the mask. For example, if a subsystem mask defines parameter A, and a port of the subsystem uses A to set some port attribute, resolving A by starting within the masked block makes A externally visible, though it should be visible only within the mask.

To fix this problem, in R2007a masked subsystem port parameter resolution starts in the containing system rather than within the masked subsystem, then proceeds hierarchically upward as it did before. This change preserves the integrity of the masked subsystem, but can change model behavior if any subsystem port previously depended for resolution on a variable defined within the mask.

Compatibility Considerations

A model whose ports did not reference variables defined within a mask are unaffected. A model that resolved any port parameter by accessing a variable within a masked block may behave differently or become vulnerable to future changes in behavior, as follows:

- If the port parameter's value cannot be evaluated, because the evaluation would require access to a variable defined only within the mask, an error occurs.
- If an appropriate variable exists outside the mask but has a different value than the corresponding variable within the mask, no error occurs, but model behavior may change.
- If an appropriate variable exists and has the same value inside and outside the mask, no behavioral change occurs, but later changes to the variable outside the mask may have unexpected effects.

To ensure correct results, change the model as needed so that any port parameter that previously depended on any variables defined within a mask give the intended results using the new resolution search path.

Data Type Objects Can Be Passed Via Mask Parameters

Previously, if a masked subsystem contained a block that needed to specify a data type using a data type object, the block could access the object only in the base workspace. The data type object could *not* be passed into the subsystem through a mask parameter. Parameterizing data types used by blocks under a mask was therefore not possible.

To support parameterized data types inside masked subsystems, you can now use a mask parameter to pass a data type object into a subsystem. Blocks in the subsystem can then use the object to specify data types under the mask.

Expanded Options for Displaying Subsystem Port Labels

This release provides an expanded set of options for displaying port labels on a subsystem block. The options include displaying:

• The label on the corresponding port block
• The name of the corresponding port block
• The name of the signal connected to the corresponding block

See the documentation for the **Show Port Labels** option on the Subsystem block's parameter dialog box for more information.

Model Explorer Customization Option Displays Properties of Selected Object

This release introduces a `Selection Properties` node to the Model Explorer's **Customize Contents** pane. The node allows you to customize the Model Explorer's **Contents** pane to display only the properties of the currently selected object. See The Model Explorer: Overview for more information.

Change to PaperPositionMode Parameter

In this release, when exporting a diagram as a graphic with the PaperPositionMode model parameter set to `auto`, Simulink software sizes the exported graphic to be the same size as the diagram's image on the screen when viewed at normal size. When PaperPositionMode is set to `manual`, Simulink software sizes the exported image to have the height and width specified by the model's PaperPosition parameter.

Compatibility Considerations

In previous releases, a model's PaperPosition parameter determined the size of the exported graphic regardless of the setting of the model's PaperPositionMode parameter.

To reproduce the behavior of previous releases, set the PaperPositionMode parameter to `manual`.

New Simulink.Bus.objectToCell Function

A new function, Simulink.Bus.objectToCell, is available for converting bus objects to a cell array that contains bus information. For details, see the description of Simulink.Bus.objectToCell.

Simulink.Bus.save Function Enhanced To Allow Suppression of Bus Object Creation

The Simulink.Bus.save function has been enhanced such that when using the `'cell'` format you have the option of suppressing the creation of bus objects when the saved M-file executes. To suppress bus object creation, specify the optional argument `'false'` when you execute the saved M-file.

For more detail, see the description of Simulink.Bus.save.

Change in Version 6.5 (R2006b) Introduced Incompatibility

A change introduced in Version 6.5 (R2006b) introduces an incompatibility between this release and releases preceding Version 6.5 (R2006b). See "Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error" on page 27-6 for more information.

Nonverbose Output During Code Generation

Simulink Accelerator now defaults to nonverbose output when generating code. A new parameter, `AccelVerboseBuild`, has been added to control how much information is displayed. See Customizing the Build Process for more information.

SimulationMode Removed From Configuration Set

Previously, the `SimulationMode` property was attached to the configuration set for a model. In R2007a, the property has been removed from the configuration set. Now you set the simulation mode for the model using the **Simulation** menu in the model window or the set_param function with the `SimulationMode` model parameter.

Compatibility Considerations

Using `Simulink.ConfigSet.SimulationMode` is not recommended. Use `set_param(`*`modelName`*`,'SimulationMode',`*`value`*`)` instead.

R2006b

Version: 6.5

New Features

Bug Fixes

Compatibility Considerations

Model Dependency Viewer

The Model Dependency Viewer displays a dependency view of a model that shows models and block libraries directly or indirectly referenced by the model. The dependency view allows you to quickly determine your model's dependencies on referenced models and block libraries. See Model Dependencies for more information.

Enhanced Lookup Table Blocks

This release replaces the PreLookup Index Search and Interpolation (n-D) Using PreLookup blocks with two new blocks: Prelookup and Interpolation Using Prelookup. The new blocks provide fixed-point arithmetic, consistency checking, more efficient code generation, and other enhancements over the blocks they replace.

Compatibility Considerations

MathWorks plans on obsoleting the PreLookup Index Search and Interpolation (n-D) Using PreLookup blocks in a future release. In the meantime, MathWorks will continue to support and enhance these blocks. For example, this release improves the precision with which the PreLookup Index Search block computes its fraction value if its **Index search method** parameter specifies `Evenly Spaced Points`.

