Simulink® Release Notes

Contents

Summary by Version

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

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V6.6.1 (R2007a+)	No	No	Bug Reports Includes fixes	Printable Release Notes: PDF No changes to documentation
V6.6 (R2007a)	Yes Details	Yes Summary	Bug Reports Includes fixes	Current product documentation
V6.5 (R2006b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V6.4.1 (R2006a+)	No	No	Bug Reports	No
V6.4 (R2006a)	Yes Details	Yes Summary	Bug Reports	No
V6.3 (R14SP3)	Yes Details	Yes Summary	Bug Reports	No
V6.2 (R14SP2)	Yes Details	Yes Summary	Bug Reports	No
V6.1 (R14SP1)	Yes Details	Yes Summary	Fixed Bugs	No

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
V6.0 (R14)	Yes Details	Yes Summary	Fixed Bugs	No
V5.1 (R13SP1)	Yes Details	No	Fixed Bugs	Printable Release Notes: PDF V5.1 product documentation
V5.0.1 (R13.0.1)	No	Yes Summary	Fixed Bugs	No
V5.0 (R13)	Yes Details	Yes Summary	Fixed Bugs	No
V4.1 (R12+)	Yes Details	Yes Summary	Fixed Bugs	No
V4.0 (R12)	Yes Details	Yes Summary	No	No

About Release Notes

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

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

New Features and Changes

These include

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

Version Compatibility Considerations and Limitations

When a new feature or change introduces a known incompatibility between versions or limitations for new or existing features, its description includes a **Compatibility Considerations and Limitations** subsection that details the impact. For a list of all new features and changes that have compatibility impact or limitations, see the "Compatibility and Limitations Summary for Simulink" on page 130.

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

Fixed Bugs and Known Problems

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

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

Related Documentation at Web Site

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

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

Version 6.6.1 (R2007a+) Simulink

This table summarizes what's new in V6.6.1 (R2007a+):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
No	No	Bug Reports Includes fixes	Printable Release Notes: PDF No changes to documentation: Current product documentation (V6.6)

Version 6.6 (R2007a) Simulink

This table summarizes what's new in Version 6.6 (R2007a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes Summary	Bug Reports Includes fixes	Current product documentation

New features and changes introduced in this version are

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®
- S-functions, including Level-2 M-File S-functions
- Stateflow® charts

For further details, see:

Simulink 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 Using 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
- 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 M-File 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 S-Function M-Files

To use multidimensional signals in Level-2 S-function M-files, you must use the new `Simulink.MSFcnRunTimeBlock` method, `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 Using Simulink documentation for details.
- `PreDeleteFcn`: Allows you to assign a function to call before the block is deleted. See "Block Callback Parameters" in the Using Simulink documentation for details.
- `StaticLinkStatus`: Allows you to obtain the link status of a block without updating out-of-date reference blocks See "Library Link Status" in the Using 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 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 M-File S-Function Block

If a model includes a Level-2 M-File 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. (see Model File Change Notification in “General Simulink Preferences”).
- Warning to notify the user if multiple models or libraries with the same name exist, as Simulink may not be using the one the user expects. (see “Avoiding Problems with 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
- "Using the Legacy Code Tool to Automate the Generation of Files for Fully Inlined S-Functions" 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. 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

Embedded MATLAB introduces 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 now supports 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. Supported functions in the "Embedded MATLAB Function Library Reference" 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 "Working with Embedded MATLAB" 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 Using 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 20
- "Derivative and Integral Functions" on page 20
- "Discrete Math Functions" on page 20
- "Exponential Functions" on page 20
- "Filtering and Convolution Functions" on page 20
- "Logical Operator Functions" on page 20
- "Matrix and Array Functions" on page 20
- "Polynomial Functions" on page 21
- "Set Functions" on page 21
- "Specialized Math" on page 21
- "Statistical Functions" on page 21

See "Embedded MATLAB Function Library Reference" 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 no longer supports `&` 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 now supports the Fixed-Point Toolbox `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 *Embedded 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 Models" and "Specifying a Simulation Start and Stop Time" for information about these capabilities. See "Referencing Configuration Sets" on page 25 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 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 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 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. The MathWorks discourages using implicit signal resolution except for fast prototyping, because implicit resolution slows performance, complicates model validation, and can have nondeterministic effects. Simulink 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 "Configuration Sets" and "Referencing Configuration Sets" 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 "Creating a Freestanding Configuration Set" 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 transparently converts the bus to a vector, allowing the block to accept the bus.

However, The 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 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.
- "Intermixing Composite Signal Types" — Ambiguities that arise when composite signal types are intermixed, and the tools available for eliminating such mixtures.
- "Using Diagnostics for Mixed Composite Signals" — Two diagnostic options for detecting mixed composite signals: and .
- "Using the Model Advisor for Mixed Composite Signals" — 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 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 "Customize Contents Pane" 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 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 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 37 for more information.

Version 6.5 (R2006b) Simulink

This table summarizes what's new in Version 6.5 (R2006b):

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

New features and changes introduced in this version are

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

The MathWorks plans on obsoleting the PreLookup Index Search and Interpolation (n-D) Using PreLookup blocks in a future release. In the meantime, the 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 *Writing S-Functions* for more information.

Simulink Now Uses Internal MATLAB Functions for Math Operations

In previous releases, Simulink 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®. In this release, Simulink 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 operate on the same platform, they produce the same numerical results.

In particular, Simulink 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 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 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 to the base workspace during simulation.

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 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 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 "Determining Why Subsystem Code Is Not Reused".

Sample Hit Time Adjusting Diagnostic

The **Sample hit time adjusting** diagnostic controls whether Simulink 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 "Referencing Model 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 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`.

Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error

In this release, attempting to reference an error 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 "Linking Mask Parameters to Block 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 gives 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 39
- "Discrete Math" on page 39
- "Exponential" on page 39
- "Interpolation and Computational Geometry" on page 39
- "Linear Algebra" on page 40
- "Logical" on page 40

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

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.) Embedded MATLAB does not compile or generate code for extrinsic functions; instead, it sends 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 uses implicit rules to handle calls to external functions:

- For simulation, Embedded MATLAB sends the function to MATLAB for execution
- For code generation, Embedded MATLAB checks 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 proceeds with code generation, but excludes the function call from the generated code. Otherwise, Embedded MATLAB generates a compiler error.

In future releases, Embedded MATLAB will apply these rules only to external functions that you call as extrinsic functions. Otherwise, Embedded MATLAB will compile external functions by default, potentially causing unpredictable behavior or generating errors. For reliable simulation and code generation, The 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 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 silently cast character array outputs to `int8` scalar arrays. This behavior does not match MATLAB, which represents characters in 16–bit unicode.

Version 6.4.1 (R2006a+) Simulink

This table summarizes what's new in V6.4.1 (R2006a+):

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

Version 6.4 (R2006a) Simulink

This table summarizes what's new in V6.4 (R2006a):

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

New features and changes introduced in this version are

• "Embedded MATLAB Function Block Features and Changes" on page 49

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 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 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 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 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 (see Atomic Subsystem). Selecting this command converts an atomic subsystem to a model reference (see "Converting Subsystems to Model References" in the online Simulink documentation for more information).

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 . See “Consulting 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 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 opens the model referenced by the block instead of the block’s parameter dialog box as in previous releases.

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 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 detects the maximum number of consecutive zero crossings allowed. You can specify the criteria that Simulink 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 uses to trigger the **Min step size violation** diagnostic (see 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 "Setting Embedded MATLAB Function Block Properties" in the online Simulink documentation.

Nontunable Option Allows Use of Parameters in Constant Expressions

The **Data** properties dialog for the 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 an Embedded 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, including [Slope Bias] scaling (see "Specifying Fixed-Point Data Properties" in the online Simulink documentation). For information about the features added to the Fixed-Point Toolbox, 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 (see 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 50.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide new runtime library functions in the following categories:

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 does 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 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 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 generates 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 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.

Version 6.3 (R14SP3) Simulink

This table summarizes what's new in V6.3 (R14SP3):

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

New features and changes introduced in this version are organized by these topics:

Model Referencing

This topic contains new features and changes for model reference:

New Features and Changes

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 Models" in the Using 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 the Real-Time Workshop to generate a standalone executable (Real-Time Workshop target) for the model. See "Model Referencing Limitations" in the Simulink user's guide documentation for information on other limitations.

Referenced Models Without Root I/O Can Inherit Sample Times. Previous releases of Simulink 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 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 "Browsing Model Reference Dependencies" in the Using Simulink documentation 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 introduces 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 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 59).