We recommend that you use the Prelookup and Interpolation Using Prelookup blocks for all new model development.

Legacy Code Tool

The Legacy Code Tool generates an S-function from existing C code and specifications that you supply. It enables you to transform your C functions into C MEX S-functions for inclusion in a Simulink model. See Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in Developing S-Functions for more information.

Simulink Software Now Uses Internal MATLAB Functions for Math Operations

In previous releases, Simulink software used the host compiler's C++ Math Library functions to perform most mathematical operations on floating-point data. Some of those functions produced results that were slightly inconsistent with MATLAB results. In this release, Simulink software calls the same internal routines that MATLAB calls for most trigonometric, exponential, and rounding and remainder operations involving floating-

point data. This ensures that when Simulink and MATLAB products operate on the same platform, they produce the same numerical results.

In particular, Simulink software now performs mathematical operations with the same internal functions that MATLAB uses to implement the following M-functions:

- `sin`, `cos`, `tan`
- `asin`, `acos`, `atan`, `atan2`
- `sinh`, `cosh`, `tanh`
- `asinh`, `acosh`, `atanh`
- `log`, `log2`, `log10`
- `mod`, `rem`
- `power`

Note: By default, in this release Real-Time Workshop software continues to use C Math Library functions in the code that it generates from a Simulink model.

Enhanced Integer Support in Math Function Block

The `sqrt` operation in the Math Function block now supports built-in integer data types.

Configuration Set Updates

This release includes the following changes to model configuration parameters and configuration sets.

- This release includes a new command, openDialog, that displays the **Configuration Parameters** dialog box for a specified configuration set. This command allows display of configuration sets that are not attached to any model.
- The attachConfigSet command now includes an `allowRename` option that determines how the command handles naming conflicts when attaching a configuration set to a model.
- This release includes a new attachConfigSetCopy command that attaches a copy of a specified configuration set to a model.
- The new **Sample hit time adjusting** diagnostic controls whether Simulink software notifies you when the solver has to adjust a sample time specified by your model to solve the model. The associated model parameter is `TimeAdjustmentMsg`.

- The default value of the **Multitask data store** diagnostic has changed from `Warning` to `Error` for new models. This change does not affect existing models.
- The name of the **Block reduction optimization** parameter has changed to **Block reduction**.

Command to Initiate Data Logging During Simulation

The command

```
set_param(bdroot, 'SimulationCommand', 'WriteDataLogs')
```
writes all logging variables during simulation. See Exporting Signal Data Using Signal Logging for more information.

Commands for Obtaining Model and Subsystem Checksums

This release includes commands for obtaining model and subsystem checksums.

- Simulink.BlockDiagram.getChecksum

  Get checksum for a model. Simulink Accelerator software uses this checksum to control regeneration of simulation targets. You can use this command to diagnose target rebuild problems.

- Simulink.SubSystem.getChecksum

  Get checksum for a subsystem. Real-Time Workshop software uses this checksum to control reuse of code generated from a subsystem that occurs more than once in a model. You can use the checksum to diagnose code reuse problems. See Determine Why Subsystem Code Is Not Reused.

Sample Hit Time Adjusting Diagnostic

The **Sample hit time adjusting** diagnostic controls whether Simulink software notifies you when the solver has to adjust a sample time specified by your model to solve the model. The associated model parameter is `TimeAdjustmentMsg`.

Function-Call Models Can Now Run Without Being Referenced

This release allows you to simulate a function-call model, i.e., a model that contains a root-level function-call trigger block, without having to reference the model. In previous

releases, the function-call model had to be referenced by another model in order to be simulated.

Signal Builder Supports Printing of Signal Groups

This release adds printing options to the Signal Builder block's editor. It allows you to print waveforms displayed in the editor to a printer, file, the clipboard, or a figure window. For details, see Printing, Exporting, and Copying Waveforms.

Method for Comparing Simulink Data Objects

This release introduces an `isContentEqual` method for Simulink data objects that allows you to determine whether a Simulink data object has the same property values as another Simulink data object. For more information, see Comparing Data Objects.

Unified Font Preferences Dialog Box

In this release, the **Simulink Preferences** dialog box displays font settings for blocks, lines, and annotations on a single pane instead of on separate tabbed panes as in previous releases. This simplifies selection of font preferences.

Limitation on Number of Referenced Models Eliminated for Single References

In previous releases, all distinct models referenced in a model hierarchy counted against the limitation imposed by Microsoft Windows on the number of distinct referenced models that can occur in a hierarchy. In this release, models configured to be instantiable only once do not account against this limit. This means that a model hierarchy can reference any number of distinct models on Windows platforms as long as they are referenced only once and are configured to be instantiable only once (see Model Referencing Limitations for more information).

Parameter Objects Can Now Be Used to Specify Model Configuration Parameters

This release allows you to use Simulink.Parameter objects to specify model configuration as well as block parameters. For example, you can specify a model's fixed step size as `Ts`

and its stop time as `20*Ts` where `Ts` is a workspace variable that references a parameter object. When compiling a model, Simulink software replaces a reference to a parameter object in a model configuration parameter expression with the object's value.

Compatibility Considerations

In previous releases, you could use expressions of the form `p.Value()`, where `p` references a parameter object, in model configuration parameter expressions. Such expressions cause expression evaluation errors in this release when you compile a model. You should replace such expressions with a simple reference to the parameter object itself, i.e., replace `p.Value()` with `p`.

Parameter Pooling Is Now Always Enabled

In previous releases, the **Parameter Pooling** optimization was optional and was enabled by default. Due to internal improvements, disabling **Parameter Pooling** would no longer be useful in any context. The optimization is therefore part of standard R2006b operation, and has been removed from the user interface.

Compatibility Considerations

Upgrading a model to R2006b inherently provides the effect that enabling **Parameter Pooling** did in previous releases. No compatibility considerations result from this change. If the optimization was disabled in an existing model, a warning is generated when the model is first upgraded to R2006b. This warning requires no action and can be ignored.

Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error

In this release, attempting to reference an symbol in an uninitialized mask workspace generates an error. This can happen, for example, if a masked subsystem's initialization code attempts to set a parameter of a block that resides in a masked subsystem in the subsystem being initialized and one or more of the block's parameters reference variables defined by the mask of the subsystem in which it resides (see Initialization Command Limitations for more information).

Compatibility Considerations

In this release, updating or simulating models created in previous releases may generate unresolvable symbol error messages. This can happen if the model contains masked subsystems whose initialization code sets parameters on blocks residing in lower-level masked subsystems residing in the top-level masked subsystem. To eliminate these errors, change the initialization code to avoid the use of `set_param` commands to set parameters in lower-level masked subsystems. Instead, use mask variables in upper-level masked subsystems to specify the values of parameters of blocks residing in lower-level masked subsystems. See Defining Mask Parameters for information on using mask variables to specify block parameter values.

Changes to Integrator Block's Level Reset Options

This release changes the behavior of the `level` reset option of the Integrator block. In releases before Simulink 6.3, the `level` reset option resets the integrator's state if the reset signal is nonzero or changes from nonzero in the previous time step to zero in the current time step. In Simulink 6.3, 6.4, and 6.4.1, the option resets the integrator only if the reset signal is nonzero. This release restores the `level` reset behavior of releases that preceded Simulink 6.3. It also adds a `level hold` option that behaves like the `level` reset option of Simulink 6.3, 6.4, and 6.4.1.

Compatibility Considerations

A model that uses the `level` reset option could produce results that differ in this release from those produced in Simulink 6.3, 6.4, and 6.4.1. To reproduce the results of previous releases, change the model to use the new `level hold` option instead.

Embedded MATLAB Function Block Features and Changes

Support for Structures

You can now define structures as inputs, outputs, local, and persistent variables in Embedded MATLAB Function blocks. With support for structures, Embedded MATLAB Function blocks give you the ability to read and write Simulink bus signals at inputs and outputs of Embedded MATLAB Function blocks. See "Using Structures" in the Embedded MATLAB documentation.

Embedded MATLAB Editor Analyzes Code with M-Lint

The Embedded MATLAB Editor uses the MATLAB M-Lint Code Analyzer to automatically check your Embedded MATLAB function code for errors and recommend corrections. The editor displays an M-Lint bar that highlights offending lines of code and displays Embedded MATLAB diagnostics as well as MATLAB messages. See "Using M-Lint with Embedded MATLAB" in the Embedded MATLAB documentation.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide 36 new runtime library functions in the following categories:

- "Data Analysis" on page 27-8
- "Discrete Math" on page 27-8
- "Exponential" on page 27-8
- "Interpolation and Computational Geometry" on page 27-9
- "Linear Algebra" on page 27-9
- "Logical" on page 27-9
- "Specialized Plotting" on page 27-10
- "Transforms" on page 27-10
- "Trigonometric" on page 27-10

Data Analysis

- cov
- ifftshift
- std
- var

Discrete Math

- gcd
- lcm

Exponential

- expm1

- log10
- log1p
- log2
- nextpow2
- nthroot
- reallog
- realpow
- realsqrt

Interpolation and Computational Geometry

- cart2pol
- cart2sph
- pol2cart
- sph2cart

Linear Algebra

- cond
- det
- ipermute
- kron
- permute
- planerot
- rand
- randn
- rank
- shiftdim
- squeeze
- subspace
- trace

Logical

- isstruct

Specialized Plotting

- histc

Transforms

- bitrevorder

Trigonometric

- hypot

New Requirement for Calling MATLAB Functions from Embedded MATLAB Function Blocks

To call external MATLAB functions from Embedded MATLAB Function blocks, you must first declare the functions to be extrinsic. (External MATLAB functions are functions that have not been implemented in the Embedded MATLAB runtime library.) MATLAB Function blocks do not compile or generate code for extrinsic functions; instead, they send the function to MATLAB for execution during simulation. There are two ways to call MATLAB functions as extrinsic functions in Embedded MATLAB Function blocks:

- Use the new construct `eml.extrinsic` to declare the function extrinsic
- Call the function using `feval`

For details, see Calling MATLAB Functions in the Embedded MATLAB documentation.

Compatibility Considerations

Currently, Embedded MATLAB Function blocks use implicit rules to handle calls to external functions:

- For simulation, Embedded MATLAB Function blocks send the function to MATLAB for execution
- For code generation, Embedded MATLAB Function blocks check whether the function affects the output of the Embedded MATLAB function in which it is called. If there is no effect on output, Embedded MATLAB Function blocks proceed with code generation, but exclude the function call from the generated code. Otherwise, Embedded MATLAB Function blocks generate a compiler error.