Modeling Enhancements

Annotations

This release introduces the following enhancements to model annotations:

- Annotation properties dialog box (see “Annotations Properties Dialog Box” in the Using Simulink documentation)
- Annotation callback functions (see “Annotation Callback Functions” in the Using Simulink documentation)
- Annotation application programming interface (see “Annotations API” in the Using Simulink documentation)

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 “Adding Custom Viewers and Generators to the Signal & Scope Manager” in the Using 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

In this release, the Embedded MATLAB Function block supports many Fixed-Point Toolbox functions. This allows you to generate code from models that contain fixed-point M functions. See "Working with the Fixed-Point Embedded MATLAB Subset" in the Fixed-Point Toolbox documentation for more information.

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

Embedded MATLAB now supports input triggers and function-call outputs. See "Ports and Data Manager" in the Using Simulink documentation 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 Using Simulink 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 Using 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 Data from the MATLAB Workspace” in the Using 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 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 releases adds the following simulation diagnostics:

- in the Using Simulink documentation
- in the Using Simulink documentation
- in the Using Simulink documentation

- in the Using Simulink documentation
- in the Using 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 in the Using Simulink documentation for more information.

Improved Sample-Time Independence Error Messages

When you enable the `Ensure sample time independent` solver constraint (see for more information), Simulink 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 “View Command History” in the Using Simulink documentation)
- Keyboard shortcuts for panning model views (see “Model Viewing Shortcuts” in the Using Simulink documentation)

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 Using Simulink documentation)

- Disable, or hide widgets on Simulink dialog boxes (see "Disabling and Hiding Dialog Box Controls" in the Using Simulink documentation)

MEX-Files on Windows

In this release, the extension for files created by the MATLAB `mex` command on Windows 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

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 builtin data types and `Simulink.NumericType` objects to specify the data types of `Simulink.Signal` objects. See the `Simulink.Signal` documentation for more information.

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.

Version 6.2 (R14SP2) Simulink

This table summarizes what's new in V6.2 (R14SP2):

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

New features, changes, and limitations in this version are

See the Simulink 6.2 documentation for more information on these enhancements.

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 workspace.

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.

Version 6.1 (R14SP1) Simulink

This table summarizes what's new in V6.1 (R14SP1):

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

New features and changes introduced in this version are:

In this release, Simulink no longer provides the user with the ability to change the values of source block parameters through either a dialog box or the Model Explorer while a simulation is running.

**Changes described in this section reflect Simulink reprogramming implemented to comply with a court decision regarding patent litigation*.

Changed Source Dialog Box Behavior

In this release, opening the dialog box of a source block with tunable parameters causes a running simulation to pause. While the simulation is paused, you can edit the parameter values displayed on the dialog box. However, you must close the dialog box to have the changes take effect and allow the simulation to continue. Similarly, starting a simulation causes any open dialog boxes associated with source blocks with tunable parameters to close.

Since you can no longer change source block parameters while a simulation is running, this release removes the **Apply** button from the dialog boxes of source blocks.

Note In this release, as in previous releases, if you enable the **Inline parameters** option, Simulink does not pause the simulation when you open a source block's dialog box because all of the parameter fields are disabled and can be viewed but cannot be changed.

Changed Model Explorer Source Behavior

In this release, the parameter fields in both the list view and the dialog pane of the Model Explorer have been disabled and the **Apply** button has been removed for source blocks with tunable parameters while a simulation is running. As a result, you can no longer use the Model Explorer to change source block parameters while a simulation is running.

Affected Blocks

Blocks affected are all source blocks with tunable parameters, including the following blocks.

- Simulink source blocks, including
  - Band-Limited White Noise
  - Chirp Signal
  - Constant
  - Pulse Generator
  - Ramp
  - Random Number
  - Repeating Sequence
  - Signal Generator
  - Sine Wave
  - Step

  - Uniform Random Number
- User-developed masked subsystem blocks that have one or more tunable parameters and one or more output ports, but no input ports.
- S-Function and M-file (level 2) S-Function blocks that have one or more tunable parameters and one or more output ports but no input ports.
- Source blocks in other MathWorks products, including:
  - CDMA Reference Blockset
  - Communications Blockset
  - Embedded Target for TI C6000 DSP
  - Signal Processing Blockset (formerly DSP Blockset)
  - Simulink Fixed Point (formerly Fixed-Point Blockset)
  - System Identification Toolbox
  - xPC Target
  - xPC TargetBox®

  See the release notes for each product for a list of that product's source blocks affected by the changes in this release.

Model Load Warnings

Compatibility Considerations

If you open a model in Simulink 6.0 that was created or saved with Simulink 6.1, Simulink 6.0 displays warnings that the following parameters are undefined:

- `StrictBusMsg`
- `MdlSubVersion`

Depending on the model, Simulink 6.0 may also display a warning that the parameter `BusObject` is not defined. You can safely ignore these warnings.

Version 6.0 (R14) Simulink

This table summarizes what's new in V6.0 (R14):

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

New features and changes introduced in this version are organized by these topics:

Model Explorer

The Model Explorer is a new tool that lets you quickly navigate, view, create, configure, search, and modify all data and properties of a Simulink model or Stateflow chart. See "The Model Explorer" in the online Simulink help for more information.

Configuration Sets

This release introduces configuration sets. A configuration set is a named set of values for simulation parameters, such as solver type and simulation start or stop time. Every new model is created with a configuration set that is initialized from a global default configuration set. You can create additional configuration sets for a given model and designate any of them as the active set with the click of a mouse button. See "Configuration Sets" in the online Simulink documentation for more information.

Configuration Parameters Dialog Box

This release replaces the **Simulation Parameters** dialog box with the **Configuration Parameters** dialog box. The **Configuration Parameters** dialog box allows you to set a model's active configuration parameters. You can also use the Model Explorer to set the active configuration parameters as well as inactive parameters. See for more information.

Model Referencing

This release introduces model referencing, a feature that lets a model include other models as modular components. You include other models in a model by using Model blocks to reference the included models. Like subsystems, model referencing allows you to organize large models hierarchically, with Model blocks representing major subsystems. However, model referencing

has significant advantages over subsystems in many applications. The advantages include:

- Modular development

  You can develop the referenced model independently from the models in which it is used.

- Inclusion by reference

  You can reference a model multiple times in another model without having to make redundant copies. Multiple models can also reference the same model.

- Incremental loading

  The referenced model is not loaded until it is needed, speeding up model loading.

- Incremental code generation

  Simulink and the Real-Time Workshop create binaries to be used in simulations and standalone applications to compute the outputs of the included blocks. Code generation occurs only for models that have changed.

See "Referencing Models" in the online Simulink documentation for more information. For a demonstration of a way to automate conversion of an existing model's subsystems to model references, execute `mdlref_conversion` at the MATLAB Command Line. For a summary of limitations on the use of model referencing in this release, see "Model Referencing Limitations".

Model Workspaces

In this release, Simulink provides each model with its own workspace for storing data. Models can access data in their own workspaces as well as data in models that reference them and in the base (i.e., MATLAB) workspace. Model workspaces allow you to create data for one model without fear of inadvertently altering another model's data. See "Working with Model Workspaces" for more information.

Implicit Fixed-Step Solver

This release includes a new fixed-step solver named `ode14x`. This is an implicit, extrapolating fixed-step solver whose extrapolation order and number of Newton's method iterations can be specified via Simulink configuration parameters. The ode14x solver is faster than Simulink's explicit fixed-step solvers for certain types of stiff systems that require a very small step size to avoid unstable solutions.

The Signal and Scope Manager

The Signal and Scope Manager is a new Simulink feature that enables you to globally manage signal generators and viewers. See "Signal & Scope Manager" in the online Simulink help for more information.

Data Object Type Enhancements

This release introduces the following types of objects for specifying the properties of model signals and parameters (i.e., model data):

Object Class	Purpose
`Simulink.AliasType`	Specify another name for a data type.
`Simulink.NumericType`	Define a custom data type.
`Simulink.StructType`	Define a data structure, i.e., a type of signal or parameter comprising data of different types.
`Simulink.Bus`	Define a signal bus.

See "Working with Data Types" and "Data Object Classes" in the Simulink online documentation for more information.

This release also adds the following properties to `Simulink.Signal` class:

- `Dimensions`
- `SampleTime`
- `SamplingMode`

- `DataType`
- `Complexity`

Simulink checks the consistency of these properties against the values set on the ports/dwork elements associated with each signal object.

Note If an attribute is set as `auto` / `-1` (not specified), then no consistency checking is done.

Block Enhancements

This release includes the following block-related enhancements.

New Blocks

This release introduces the following blocks.

- The Signal Conversion block enables you to convert virtual buses to nonvirtual buses, and vice versa.
- The Environment Controller block's output depends on whether the model is being used for simulation or code generation.
- The Bias block adds a specified bias value to its input and outputs the result.
- Embedded MATLAB Function block enables you to include MATLAB code in models from which you intend to generate code, using the Real-Time Workshop.
- The Model block allows you to include other models in a model (see "Model Referencing" on page 71).

Fixed-Point-Capable Blocks

This release adds fixed-point data capability to many existing Simulink blocks and includes fixed-point blocks previously available only with the Fixed-Point Blockset. To use the fixed-point data capability of these blocks, you must install the Simulink Fixed-Point product on your system. See "Fixed-Point Data" in the online Simulink documentation for more information.

Port Values Display

This release of Simulink can display block outputs as data tips on a block diagram while a simulation is running. This allows you to observe block outputs without having to insert Scope or Display blocks. See "Displaying Block Outputs" in the online Simulink documentation for more information.

User-Specifiable Sample Times

This release expands the number of blocks with user-specifiable sample times to include most builtin Simulink blocks. In previous releases, most builtin blocks inherited their sample times directly or indirectly from the blocks to which they were connected. In this release, most blocks continue to inherit their sample times by default. However, you can override this default setting in most cases to specify a nondefault sample time, using either the block's parameter dialog box or a `set_param` command. This avoids the need to use Signal Specification blocks to introduce nondefault sample times at specific points in a model.

Improved Initial Output Handling

In previous Simulink releases, the Constant, Initial Condition, Unit Delay, and other blocks write out their initial output values in their `mdlStart` method. This behavior can cause unexpected block output initialization. For example, if a Constant block in an enabled subsystems feeds an Outport block whose IC is set to `[]`, the Constant value appears even when the enabled subsystem is not enabled.

It is desirable in some cases for a block to write its initial output value in its `mdlStart` method. For example, discrete integrator block may need to read the value from its external IC port to setting the initial state in `mdlInitialize` method.

This release addresses these problems by implementing a hand-shaking mechanism for handling block initial output. Under this mechanism, a block only computes its initial output value when it is requested by its downstream block. For example, if a Constant block feeds the external IC port of a Discrete Integrator block, the discrete integrator block's external IC port requests the Constant block to compute its initial output value in its `mdlStart` method.

Bus-Capable Nonvirtual Blocks

In previous releases, Simulink propagated buses only through virtual blocks, such as subsystems. In this release, Simulink also propagates buses through the following nonvirtual blocks:

- Memory
- Merge
- Switch
- Multiport Switch
- Rate Transition
- Unit Delay
- Zero-Order Hold

Some of these blocks impose constraints on bus propagation through them. See the documentation for the individual blocks for more information.

Duplicate Input Ports

This release allows you to create duplicates of Inport blocks in a model. A model can contain any number of duplicates of an original Inport block. The duplicates are graphical representations of the original intended to simplify block diagrams by eliminating unnecessary lines. The duplicate has the same port number, properties, and output as the original. Changing a duplicate's properties changes the original's properties and vice versa. See the Inport block documentation for more information.

Inport/Outport Block Display Options

Inport and Outport blocks can now optionally display their port number, signal name, or both the number and the name. See the online documentation for the Inport and Outport blocks for more information.

Zero- and One-Based Indexing

In this release, some blocks that use indices provide the option for indices to start at 0 or 1. The default is one-based indexing to maintain compatibility

with previous releases. Blocks that now support zero- or one-based indexing include

- Selector
- For Iterator
- Assignment

Runtime Block API

This release introduces an application programming interface (API) that enables programmatic access to block data, such as block inputs and outputs, parameters, states, and work vectors, while a simulation is running. You can use this interface to develop MATLAB programs capable of accessing block data while a simulation is running or to access the data from the MATLAB command line. See "Accessing Block Data During Simulation" for more information.

Command-Line API to Signal Builder Block

This release provides a command, `signalbuilder`, for creating and accessing Signal Builder blocks in a model.

Signal Enhancements

This release includes the following signal-related enhancements.

Test Point Indicators

This release can optionally use indicators on a block diagram to indicate signals that are test points. See "Displaying Test Point Indicators" in the online documentation for more information.

Signal Logging

This release allows you to log signal data to the workspace during simulation (see "Logging Signals" for more information).

Internal Signal Structures Revamped

This release revamps the `sigmap`, `siglists` and `sigregions` structures to support signal logging and other signal-related enhancements.

Compatibility Considerations. S-functions created prior to Version 6 (R14). that access the `sigmap`, `siglists` and `sigregions` structures might generate segmentation violations. To avoid this, recompile the S-functions in Version 6 (R14) or subsequent releases.

Edit-Time Signal Label Propagation

In this release, when you change a signal label, Simulink automatically propagates the change to all downstream instances of the label. You do not have to update the diagram as in previous releases.

Bus Editor

The new Bus Editor enables you to create and modify bus objects in Simulink's base (MATLAB) workspace. See "Bus Editor" for more information.

Rate Transition Enhancements

This release provides the following enhancements to the handling of rate transitions in models.

Rate Transition Block Determines Transition Type Automatically

The Rate Transition block now determines the type of transition that occurs between the source and destination block (i.e., fast-to-slow or slow-to-fast). Therefore, this release eliminates the transition type option on the block's parameter dialog.

Automatic Insertion of Rate Transition Blocks

This release introduces an option to insert hidden rate transition blocks automatically between blocks that operate at different rates. This saves you from having to insert rate transition blocks manually in order to avoid illegal rate transitions. The inserted blocks are configured to ensure that data is transferred deterministically and that data integrity is maintained during the transfer. See in the online Simulink documentation for more information.

User-Specifiable Output Sample Time

The Rate Transition Block's parameter dialog box contains a new parameter: **Output Port Sample Time**. This parameter allows you to specify the output rate to which the input rate is converted. If you do not specify a rate, the Rate Transition block inherits its output rate from the block to which its output is connected.

Execution Context Enhancements

This releases introduces the following enhancements to execution context propagation.

Enabling Execution Context Propagation

This release allows you to specify whether to permit execution contexts to be propagated across a conditionally executed subsystem's boundary. See the documentation for the Subsystem block for more information.

Execution Context Indicator

This release optionally displays a bar across each input port and output port of a subsystem that does not permit propagation of the subsystem's execution context. To enable this option, select **Block Displays->Execution context indicator** from the model editor's **Format** menu.

Algebraic Loop Minimization

This release can eliminate some types of algebraic loops involving atomic or enabled subsystems or referenced models. See "Eliminating Algebraic Loops" in the online Simulink documentation for more information.

Level-2 M-File S-Functions

This release introduces a new application programming interface (API) for creating custom Simulink blocks based on M code. In contrast to the previous API, designated Level 1, which supported a restricted set of block features, the new API, designated Level 2, supports most standard Simulink block features, including support for matrix signals and nondouble data types. See "Writing Level-2 M-File S-Functions" in the online documentation for more information.

Panning Model Diagrams

You can now use the mouse to pan around model diagrams that are too large to fit in the model editor's window. To do this, position the mouse over the diagram and hold down the left mouse button and the P or Q key on the keyboard. Moving the mouse now pans the model diagram in the editor window.

MATLAB Data Type Conversions

Release 14 introduces changes in the way MATLAB handles conversions from double to standard MATLAB nondouble data types (e.g., int8, uint8, etc.) and from one nondouble data type to another.

Compatibility Considerations

Previous releases of MATLAB use truncation to convert a floating point value to an integer value, e.g., `int8(1.7) = 1`. Release 14 uses rounding, e.g., `int8(1.7) = 2`. See "New Nondouble Mathematics Features" in the Release 14 MATLAB Release Notes for a complete description of the changes in data type conversion algorithms introduced in Release 14.

Such changes could affect the behavior of models that rely on nondouble data type conversions of signals and block parameters. For example, a Gain parameter entered as `int8(3.7)` ends up as `4` in this release as opposed to `3` in previous releases and this difference could change the simulation results. Therefore, if the simulation results for your model differ in Release 14 from previous releases, you should investigate whether the differences result from the changes in data type conversion algorithms, and, if so, modify your model accordingly.

Signal Object Resolution Changes

In previous releases, Simulink attempted to resolve every named signal to a `Simulink.Signal` object of the same name in the MATLAB workspace.

Compatibility Considerations

In this release, Simulink lets you specify whether a named signal or discrete state should resolve to a signal object, using the **Signal Properties** dialog box and the **State Properties** of blocks that have discrete states, such as the

Discrete-Time Integrator. By default, Simulink attempts to resolve every named signal or state to a signal object regardless of whether the model specifies that the signal or state should resolve to a signal object. If the model does not specify resolution for a signal or state and it does resolve, Simulink displays a warning. You can also specify that Simulink attempt to resolve all named signals or states without warning of implicit resolutions (the behavior in previous releases) or that it only resolve signals and states that the model specifies should resolve (explicit resolution).