In future releases, Embedded MATLAB Function blocks will apply these rules only to external functions that you call as extrinsic functions. Otherwise, they will compile external functions by default, potentially causing unpredictable behavior or generating

errors. For reliable simulation and code generation, MathWorks recommends that you call external MATLAB functions as extrinsic functions.

Type and Size Mismatch of Values Returned from MATLAB Functions Generates Error

Embedded MATLAB Function blocks now generate an error if the type and size of a value returned by a MATLAB function does not match the predeclared type and size.

Compatibility Considerations

In previous releases, Embedded MATLAB Function blocks attempted to silently convert values returned by MATLAB functions to predeclared data type and sizes if a mismatch occurred. Now, such mismatches always generate an error, as in this example:

```
x = int8(zeros(3,3)); % Predeclaration
x = eval('5'); % Calls MATLAB function eval
```

This code now generates an error because the Embedded MATLAB function predeclares x as a 3–by-3 matrix, but MATLAB function returns x as a scalar double. To avoid errors, reconcile predeclared data types and sizes with the actual types and sizes returned by MATLAB function calls in your Embedded MATLAB Function blocks.

Embedded MATLAB Function Blocks Cannot Output Character Data

Embedded MATLAB Function blocks now generate an error if any of its outputs is character data.

Compatibility Considerations

In the previous release, Embedded MATLAB Function blocks silently cast character array outputs to int8 scalar arrays. This behavior does not match MATLAB, which represents characters in 16–bit unicode.

R2006a+

Version: 6.4.1

No New Features or Changes

R2006a

Version: 6.4

New Features

Compatibility Considerations

Signal Object Initialization

This release introduces the use of signal objects to specify initial values for signals and states. This allows you to initialize signals or states in the model, not just those generated by blocks that have initial condition or value parameters. For details, see Using Signal Objects to Initialize Signals and Discrete States in the online Simulink documentation.

Icon Shape Property for Logical Operator Block

The Logical Operator block's parameter dialog box contains a new property, **Icon shape**, settings for which can be either `rectangular` or `distinctive`. If you select `rectangular` (the default), the block appears as it does in previous releases. If you select `distinctive`, the block appears as the IEEE® standard graphic symbol for the selected logic operator.

Data Type Property of Parameter Objects Now Settable

This release allows you to set the data type of a Simulink.Parameter object via either its `Value` property or via its `Data type` property. In previous releases, you could specify the data type of a parameter object only by setting the object's `Value` property to a typed value expression.

Range-Checking for Parameter and Signal Object Values

This release introduces range checking for Simulink.Parameter and Simulink.Signal objects. Simulink software checks whether a parameter's **Value** or a signal's **Initial value** falls within the values you specify for the object's **Minimum** and **Maximum** properties. If not, Simulink software generates a warning or error.

Compatibility Considerations

Previous releases ignored such violations since the `Minimum` and `Maximum` properties were intended for use in documenting parameter and signal objects. In this release, Simulink software displays a warning if you load a parameter object or a signal object does not specify a valid range or its value falls outside the specified range. If you get such a warning, change the parameter or signal object's `Value` or `Minimum` or `Maximum` values so that the `Value` falls within a valid range.

Expanded Menu Customization

The previous release of Simulink software allows you to customize the Simulink editor's **Tools** menu. This release goes a step further and allows you to customize any Simulink (or Stateflow) editor menu (see Customizing the Simulink User Interface in the online Simulink documentation).

Bringing the MATLAB Desktop Forward

The Model Editor's **View** menu includes a new command, **MATLAB Desktop**, that brings the MATLAB desktop to the front of the windows displayed on your screen.

Converting Atomic Subsystems to Model References

This release adds a command, **Convert to Model Block**, to the context (right-click) menu of an atomic subsystem. Selecting this command converts an atomic subsystem to a model reference . See Atomic Subsystem and Converting a Subsystem to a Referenced Model for more information.

The function `sl_convert_to_model_reference`, which provided some of the same capabilities as **Convert to Model Block**, is obsolete and has been removed from the documentation. The function continues to work, so no incompatibility arises, but it posts a warning when called. The function will be removed in a future release.

Concatenate Block

The new Concatenate block concatenates its input signals to create a single output signal whose elements occupy contiguous locations in memory. The block typically uses less memory than the Matrix Concatenation block that it replaces, thereby reducing model memory requirements.

Compatibility Considerations

This release replaces obsolete Matrix Concatenation blocks with Concatenate blocks when loading models created in previous releases.

Model Advisor Changes

Model Advisor Tasks Introduced

This release introduces Model Advisor tasks for referencing models and upgrading a model to the current version of Simulink software. See Consulting the Model Advisor in the online Simulink documentation for more information.

Model Advisor API

This release introduces an application program interface (API) that enables you to run the Model Advisor from the MATLAB command line or from M-file programs. For example, you can use the API to create M-file programs that determine whether a model passes selected Model Advisor checks whenever you open, simulate, or generate code from the model. See Running the Model Advisor Programmatically in the online Simulink documentation for more information.

Built-in Block's Initial Appearance Reflects Parameter Settings

In this release, when you load a model containing nonmasked, built-in blocks whose appearance depends on their parameter settings, such as the Selector block, the appearance of the blocks reflect their parameter settings. You no longer have to update the model to update the appearance of such blocks.

Compatibility Considerations

In previous releases, model or block callback functions that use set_param to set a built-in, nonmasked block's parameters could silently put the block in an unusable state. In this release, such callbacks will trigger error messages if they put blocks in an unusable state.

Double-Click Model Block to Open Referenced Model

In this release, double-clicking a Model block that specifies a valid referenced model opens the referenced model, rather than the Block Parameters dialog box as in previous releases. To open the Block Parameters dialog box, choose **Model Reference Parameters** from the **Context** or **Edit** menu. See Navigating a Model Block for details.

Signal Logs Reflect Bus Hierarchy

In this release, signal logs containing buses reflect the structure of the buses themselves instead of flattening bus data as in previous releases (see Simulink.TsArray).

Tiled Printing

This release introduces a tiled printing option that allows you to distribute a block diagram over multiple pages. You can control the number of pages over which Simulink software distributes the block diagram, and hence, the total size of the printed image. See Tiled Printing in the online Simulink documentation for more information.

Solver Diagnostic Controls

In this release, the **Configuration Parameters** dialog box includes the following enhancements:

- The **Diagnostics** pane contains a new diagnostic, **Consecutive zero crossings violation**, that alerts you if Simulink software detects the maximum number of consecutive zero crossings allowed. You can specify the criteria that Simulink software uses to trigger this diagnostic using two new **Solver diagnostic controls** on the **Solver** pane:

  - **Consecutive zero crossings relative tolerance**
  - **Number of consecutive zero crossings allowed**

  For more information, see Preventing Excessive Zero Crossings in the online Simulink documentation.

- The **Solver** pane contains a new solver diagnostic control, **Number of consecutive min step size violations allowed**, that Simulink software uses to trigger the **Min step size violation** diagnostic (see Number of consecutive min steps in the online Simulink documentation).

Diagnostic Added for Multitasking Conditionally Executed Subsystems

This release adds a sample-time diagnostic that detects an enabled subsystem in multitasking solver mode that operates at multiple rates or a conditionally executed subsystem that contain an asynchronous subsystem. Such subsystems can cause

corrupted data or non-deterministic behavior in a real-time system using code generated from the model. See the documentation for the **Multitask Conditionally Executed Subsystem** diagnostic for more information.

Embedded MATLAB Function Block Features and Changes

Option to Disable Saturation on Integer Overflow

The properties dialog for Embedded MATLAB Function blocks provides a new **Saturate on Integer Overflow** check box that lets you disable saturation on integer overflow to generate more efficient code. When you enable saturation on integer overflow, Embedded MATLAB Function blocks add additional checks in the generated code to detect integer overflow or underflow. Therefore, it is more efficient to disable this option if your algorithm does not rely on overflow behavior. For more information, see MATLAB Function Block Properties in the online Simulink documentation.

Nontunable Option Allows Use of Parameters in Constant Expressions

The **Data** properties dialog for the MATLAB Function (formally called Embedded MATLAB Function) block provides a new **Tunable** check box that lets you specify the tunability (see Tunable Parameters in the online Simulink documentation) of a workspace variable or mask parameter used as data in Embedded MATLAB code. The option is checked by default. Unchecking the option allows you to use a workspace variable or mask parameter as data wherever Embedded MATLAB requires a constant expression, such as a dimension argument to the zeros function. For more information, see Adding Data to a MATLAB Function Block in the online Simulink documentation.

Enhanced Support for Fixed-Point Arithmetic

Embedded MATLAB Function blocks support the new fixed-point features introduced in Version 1.4 (R2006a) of the Fixed-Point Toolbox software, including [Slope Bias] scaling (see Specifying Simulink Fixed Point Data Properties in the online Simulink documentation). For information about the features added to the Simulink Fixed Point software, see Fixed-Point Toolbox Release Notes.

Support for Integer Division

Embedded MATLAB Function blocks support the new MATLAB function idivide, which performs integer division with a variety of rounding options. It is recommended that the rounding option used for integer division in Embedded MATLAB Function blocks match the rounding option in the parent Simulink model.

The default rounding option for `idivide` is `'fix'`, which rounds toward zero. This option corresponds to the choice **Zero** in the submenu for **Signed integer division rounds to:**, a parameter that you can set in the Hardware Implementation Pane of the Configuration Parameters dialog in Simulink software (see Hardware Implementation Pane in the online Simulink documentation). If this parameter is set to **Floor** in the Simulink model that contains the Embedded MATLAB Function block, it is recommended that you pass the rounding option `'floor'` to `idivide` in the block.

For a complete list of Embedded MATLAB runtime library functions provided in this release, see "New Embedded MATLAB Runtime Library Functions" on page 29-7.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide new runtime library functions in the following categories:

- "Integer Arithmetic" on page 29-7
- "Linear Algebra" on page 29-7
- "Logical" on page 29-8
- "Polynomial" on page 29-8
- "Trigonometric" on page 29-8

Integer Arithmetic

- idivide

Linear Algebra

- compan
- dot
- eig
- fliplr
- flipud
- freqspace
- hilb
- ind2sub
- invhilb
- linspace

- logspace
- magic
- median
- meshgrid
- pascal
- qr
- rot90
- sub2ind
- toeplitz
- vander
- wilkinson

Logical

- isequal
- isinteger
- islogical

Polynomial

- polyfit
- polyval

Trigonometric

- acosd
- acot
- acotd
- acoth
- acsc
- acscd
- acsch
- asec
- asecd

- asech
- asind
- atand
- cosd
- cot
- cotd
- coth
- csc
- cscd
- csch
- sec
- secd
- sech
- sind
- tand

Setting FIMATH Cast Before Sum to False No Longer Supported in Embedded MATLAB Function Blocks

You can no longer set the FIMATH property CastBeforeSum to false for fixed-point data in Embedded MATLAB Function blocks.