Explicit signal resolution is the recommended approach for doing signal resolution as it ensures that signals that should be resolved are resolved and signals that should not resolve are not resolved. This release includes a script that facilitates converting models that use implicit signal resolution to use explicit resolution. Enter `help disableimplicitsignalresolution` at the MATLAB command line for more information.

Loading Models Containing Non-ASCII Characters

Release 14 of MATLAB introduces Unicode support. This enhancement allows MATLAB and Simulink to support character sets from different encoding systems.

Compatibility Considerations

This change causes Simulink to behave differently from previous releases when loading a model containing non-ASCII characters. Previous releases load such models regardless of whether the non-ASCII characters are compatible with the current encoding system used by MATLAB. In Release 14, Simulink checks the characters in the model against the current encoding setting of MATLAB. If they are incompatible, Simulink does not load the model. Instead, it displays an error message that prompts you to change to a compatible MATLAB encoding setting, using the `slCharacterEncoding` command.

Change in Sample Time Behavior of Unary Minus Block

Release 14 changes the sample time behavior of the Unary Minus block.

Compatibility Considerations

In Release 13, if the sample time of this block's input is continuous, the sample time of the block and its output is fixed in minor time step. This block is fixed in minor step and the output signal is fixed in minor step when the input is a continuos sample time signal. In Release 14, if the input is continuous, the block and output sample time are continuous also.

Initial Output of Conditionally Executed Subsystems

In this release, the initial output is undefined if the **Initial output** port specifies `[]`.

Compatibility Considerations

In previous releases, if the **Initial output** parameter of an Outport block in a conditionally executed subsystem specified `[]` as the initial output, the initial output of this port was the initial output of the block driving the Outport block.

Execution Context Default Changes

In R14, execution context propagation does not cross a conditionally executed subsystem boundaries by default.

Compatibility Considerations

In R13 SP1 and DACORE2, execution contexts propagate across conditionally executed subsystem boundaries by default. You need to choose the **Propagate execution context across subsystem boundary** option in the subsystem's parameter dialog box.

Version 5.1 (R13SP1) Simulink

This table summarizes what's new in V5.1 (R13SP1):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Fixed Bugs	Printable Release Notes: PDF V5.1 product documentation

New features and changes introduced in this version are:

Sample Time Parameters Exposed

Sample time parameters of most Simulink built-in library blocks have been exposed to the user. That is, the sample time parameter of these blocks has been made accessible via the block's dialog box or `set_param`. This means that most nonvirtual blocks in the Simulink library have a user settable sample time parameter. Prior to this exposure, these blocks had an internal inherited sample time with the exception of the Constant block, which had a constant (`inf`) sample time. By providing access to the sample time parameter, you no longer need to use the Signal Specification block to apply a nondefault sample times to these blocks.

Enhanced Debugger

This release includes enhancements to the Simulink debugger that enable you to step through a simulation showing information not visible in previous releases. The enhancements include

- An expanded command set that now enables you to step a simulation method by method. Previous releases showed only output methods.
- An expanded toolbar that gives you push button access to new debugger commands
- A Simulation Loop pane that shows the current state of the simulation at a glance

---

Note Methods are functions that Simulink uses to solve a model at each time step during the simulation. Blocks are made up of multiple methods. "Block execution" in this documentation is shorthand notation for "block methods execution." Block diagram execution is a multi-step operation that requires execution of the different block methods in all the blocks in a diagram at various points during the process of solving a model at each time step during simulation, as specified by the simulation loop.

These changes allow you to pinpoint problems in your model with greater accuracy. The following sections briefly describe the debugger enhancements. See the Simulink documentation for a detailed description of the new features and their usage.

Enhanced Debugger Commands

This release enhances the following debugger commands:

- `step`

  In previous releases, this command advanced the simulation from the current block Outputs method over any intervening methods to the next block Outputs method. In this release, `step` advances the simulation method by method, or into, over, or out of methods, from the first method executed during the simulation to the last. This allows you to determine the result of executing any model, subsystem, or block method executed

during the simulation, including block Outputs, Update, and Derivative methods as well as solver methods.

- `next`

  In previous releases, this command advanced the simulation to the first block Outputs method executed during the next time step. In this release, it advances the simulation over the next method to be executed, executing any methods invoked by the next method.

- `break`

  In previous releases, this command set a breakpoint at the Outputs method of a specified block. In the current release, it sets a breakpoint at any specified method or on all the methods of a specified block.

- `bafter`

  In previous releases, this command set a breakpoint after the Outputs method of a specified block. In this release, it sets a breakpoint after a specified method or after each of the methods of a specified block.

- `minor`

  In previous releases, this command enabled or disabled stepping across Outputs methods in minor time steps. In the current release, it enables or disables in minor time steps breakpoints set by block for all methods.

New Debugger Commands

This release introduces the following debugger commands:

- `elist`

  Displays the method execution lists for the root system and the nonvirtual subsystems of the model being debugged.

- `etrace`

  Causes the debugger to display a message in the MATLAB Command Window every time a method is entered or exited while the simulation is running.

- where

  Displays the call stack of the method at which the simulation is currently suspended.

Enhanced Debugger Toolbar

The debugger toolbar has been expanded to include buttons for the following versions of the `step` command: `step into`, `step over`, `step out`, and `step top`.

Simulation Loop Pane

This release adds a **Simulation Loop** pane to the debugger GUI that displays by method the point in the simulation loop at which the simulation is currently suspended. The debugger updates the pane after each `step`, `next`, or `continue` command, enabling you to determine at a glance the point to which the command advanced the simulation. The pane also allows you to set breakpoints on simulation loop methods and to navigate to the block at whose method the simulation is currently suspended.

Sorted List Pane

This release renames the **Block Execution List** pane of the debugger GUI to the **Sorted List** pane to reflect more accurately what the pane contains. The Sorted List pane displays for the root system and each nonvirtual subsystem of the model being debugged a sorted list of the subsystem's blocks. The sorted lists enable you to determine the block IDs of a model's blocks.

Context-Sensitive Data Typing of Tunable Parameters

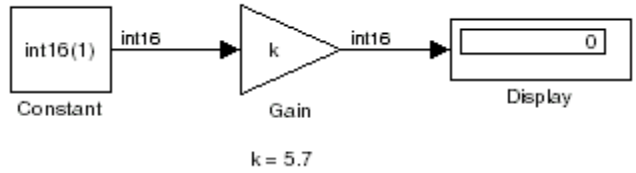
In this release, if a model's **Inline parameters** setting is selected, Simulink regards the data type of a tunable parameter as context-sensitive if the data type is not specified. In particular, this release allows the block that uses the parameter to determine the parameter's data type. By contrast, Release 13 regards the type of the parameter to be `double` regardless of where it is used.

Change in Simulink Behavior

This change affects the behavior of Simulink in two cases. First, in Release 13, if a tunable parameter's data type is unspecified and a block that uses it needs to convert its type from `double` to another type, Simulink by default stops and displays an error message when you update or simulate the model. The error alerts the user to the fact that the type conversion is a downcast and hence could result in a loss of precision. In this release, by contrast, a typecast never occurs because the block itself determines the appropriate

type for the parameter. Hence, in this release, Simulink never generates a downcast error for tunable parameters of unspecified data type.

The following model illustrates the difference in behavior between this release and Release 13 in this case.



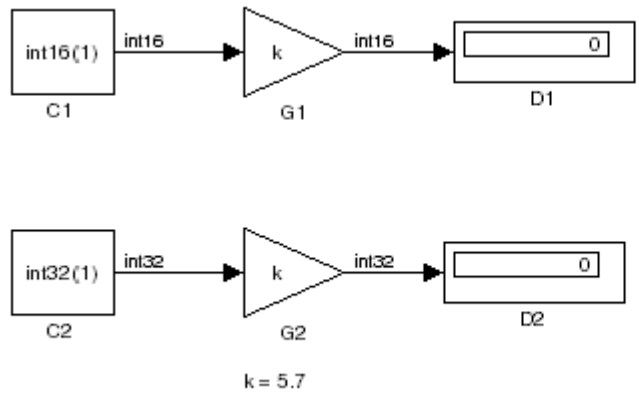
Assume that the model's **Inline parameters** setting is selected (thereby making parameters nontunable by default) and the model declares `k` as a tunable parameter on the **Model Configuration Parameters** dialog box. Also assume that the user has specified the value of `k` on the MATLAB command line as follows:

```
>> k = 5.7
```

In other words, the user has specified a value for `k` but not a data type. In this case, this release regards the type of `k` to be `int16`, the type required by the Gain block to compute its output. By contrast, Release 13 regards the type of `k` to be `double` and hence assumes that the Gain block must downcast `k` to compute its output. Release 13 therefore stops and displays an error message by default in this case when you update or simulate the model.