Compatibility Considerations

The reason for the restriction is that Embedded MATLAB Function blocks do not produce the same numerical results as MATLAB when CastBeforeSum is false. In the previous release, Embedded MATLAB Function blocks set CastBeforeSum to false by default for the default FIMATH object. If you have existing models that contain Embedded MATLAB Function blocks in which CastBeforeSum is false, you will get an error when you compile or update your model. To correct the issue, you must set CastBeforeSum to true. To automate this process, you can run the utility slupdate either from the Model Advisor or by typing the following command at the MATLAB command line:

```
slupdate ('modelname')
```

where '*modelname*' is the name of the model containing the Embedded MATLAB Function block that generates the error. `slupdate` prompts you to update this property by selecting one of these options:

Option	Action
Yes	Updates the first occurrence of `CastBeforeSum=false` in Embedded MATLAB Function blocks in the offending model and then prompts you for each subsequent one found in the model.
No	Does not update any occurrences of `CastBeforeSum=false` in the offending model.
All	Updates all occurrences of `CastBeforeSum=false` in the offending model.

Note `slupdate` detects `CastBeforeSum=false` only in *default* FIMATH objects defined for Simulink software signals in Embedded MATLAB Function blocks. If you modified the FIMATH object in an Embedded MATLAB Function block, update `CastBeforeSum` manually in your model and fix the errors as they are reported.

Type Mismatch of Scalar Output Data in Embedded MATLAB Function Blocks Generates Error

Embedded MATLAB Function blocks now generate an error if the output type inferred by the block does not match the type you explicitly set for a scalar output.

Compatibility Considerations

In previous releases, a silent cast was inserted from the computed type to the set type when mismatches occurred. In most cases, you should not need to set the output type for Embedded MATLAB Function blocks. When you do, insert an explicit cast in your Embedded MATLAB script. For example, suppose you declare a scalar output *y* to be of type int8, but its actual type is double. Replace *y* with a temporary variable *t* in your script and then add the following code:

```
y = int8(t);
```

Implicit Parameter Type Conversions No Longer Supported in Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now generate an error if the type of a parameter inferred by the block does not match the type you explicitly set for the parameter.

Compatibility Considerations

In the previous release, if the type you set for a parameter did not match the actual parameter value, Embedded MATLAB Function blocks implicitly cast the parameter to the specified type. Now you receive a compile-time error when type mismatches occur for parameters defined in Embedded MATLAB Function blocks.

There are two workarounds:

- Change the scope of the data from **Parameter** to **Input**. Then, connect to the input port a **Constant** block that brings in the parameter and casts it to the desired type.
- Cast the parameter inside your Embedded MATLAB function to the desired type.

Fixed-Point Parameters Not Supported

Embedded MATLAB Function blocks generate a compile-time error if you try to bring a `fi` object defined in the base workspace into Embedded MATLAB Function blocks as a parameter.

There are two workarounds:

- Change the scope of the data from **Parameter** to **Input**. Then, connect to the input port a **Constant** block that brings in the parameter and casts it to fixed-point type.
- Cast the parameter inside your Embedded MATLAB function to fixed-point type.

Embedded MATLAB Function Blocks Require C Compiler for Windows 64

No C compiler ships with MATLAB and Simulink products on Windows 64. Because Embedded MATLAB Function blocks perform simulation through code generation, you must supply your own MEX-supported C compiler to use these blocks. The C compilers available at the time of this writing for Windows 64 include Microsoft Visual Studio® 2005 and the Microsoft Platform SDK.

R14SP3

Version: 6.3

New Features

Compatibility Considerations

Model Referencing

Function-Call Models

This release allows you to use a block capable of emitting a function-call signal, such as a Function-Call Generator or a custom S-function, in one model to control execution of another model during the current time step. See Function-Call Subsystems in the Simulink documentation for more information.

Using Noninlined S-Functions in Referenced Models

This release adds limited support for use of noninlined S-functions in models referenced by other models. For example, you can simulate a model that references models containing noninlined S-functions. However, you cannot use Real-Time Workshop software to generate a standalone executable (Real-Time Workshop target) for the model. See Model Referencing Limitations in the Simulink documentation for information on other limitations.

Referenced Models Without Root I/O Can Inherit Sample Times

Previous releases of Simulink software do not allow referenced models without root-level input or output ports to inherit their sample time. This release removes this restriction.

Referenced Models Can Use Variable Step Solvers

Previous releases of Simulink software do not allow models to reference models that require variable-step solvers. This release removes this restriction.

Model Dependency Graphs Accessible from the Tools Menu

This release adds a **Model Reference Dependency Graph** item to the Model Editor's **Tools** menu. The item displays a graph of the models referenced by the model displayed in the Model Editor. You can open any model in the dependency graph by clicking its node. See Viewing a Model Reference Hierarchy in the Simulinkdocumentation for more information.

Command That Converts Atomic Subsystems to Model References

This release introduces a MATLAB command that converts an atomic subsystem to a model reference. See Simulink.SubSystem.convertToModelReference in the Simulink Reference documentation for more information.