The behavior of this release also differs from Release 13 in the case where a model uses a tunable parameter of unspecified data type in more than one place in the model and the required data type differs in different places. This case creates a conflict under the assumption that the block in which the parameter is used determines the parameter's data type. This assumption requires Simulink to assign different data types to the same parameter, which is impossible. Therefore, in this release, Simulink signals an error to alert the user to the conflict. By contrast, in Release 13, Simulink does not throw an error because the data type of the parameter is `double` regardless of where it is used. You can avoid the conflicting data types error in Release 13SP1 by specifying the tunable parameter's data type.

The following model illustrates this change in behavior.



The two Gain blocks in this model both use `k`, a tunable parameter of unspecified type, as their gain parameter. Computing the outputs of the blocks requires that the gain parameter be of types `int16` and `int32`, respectively. In Release 13, Simulink regards the data type of `k` to be `double` and the Gain blocks use typecasts to convert `k` to the required type in each case. Simulink simulates the model without error (if the `parameter downcasting` diagnostic is set to `none` or `warning`). By contrast, this release signals an error because this model requires `k` to be both type `int16` and `int32`, an impossibility. You can avoid this error by explicitly specifying `k`'s data type; for example:

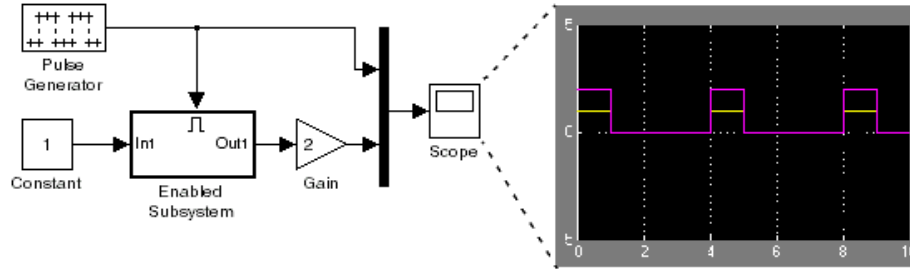
```
k = int16(6);
```

Conditional Execution Behavior

This release augments the conditional input branch behavior of the previous release with a more generalized behavior called *conditional execution (CE)* behavior. The new behavior speeds simulation of models by eliminating unnecessary execution of blocks connected to Switch, Multiport Switch, and conditionally executed blocks.

Note The Simulink documentation has not yet been updated to reflect the new behavior. Consequently, the remainder of this release note provides a detailed explanation of how the behavior works.

As with the conditional input branch behavior available in the previous release, the new behavior ensures that the block methods that make up an input branch of a Switch or Multiport Switch block execute only when the model selects the corresponding switch input. In addition, the new behavior option generalizes this behavior to conditionally executed subsystems. Consider, for example, the following model.



Simulink computes the outputs of the Constant block and Gain Block only when the Enabled Subsystem executes (i.e., at time steps 0, 4, 8, and so on). This is because the output of the Constant block is required and the input of the Gain block changes only when the Enabled Subsystem executes. When CE behavior is off, Simulink computes the outputs of the Constant and Gain blocks at every time step, regardless of whether the outputs are needed or change.

In this example, Simulink regards the Enabled Subsystem as defining an execution context for the Constant and Gain blocks. Although the blocks reside in the model's root system, their block methods are executed as if the blocks reside in the Enabled Subsystem.

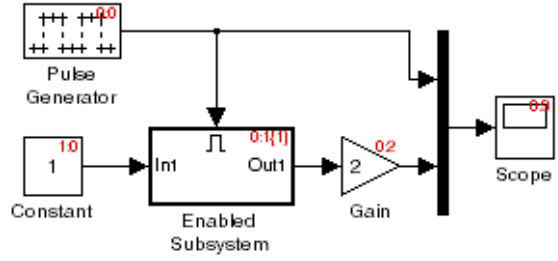
In general, Simulink defines an *execution context* as a set of blocks to be executed as a unit. At model compilation time, Simulink associates an execution context with the model's root system and with each of its

nonvirtual subsystems. Initially, the execution context of the root system and each nonvirtual subsystem is simply the blocks that it contains. Simulink examines whether a block's output is required only by a conditionally executed subsystem or whether the block's input changes only as a result of the execution of a conditionally executed subsystem. If so, Simulink moves the block into the execution context of the conditionally executed system. This ensures that the block methods are executed during the simulation loop only when the corresponding conditionally executed subsystem executes.

Note This behavior treats the input branches of a Switch or Multiport Switch block as invisible, conditionally executed subsystems, each of which has its own execution context that is enabled only when the switch's control input selects the corresponding data input. As a result, switch branches execute only when selected by switch control inputs.

To determine the execution context to which a block belongs, select **Sorted order** from the model window's **Format** menu. Simulink displays the sorted order index for each block in the model in the upper right corner of its icon. The index has the format `s:b`, where `s` specifies the subsystem to whose execution context the block, `b`, belongs.

Simulink also expands the sorted order index of conditionally executed subsystems to include the system ID of the subsystem itself in curly brackets as illustrated in the following figure.



In this example, the sorted order index of the enabled subsystem is `0:1{1}`. The `0` indicates that the enable subsystem resides in the model's root system.

The first 1 indicates that the enabled subsystem is the second block on the root system's sorted list (zero-based indexing). The 1 in curly brackets indicates that the system index of the enabled subsystem itself is 1. Thus any block whose system index is 1 belongs to the execution context of the enabled subsystem and hence executes when it does. For example, the constant block's index, 1:0, indicates that it is the first block on the sorted list of the enabled subsystem, even though it resides in the root system.

Function-Call Subsystem Enhancements

This releases adds the following function-call subsystem-related parameters to the Trigger block:

- The **States when enabling** parameter specifies whether a function-call enable trigger causes Simulink to reset the states of the subsystem containing this Trigger block to their initial values.
- The **Sample time type** parameter specifies whether the function-call subsystem containing the Trigger block is invoked periodically.
- The **Sample time** parameter species the rate at which the function-call subsystem containing the Trigger block is invoked.

See the Trigger block documentation for additional information.

External Increment Option Added To For Iterator Block

This release adds an external increment option to the For Iterator block. Selecting this option causes the block to display an input port for the external increment. The value of this input port at the current time step is used as the value of the block's iteration variable at the next iteration. You can select this option by checking the **Set next i (iteration variable)** externally option on the block's parameter dialog box or by setting its `ExternalIncrement` parameter to `'on'`. See the documentation for the For Iterator block for more information.

Note This enhancement is not backward compatible with R13. Loading models containing For Iterator blocks with this option selected in R13 produces a warning message. Simulating such models in R13 can produce incorrect results.

Performance Improvements

Release R13SP1 includes many performance improvements that were designed to particularly benefit large models (containing on the order of 100,000 blocks and/or more than a few megabytes of parameter data). Speed has been improved and memory consumption reduced for model loading, compilation, code generation, and closing. The various improvements span the Simulink, Stateflow, and Real-Time Workshop products and include:

- Increased speed and decreased memory consumption through improved incremental loading of library blocks that contain Stateflow blocks.
- Increased speed and decreased memory usage through the introduction of a redesigned Signal Specification block. Models saved with the old version of the Signal Specification block should automatically start using the new block when you load the model with this release.
- Increased speed in datatype and sample time propagation during the compile phase of certain models.
- Increased speed in the Stateflow build process for both simulation and Real-Time Workshop targets.
- Increased speed and decreased memory consumption when using N-D Lookup Table blocks that utilize large parameter data.
- Increased speed and decreased memory usage when generating code with Real-Time Workshop or the Simulink Accelerator for models with large parameter sets. This improvement involves writing out parameter references instead of the entire parameter data into the RTW file for parameters whose size exceeds 10 elements. The parameter values for such references are retrieved directly from Simulink during the code generation process.
- Decreased memory usage during various phases of code generation process in Real-Time Workshop or the Simulink Accelerator.

- Improved speed during model close through streamlining of the close process.

Other minor improvements have also been made to improve performance. Your models should experience corresponding speed and memory improvements, to the extent that these changes apply to your specific models and usage scenarios.