Model Reference Demos

This release has the following model reference demo changes:

- Model reference demo names are now prepended with `sldemo_`. For example, the demo `mdlref_basic.mdl` is now `sldemo_mdlref_basic.mdl`.
- You can no longer use the `mdlrefdemos` command from the MATLAB command prompt to access model reference demos. Instead, you can navigate to the Simulink demos tab either though the Help browser, or by typing `demos` at the command prompt, then navigating to the Simulink demos category and browsing the demos.

Block Enhancements

Variable Transport Delay, Variable Time Delay Blocks

This release replaces the Variable Transport Delay block of previous releases with two new blocks. The Variable Transport Delay block of previous releases implemented a variable time delay behavior, which is now implemented by the Variable Time Delay block introduced in this release. This release changes the behavior of the Variable Transport Delay block to model variable transport delay behavior, e.g., the behavior of a fluid flowing through a pipe.

Additional Reset Trigger for Discrete-Time Integrator Block

This release adds a `sampled level` trigger option for causing the Discrete-Time Integrator to reset. The new reset trigger is more efficient than the `level` reset option, but may introduce a discontinuity when integration resumes.

Note: In Simulink 6.2 and 6.2.1, the level reset option behaves like the sampled level option in this release. This release restores the level reset option to its original behavior.

Input Port Latching Enhancements

This release includes the following enhancements to the signal latching capabilities of the Inport block.

Label Clarified for Triggered Subsystem Latch Option

The dialog box for an Inport block contains a check box to latch the signal connected to the system via the port. This check box applies only to triggered subsystems and hence is enabled only when the Inport block resides in a triggered subsystem. In this release, the label for the check box that selects this option has changed from **Latch (buffer) input** to **Latch input by delaying outside signal**. This change is intended to make it clear what the option does, i.e., cause the subsystem to see the input signal's value at the previous time step when the subsystem executes at the current time step (equivalent to inserting a Memory block at the input outside the subsystem). The Inport block's icon displays <Lo> to indicate that this option is selected.

Latch Option Added for Function-Call Subsystems

This release adds a check box labeled **Latch input by copying inside signal** to the Inport block's dialog box. This option applies only to function-call subsystems and hence is enabled only if the Inport block resides in a function-call subsystem. Selecting this option causes Simulink software to copy the signal output by the block into a buffer before executing the contents of the subsystem and to use this copy as the block's output during execution of the subsystem. This ensures that the subsystem's inputs, including those generated by the subsystem's context, will not change during execution of the subsystem. The Inport block's icon displays <Li> to indicate that this option is selected.

Improved Function-Call Inputs Warning Label

In previous releases, the dialog box for a function-call subsystem contains a check box labeled **Warn if function-call inputs arise inside called context**. This release changes the label to **Warn if function-call inputs are context-specific**. This change is intended to indicate more clearly the warning's purpose, i.e., to alert you that some or all of the function-call inputs come from the function-call subsystem's context and hence could change while the function-call subsystem is executing.

Note: In this release, you can avoid this function-call inputs problem by selecting the **Latch input by copying inside signal** option on the subsystem's Inport blocks (see "Latch Option Added for Function-Call Subsystems" on page 30-5).

Parameter Object Expressions No Longer Supported in Dialog Boxes

Compatibility Considerations

Previous releases allow you to specify a Simulink.Parameter object as the value of a block parameter by entering an expression that returns a parameter object in the parameter's value field in the block's parameter dialog box. In this release, you must enter the name of a variable that references the object in the MATLAB or model workspace.

Modeling Enhancements

Annotations

This release introduces the following enhancements to model annotations:

- Annotation properties dialog box (see Annotations Properties Dialog Box in the Simulinkdocumentation)
- Annotation callback functions (see Annotation Callback Functions in the Simulinkdocumentation)
- Annotation application programming interface (see Annotations API in the Simulinkdocumentation)

Custom Signal Viewers and Generators

This release allows you to add custom signal viewers and generators so that you can manage them in the Signal & Scope Manager. See Visualizing and Comparing Simulation Results in the Simulink documentation for further details.

Model Explorer Search Option

This release adds an `Evaluate Property Values During Search` option to the Model Explorer. This option applies only for searches by property value. If enabled, the option causes the Model Explorer to evaluate the value of each property as a MATLAB expression and compare the result to the search value. If disabled (the default), the Model Explorer compares the unevaluated property value to the search value.

Using Signal Objects to Assign Signal Properties

Previous releases allow you to use signal objects to check signal property values assigned by signal sources. This release allows you, in addition, to use signal objects to assign values to properties not set by signal sources. See Simulink.Signal in the Simulink Reference documentation for more information.

Bus Utility Functions

This release introduces the following bus utility functions:

- Simulink.Bus.save
- Simulink.Bus.createObject
- Simulink.Bus.cellToObject

Fixed-Point Support in Embedded MATLAB Function Blocks

In this release, the Embedded MATLAB Function block supports manyFixed-Point Toolbox functions. This allows you to generate code from models that contain fixed-point MATLAB functions. For more information, see Code Acceleration and Code Generation from MATLAB for Fixed-Point Algorithms in the Fixed-Point Toolbox documentation.

Note: You must have a Simulink Fixed Point license to use this capability.

Embedded MATLAB Function Editor

The Embedded MATLAB Editor has a new tool, the Ports and Data Manager. This tool helps you manage your block inputs, outputs, and parameters. The Ports and Data Manager uses the same Model Explorer dialogs for manipulating data, but restricts the view to the block you are working on. You can still access the Model Explorer via a menu item to get the same functionality as in previous releases.