Version 5.0.1 (R13.0.1) Simulink

This table summarizes what's new in V5.0.1 (R13.0.1):

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

New features and changes introduced in this version are

Tunable Parameters for Unified Fixed-Point Blocks

Compatibility Considerations

Unified fixed-point blocks with tunable parameters have compatibility problems under certain conditions in Release 13. The problem arises only if a tunable parameter is mapped to a built-in integer or `single` data type. When tunable parameters are mapped to built-in integers or `single`, the code generated by Real Time Workshop will be different for unified blocks than it was for Fixed-Point Blockset blocks in prior releases. There are no compatibility problems if a tunable parameter maps to a nonbuilt-in data type, such as a scaled fixed-point integer.

Tunable parameters are entered in a Simulink model by specifying the name of a MATLAB variable in a block's dialog. This variable can be either a plain MATLAB variable or a Simulink parameter object. In either case, a numerical value will be defined for this tunable parameter by doing an assignment in MATLAB. MATLAB supports several numerical data types including the eight Simulink built-in numerical data types: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`. One of these eight data types can be used when a value is defined for a MATLAB variable. The effect of the data type of the MATLAB variable is significantly different depending on how the tunable parameter is used in Simulink.

For Simulink built-in blocks, the legacy rule is to fully respect the data type used for the value of a MATLAB variable. Whatever data type is used in MATLAB when assigning a value to a variable is also be used when declaring that parameter in code generated by Real Time Workshop. The use of that parameter by a block may require the value to be represented using a different data type. If so, additional code is generated to convert the parameter every time it is used by the block. To get the most efficient code for a given block, the value of the MATLAB variable should use the same data type as is needed by the block.

For Fixed-Point Blockset blocks, the legacy rule is to expect no data type information from the MATLAB variable used for the tunable parameter. A fundamental reason for this is that MATLAB does not have native support for fixed-point data types and scaling, so the Simulink built-in legacy rule could not be directly extended to the general fixed-point case. Many fixed-point blocks automatically determine the data type and scaling for parameters based on what leads to the most efficient implementation of a given block. However, certain blocks such as Constant, as well as blocks that use tunable parameters in multiplication, do not imply a unique best choice for the data type and scaling of the parameter. These blocks have provided separate parameters on their dialogs for entering this information.

In Release 13, many Simulink built-in blocks and Fixed-Point Blockset blocks were unified. The Saturation block is an example of a unified block. The Saturation block appears in both the Simulink Library and in the Fixed-Point Blockset Library, but regardless of where it appears it has identical behavior. This identical unified behavior includes the treatment of tunable parameters. The dissimilarity of the legacy rules for tunable parameters has lead to a shortcoming in the unified blocks. Unified blocks obey the Simulink legacy rule sometimes and the Fixed-Point Blockset legacy rule at other times. If the block is using the parameter with built-in Simulink data types, then the Simulink legacy rule applies. If the block is using the parameter with nonbuilt-in data types, such as scaled fixed-point data types, then the Fixed-Point Blockset legacy rule applies. This gives full backwards compatibility with one important exception.

The backwards compatibility issue arises when a model created prior to R13 uses a Fixed-Point Blockset block with a tunable parameter, and the data type used by the block happens to be a built-in data type. If the block is unified, it will now handle the parameter using the Simulink legacy rule

rather than the Fixed-Point Blockset legacy rule. This can have a significant impact. For example, suppose the tunable parameter is used in a Saturation block and the data type of the input signal is a built-in `int16`. In prior releases, the Fixed-Point Blockset block would have declared the parameter as an `int16`. For legacy fixed-point models, the MATLAB variables used for tunable parameters invariably gave their value using floating-point `double`. The unified Saturation block would now declare the tunable parameter in the generated code as `double`. This has several negatives. The variable takes up six more bytes of memory as a `double` than as an `int16`. The code for the Saturation block now includes conversions from `double` to `int16` that execute every time the block executes. This increases code size and slows down execution. If the design was intended for use on a fixed-point processor, the use of floating-point variables and floating-point conversion code is likely to be unacceptable. It should be noted that the numerical behavior of the blocks is not changed even though the generated code is different.

For an individual block, the backwards compatibility issue is easily solved. The solution involves understanding that the Simulink legacy rule is being applied. The Simulink legacy rule preserves the data type used when assigning the value to the MATLAB variable. The problem is that an undesired data type will be used in the generated code. To solve this, you should change the way you assign the value of the tunable parameter. Determine what data type is desired in the generated code, then use an explicit type cast when assigning the value in MATLAB. For example, if `int16` is desired in the generated code and the initial value is `3`, then assign the value in MATLAB as `int16(3)`. The generated code will now be as desired.

A preliminary step to solving this issue with tunable parameters is identifying which blocks are affected. In most cases, the treatment of the parameter will involve a downcast from `double` to a smaller data type. On the **Diagnostics** tab of the **Simulation Parameters** dialog is a line item called `Parameter downcast`. Setting this item to `Warning` or `None` will help identify the blocks whose tunable parameters require reassignment of their variables.

In R13, the solution described above did not work for three unified blocks: Switch, Look-Up Table, and Lookup Table (2-D). These blocks caused errors when the value of a tunable parameter was specified using integer data types. This was a false error and has been removed. Using an explicit type cast when assigning a value to the MATLAB variable now solves the issue of generating code with the desired data types.

Version 5.0 (R13) Simulink

This table summarizes what's new in V5.0 (R13):

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

New features and changes introduced in this version are organized by these topics:

Note Simulink 5.0 incorporates changes introduced in Simulink 4.1.1, which was initially released in Web-downloadable form after Release 12.1 was released, but before Release 13. These Release Notes describe those changes, as well as other changes introduced after Version 4.1.1.

Block Enhancements

Simulink 5.0 includes the following block-related enhancements:

Fixed-Point Block Library

Simulink now includes the latest version (4.0) of the Fixed-Point Blockset. The library was previously available only as a separately installed option. You must have a Fixed-Point Blockset license to run models containing fixed-point blocks in fixed-point mode. However, you can open, edit, and run such models in floating-point mode, regardless of whether you have a Fixed-Point Blockset license. This change facilitates sharing of fixed-point models in large organizations by eliminating the need for all users in a group to have a Fixed-Point Blockset license in order to run or modify models containing fixed-point blocks. See "Installation and Licensing" in the Simulink Fixed-Point Blockset release notes for information on how to run models containing fixed-point blocks when you do not have a Fixed-Point Blockset license.

This release also unifies many core Simulink and Fixed-Point Blockset blocks that have similar functionality. For example, the Sum block in the Simulink Math Operations library and the Sum block in the Fixed-Point Blockset Math library are now the same block. As a result, you no longer have to replace any of the unified blocks when switching from built-in to fixed-point data types and vice versa. You can change the data types of the blocks simply by selecting the appropriate settings on their parameter dialog boxes. See "Unified Simulink and Fixed-Point Blockset Blocks" in the Simulink Fixed-Point Blockset release notes for more information and for a list of blocks that this release unifies.

Note When you open an existing model, Simulink 5.0 updates the model to use the unified version of a standard or Fixed-Point Blockset block wherever an instance of that block occurs in the model. Simulink sets the parameters of the unified block to preserve the behavior of the original block. For example, wherever your existing model contains a Sum block from the Fixed-Point Blockset library, Simulink replaces the Fixed-Point Blockset version with a unified Sum block set to operate as a fixed-point block. This automatic updating ensures that your existing model runs the same in Simulink 5.0 as it did in previous releases of Simulink.

Lookup Table Editor

The Lookup Table Editor allows you to find and edit the contents of look-up tables used by look-up table blocks. See “Lookup Table Editor” in the online Simulink documentation for more information.

Model Verification Block Library

Simulink now includes a library of model verification blocks that enable you to create self-validating models. For example, you can use the blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error-checking off by disabling the model verification blocks. You do not have to physically remove them from the model. The library includes set of blocks preconfigured to check for common types of errors, for example, signals that exceed a specified upper or lower bound. See “Model Verification” in the online Simulink documentation for more information.

Signal Builder Block

The new Signal Builder block allows you to create interchangeable groups of signal sources and quickly switch the groups into and out of a model. The Signal Builder block’s signal editor allows you to define the waveforms of the signals output by the block. You can specify any waveform that is piecewise linear. Signal groups can greatly facilitate testing a model, especially when used in conjunction with Simulink assertion blocks and the optional Model Coverage Tool. See “Working with Signal Groups” for more information.

DocBlock

The new DocBlock block allows you to create text that documents a model and save that text with the model.

Rate Transition Block

Simulink now includes a Rate Transition block that allows you to specify the data transfer mechanism between two rates of a multirate system. See Rate Transition in the online Simulink block reference for more information.

Block Library Reorganization

The Simulink Block Library has been reorganized to simplify accessing blocks with related functionality.

Model Linearization Blocks

This release introduces two blocks that generate linear models from a Simulink model at various times during a simulation. The Time-Based Linearization block generates linear models at specified time steps. The Trigger-Based Linearization block generates models when triggered by events appearing at its trigger port.

Data Store Read/Write Block Navigation

This release allows you to navigate among the blocks that define and access data stores by clicking on the names of associated blocks listed in the dialog box of each block. See Data Store Memory, Data Store Read, and Data Store Write for more information.

Enhanced S-Function Builder

The S-Function Builder has been enhanced to generate S-functions with the following additional capabilities

- Multiple ports
- Support for all builtin datatypes
- Support for 2-D signals
- Support for complex signals

See “Building S-Functions Automatically” for more information.

Miscellaneous Block Enhancements

This release introduces the following enhancements to Simulink blocks.

Math Function Block. This release significantly speeds up the simulation of the Math Function block’s exponential math functions. All functions now support both double- and single-precision floating-point inputs and outputs. The `mod` and `rem` functions also support inputs and outputs of all integer types. The `transpose` and `hermitian` functions support all data types. When optimizations are enabled, the conjugate operation on a real signal invokes the block reduction optimization, as that case is a no-op. In-place multiplies for the magnitude^2 operation are used for reused block I/O on real signals.

Gain Block. The Gain block now performs block reduction when block reduction is on, `inline parameters=ON`, and the gain is both nontunable and unity.

Width Block. The Width block now includes a parameter to specify the datatype of the output.

Real Data Type Support. The following blocks now operate on both double precision and single precision floating point signals:

- Dot Product
- Trigonometric
- Matrix Inversion

Block Data Type Table

To view a table that summarizes the data types supported by the blocks in the Simulink and Fixed-Point block libraries, execute the following command at the MATLAB command line:

```
showblockdatatypetable
```

Simulation Enhancements

Simulink 5.0 includes the following new features and enhancements to simulation of Simulink models.

Invalid Loop Highlighting

Simulink now detects and highlights several kinds of invalid loops:

- Loops that create invalid function-call connections or an attempt to modify the input/output arguments of a function call
- Loops containing non-latched triggered subsystems
- Self-triggering subsystems
- Loops containing action subsystems in a cycle

This makes it is easier to identify and fix the loop. See "Avoiding Invalid Loops" for more information.

Algebraic Loop Highlighting

Simulink now optionally highlights algebraic loops when you update or simulate a model. See "Highlighting Algebraic Loops" for more information. The `ashow` debug command without any arguments now lists all of a model's algebraic loops in the MATLAB command window.

Conditional Execution Behavior

This release introduces a new optimization called conditional execution behavior. Previously, when simulating models containing Switch or Multiport Switch blocks, Simulink executed all blocks required to compute all inputs to each switch at each time step. In this release, Simulink, by default, executes only the blocks required to compute the control input and the data input

selected by the control input at each time step. Similarly, code generated from the model by Real-Time Workshop executes only the code needed to compute the control input and the selected data input. This optimization speeds simulation and execution of code generated from the model. See “Conditional Execution Behavior” for more information.

Reorganized Simulation Diagnostics

The **Diagnostics Pane** of the **Simulation Parameters** dialog box now groups diagnostics by functionality. This makes it easier to find and configure related diagnostics.

Enhanced Diagnostic Viewer

This release introduces an enhanced Diagnostic Viewer. Improvements include

- Identical appearance on UNIX and Windows
- Hyperlinks to Simulink, Stateflow, and Real-Time Workshop objects that caused the errors displayed in the viewer
- Sortable error list

  Clicking a column head sorts the error list by the contents of that column.
- Configurable content

  The **View** menu allows you to choose which information to display in the viewer.
- Selectable font size

  The **FontSize** menu allows you to choose the size of the font used to display error messages.

See “Simulation Diagnostics Viewer” for more information.

Compatibility Considerations. New version of the Diagnostic Viewer is not supported on the HP and IBM platforms.

Modeling Enhancements

The following enhancements facilitate creation of Simulink models.

Enhanced Mask Editor

This release introduces changes to the Mask Editor designed to improve usability. Changes include

- Block parameter information moves from the **Initialization** pane to a new pane entitled **Parameters**.
- The **Parameters** pane allows you to specify a callback function to be called when the value of a parameter changes.
- The **Parameters** pane allows you to specify via check boxes whether a parameter is visible on the masked block's dialog box and whether a parameter is tunable.
- The **Icon** pane provides a list of examples of all the types of drawing commands that can be used to draw the block's icon.

See "Creating Block Masks" in the online Simulink documentation for more information.

Compatibility Considerations.

- Simulink Editor's **Find** dialog is not supported on the HP and IBM platforms. Use the `find_system` command instead.
- Enhanced version of Mask Editor is not supported on the HP and IBM platforms.

Including Symbols and Greek Letters in Block Diagrams

This release allows you to include symbols, Greek letters, and other formatting in annotations, masked subsystem port labels, and masked subsystem icon text. You do this by including TeX formatting commands in the annotation, port label, or icon text.

Production Hardware Characteristics

Production hardware characteristics is a new setting on the **Advanced** pane of the **Simulation parameters** dialog box. This setting, intended for use in modeling, simulating, and generating code for digital systems, allows you to specify the sizes of the data types supported by the system being modeled. Simulink uses this information to automate the choice of data types for signals output by some blocks.

True Color Support

This release allows you to use any color supported by your system as the foreground or background colors of a block diagram. See “Specifying Block Diagram Colors” in the online documentation for more information.

Print Details

This command generates an HTML report detailing the contents of the currently selected model (see “Generating a Model Report” in the online documentation for more information).

Boolean Logic Signals

In previous releases, the `Boolean logic signals` optimization was off by default for new models (see in the online Simulink documentation for a description of this option). In the current release, the optimization is on by default for new models. This change does not affect existing models.

Model Discretizer

The Model Discretizer tool selectively replaces continuous Simulink blocks with discrete equivalents. Discretization is critical in digital controller design for dynamic systems and for hardware in the loop simulations. You can use this tool to prepare continuous models for use with the Real-Time Workshop

Embedded Coder, which supports only discrete blocks. See “Model Discretizer” in the online documentation for more information.

Platform Limitations for HP and IBM

The following are platform limitations for Simulink 5.0 for the HP and IBM platforms that are new limitations, as of Version 5.0.

- The **Parameter** dialog for the Configuration Subsystem Block is not supported on the HP and IBM platforms. Instead, use the `set_param` command to set the block’s parameters.
- The **View Changes** dialog box for modified library links is not supported on the HP and IBM platforms. Instead, select the modified link and execute `ld=get_param(gcb,'LinkData')` to get a structure that lists the parameter differences between the library and local instance of the block. Edit this structure and execute `set_param(gcb,'LinkData',ld)` to apply the changes.
- The GUI interface to the Simulink Debugger is not supported on the HP and IBM platforms. Use the command-line interface instead.
- Model Discretizer is not supported on the HP and IBM platforms.

Note The Release 12 and 12.1 platform limitations for Simulink for the HP and IBM platforms still apply to Release 13. These are listed below.

The following Java-dependent Simulink features, introduced in Simulink 4.1, are *not* available on the HP and IBM platforms.

- Simulink Data Class Designer
- S-Function Builder
- Look-Up Table Editor

Version 4.1 (R12+) Simulink

This table summarizes what's new in V4.1 (R12+):

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

New features and changes introduced in this version are organized by these topics:

Simulink Editor

This section describes enhancements to the Simulink Editor.

Undo Move

In Simulink 4.1, the **Undo** command on the Simulink **Edit** menu restores blocks, annotations, lines, and nodes that have moved to their original locations (see "Undoing a Command" in Using Simulink).

Undo Subsystem Creation

In Simulink 4.1, the **Undo** command on the Simulink **Edit** menu restores blocks that have been grouped into a subsystem to their original level in the model (see “Undoing Subsystem Creation” in Using Simulink).

Autoconnecting Blocks

This version makes connecting blocks significantly easier. To connect a set of source blocks to a target block, simply select the source blocks, hold down the **Ctrl** key and left-click the target block. Simulink draws connecting lines between the source blocks and the destination block, neatly routing lines around intervening blocks. To connect a source block to a set of target blocks, select the target blocks, hold down the **Ctrl** key and left--click the source block. To connect two blocks, select the source block, and left-click the destination block while holding down the **Ctrl** key. Simulink connects as many ports on the two blocks as possible (see “Connecting Blocks”).

Autorouting Signal Lines

Simulink now routes signal lines around intervening blocks when you connect them either interactively (by dragging the connecting lines or using autoconnect) or programmatically via the `add_line` command’s new `'autorouting'` option (see “Autorouting Option Added to add_line Command” on page 109).

Displaying Storage Class on Lines

This version adds an item to the **Format** menu, which toggles the display of (nonAuto) storage class on signal lines.