Input Trigger and Function-Call Output Support in Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now supports input triggers and function-call outputs. See Ports and Data Manager in the Simulinkdocumentation for more information.

Find Options Added to the Data Object Wizard

This release adds find options to the **Data Object Wizard**. The options enable you to restrict the search for model data to specific kinds of objects. See Data Object Wizard in the Simulinkdocumentation for more information.

Fixed-Point Functions No Longer Supported for Use in Signal Objects

Compatibility Considerations

Previous releases allowed you to use fixed-point data type functions, such as `sfix`, to specify the value of the `DataType` property of a `Simulink.Signal` object. This release allows you to use only built-in data types and Simulink.NumericType objects to specify the data types of `Simulink.Signal` objects. See the Simulink.Signal documentation for more information.

Simulation Enhancements

Viewing Logged Signal Data

This release can display logged signal data in the MATLAB **Times Series Tools** viewer on demand or whenever a simulation ends or you pause a simulation. See "Viewing Logged Signal Data" in the Simulink documentation for more information.

Importing Time-Series Data

In this release, root-level Inport blocks can import data from time-series (see Simulink.Timeseries in the Simulink Reference documentation) and time-series array (see Simulink.TSArray in the Simulink Reference documentation) objects residing in the MATLAB workspace. See Importing MATLAB timeseries Data in the Simulink documentation for more information. From Workspace blocks can also import time-series objects. The ability to import time-series objects allows you to use data logged from one simulation as input to another simulation.

Using a Variable-Step Solver with Rate Transition Blocks

Previous releases of Simulink software generate an error if you try to use a variable-step solver to solve a model that contains Rate Transition blocks. This release allows you to use variable-step as well as fixed-step solvers to simulate a model. Note that you cannot generate code from a model that uses a variable-step solver. However, you may find it advantageous, in some cases, to use a variable-step solver to test aspects of the model not directly related to code generation. This enhancement allows you to switch back and forth between the two types of solver without having to remove and reinsert Rate Transition blocks.

Additional Diagnostics

This release adds the following simulation diagnostics:

- Enforce sample times specified by Signal Specification blocks in the online Simulink documentation
- Extraneous discrete derivative signals in the online Simulink documentation
- Detect read before write in the online Simulink documentation

- Detect write after read in the online Simulink documentation
- Detect write after write in the online Simulink documentation

Data Integrity Diagnostics Pane Renamed, Reorganized

This release changes the name of the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box to the **Data Validity** pane. It also reorganizes the pane into groups of related diagnostics. See Diagnostics Pane: Data Validity in the online Simulink documentation for more information.

Improved Sample-Time Independence Error Messages

When you enable the `Ensure sample time independent` solver constraint (see Periodic sample time constraint for more information), Simulink software generates several error messages if the model is not sample-time independent. In previous releases, these messages were not specific enough for you to determine why a model failed to be sample-time independent. In this release, the messages point to the specific block, signal object, or model parameter that causes the model not to be sample-time independent.

User Interface Enhancements

Model Viewing

This release adds the following model viewing enhancements:

- A command history for pan and zoom commands (see Viewing Command History in the Simulinkdocumentation)
- Keyboard shortcuts for panning model views (see Model Viewing Shortcuts in the Simulinkdocumentation)

Customizing the Simulink User Interface

This release allows you to use M-code to perform the following customizations of the standard Simulink user interface:

- Add custom commands to the Model Editor's **Tools** menu (see Disabling and Hiding Dialog Box Controls in the Simulink documentation)
- Disable, or hide widgets on Simulink dialog boxes (see Disabling and Hiding Dialog Box Controls in the Simulink documentation)

MEX-Files

MEX-Files on Windows Systems

In this release, the extension for files created by the MATLAB mex command on Windows systems has changed from `dll` to `mexw32` or `mexw64`.

Compatibility Considerations

If you have implemented any S-functions in C, Ada, or Fortran or have models that reference other models, you should

- Recreate any `mexopts.bat` files (other than the one in your MATLAB preferences directory) that you use to build S-functions and model reference simulation targets
- Rebuild your S-functions

MEX-File Extension Changed

In this release, the extension for files created by the MATLAB mex command has changed from `dll` to `mexw32` (and `mexw64`).

Compatibility Considerations

If you use a `mexopts.bat` file other than the one created by the `mex` command in your MATLAB preferences directory to build Accelerator targets, you should recreate the file from the `mexopts.bat` template that comes with this release.

R14SP2

Version: 6.2

New Features

Compatibility Considerations

Multiple Signals on Single Set of Axes

Viewers can now display multiple signals on a single set of axes.

Logging Signals to the MATLAB Workspace

Viewers can now log the signals that they display to the MATLAB base workspace. See Exporting Signal Data Using Signal Logging for more information.

Legends that Identify Signal Traces

Viewers can now display a legend that identifies signal traces.

Displaying Tic Labels

Viewers can now display tic labels both inside and outside scope axes.

Opening Parameters Dialog Box

You can open a viewer's parameters dialog box by right-clicking on the viewer scope.

Rootlevel Input Ports

Compatibility Considerations

If you save a model with rootlevel input ports in this release and load it in a previous release, you will get the following warning:

```
Warning: model, line xxx block_diagram does not have a parameter
named 'SignalName'.
```

You can safely ignore this warning.