Save Models in Release 11 Format

This release can save post-Release 11 models in Release 11 format. Simulink 3 (Release 11) can load and run converted models that do not use any post-Release 11 features of Simulink. Simulink 3 can load converted models that use post-Release 11 features but may not be able to simulate the model correctly. Use the **Save as** option from the Simulink **File** menu or the following command to save a model in Release 11 format.

```
slsaveas(SYS)
```

Modeling Enhancements

This section describes enhancements to Simulink dynamic system modeling tools.

Autorouting Option Added to add_line Command

The `add_line` command now optionally routes lines around intervening blocks and annotations. For example, the following command autoroutes a connection between two blocks in the `vdp` model.

```
add_line('vdp','Product/1','Mu/1','autorouting','on')
```

The autorouting option is off by default. See `add_line` in Using Simulink for more information.

S-Function Builder

The S-Function Builder generates an S-function from specifications that you enter in a dialog box. It provides an easy way for you to incorporate existing code into a Simulink model.

add_param, delete_param

With this version, you can add custom parameters to your block diagrams.

```
add_param('modelname','MyParameterName','value')
delete_param('modelname','MyParameterName')
```

You can also use the model handle in place of the model name. See `add_param` and `delete_param` in Using Simulink for more information.

Connection Callbacks

With this version, you can use `set_param` to set callbacks on ports that are triggered by changes in the ports' connectivity. The callback function parameter is named `ConnectionCallback`. When the port's connectivity changes (addition/deletion of line connected to the port, connection of new block to the port, etc.), Simulink invokes the callback function with the port handle as its argument. See "Port Callback Parameters" for more information.

Saving Block User Data in Model Files

This version adds a new block parameter, named `UserDataPersistent`, that is `off` by default. Setting this parameter `on`, e.g.,

```
set_param(block-name,'UserDataPersistent','on')
```

causes Simulink to include a block's user data (i.e., the value of the block's `UserData` parameter) in the model file when you save a model. Simulink encodes the user data as ASCII characters and saves the encoded data in a new section of the model file called `MatData`. This mechanism works with all forms of MATLAB data, including arrays, structures, objects, and Simulink data objects. See "Associating User Data with Blocks" for more information.

Absolute Tolerance Enhancements

This version adds a dialog item for setting the absolute tolerance for each state in the State-Space block, the Transfer Fcn block, and the Zero-Pole block. With this enhancement, you can now specify the absolute tolerance for solving every continuous state in your model.

Block Reduction Enhancements

S-functions may now request that they be eliminated from the compiled model. To do this, call `ssSetBlockReduction (true)` inside the S-function. This is an advanced feature provided for customers writing S-functions who want to optimize the generated code produced for their S-function. Graphical connectivity is now remapped during block reduction, eliminating a source of error during reduction (e.g., a memory reference error used to occur if Simulink eliminated a block connected to a scope). Block reduction is now `on` by default, and a Simulink preference has been added for the option.

Boolean Logic Signals Preference

The Simulink Preferences dialog box now allows you to specify the use of Boolean logic signals by default. See "Working with Simulink Preferences" in *Getting Started with Simulink* for more information.

Subsystem Semantics Demos

Typing `sl_subsys_semantics` at the MATLAB prompt now displays a set of models that illustrate the semantics of various types of subsystem blocks. The demos include formal definitions of function-call subsystems.

Enhanced Engine Model Demos

The top and bottom dead center detection in the `engine` and `enginewc` demo models now use a reset integrator. In previous versions, the models used a triggered subsystem to detect angular position. This method resulted in inefficiencies and a slower, less accurate solution. In addition, self-triggering subsystems are now illegal in Simulink.

Setting Block Sorting Priority on Virtual Subsystems

In Simulink 4.0, it was an error to specify a priority on a virtual subsystem. In Simulink 4.1, you can specify priorities on virtual subsystems.

Using ~ in Filenames on UNIX

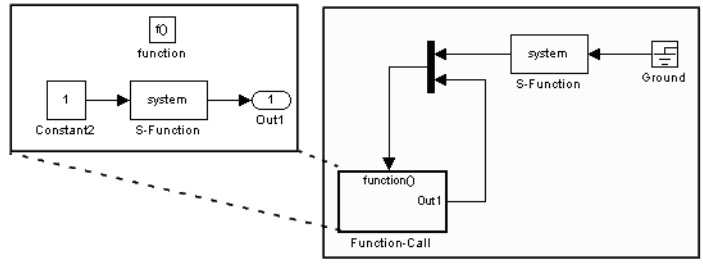
Now all filename fields in Simulink support the mapping of the ~ character in filenames. For example, in a To File block, you can specify `~/outdir/file.mat`. On most systems, this will expand to `/home/$USER/outdir/file.mat`.

Improved Warning About Slow Signals Feeding the Enable Port of an Enabled Subsystem Containing Fast Blocks

In a multitasking environment, deterministic results cannot be guaranteed if a slow signal feeds the enable port of an enabled subsystem that contains fast blocks. In previous versions, Simulink did not issue a warning in some cases where this may occur.

Flagging Function-Call Subsystem Cycles

In previous versions, Simulink allowed you to build models containing function-call-cycles, i.e., function-call subsystems that directly or indirectly call themselves.



Such models cannot be correctly simulated. Accordingly, Simulink now displays an error message when you attempt to run or update a diagram containing function-call cycles.

Simulink Debugger

This section describes enhancements to the Simulink debugger.

Enhancement to Sorted List Display

The Simulink debugger (`sldebug`) sorted list command, `slist`, now displays the names of the S-functions residing inside S-function blocks.

Improved Messages in Accelerated Mode

The `trace`, `break`, `zcbreak`, `nanbreak`, and `minor` commands now indicate that they are disabled when in accelerator mode and you need to switch to normal mode to activate them. The spacing of several messages has been fixed so the text aligns correctly.

Breakpoints on a Function-Call Subsystem

You can now put a break point on a function-call subsystem. Simulink breaks when the subsystem is executed. In Release 12, entering the `quit` command while at a breakpoint within a function-call subsystem wouldn't always quit the debugger. Now the `quit` command ends the debugging session once the initiating (calling) Stateflow chart or S-function finishes executing its time step.

Displaying and Probing Virtual Blocks

The `display` and `probe` commands now work for virtual blocks.

Stepping Stateflow Charts

You can now step execution of a model into a Stateflow chart.

Block Library

This section describes enhancements to the Simulink block libraries.

Unified Pulse Generator

This version merges the Discrete Pulse Generator block into the Pulse Generator block. The combined block has two modes: time-based and sample-based (discrete). Time-based mode varies the step size when a variable step solver is being used to ensure that simulation steps occur at pulse on/off transitions. When a fixed step solver is used, the time-based mode computes a fixed step size that ensures that a simulation step occurs at every pulse transition. The Pulse Generator block also outputs a pulse of any real data type in sample-based as well as time-based mode.

Control Flow Blocks

Simulink 4.1 adds an If block and Switch Case block that can drive conditionally executed subsystems that contain instances of the new Action Port block. Action subsystems are similar to enabled subsystems, except that all blocks must run at the same rate as the If or Switch Case block.

This version also adds a For Iterator block and a While Iterator block. When placed in a subsystem, these blocks cause all of the blocks in the system to run multiple cycles during a time step. The block cycle in a For Iterator subsystem runs a specified number of times. The block cycle in a While Iterator subsystem runs until a specified condition is false. A user can limit execution of a While Iterator subsystem to a specified number of iterations to avoid infinite loops.

The new Assignment block allows a model to assign values to specified elements of a signal.

Bus Creator

Simulink 4.1 adds a Bus Creator block that combines the output of multiple blocks into a single signal bus. A model can use the existing Signal Selector block to extract signals from the bus. The block's dialog box allows you to assign names to signals on the bus or allow the signals to inherit their names from their sources. When you double-click on a signal name in the block dialog, the source block is highlighted. There is no execution overhead in the use of bus creator/bus selector blocks.

Sine Wave Block Enhancements

The Sine Wave block now supports a bias factor that eliminates the need to sum with a Constant block. The Sine Wave block also has a new computational mode. This mode (called sample-based) eliminates the dependence on absolute time.

Enhanced Flip-Flop Blocks

Simulink Extras (`simulink_extras.mdl`) contains a set flip-flop blocks. These blocks now use the new triggered subsystem latching semantics. In addition, the S-R Flip-Flop block now models a physical `NOR` gate (i.e., `S=1`, `R=1 => Q=0`, `Q!=0`, the undefined state).

Additional Data Type Support

The Discrete-Time Integrator and Rounding Function blocks now handle `single` as well as `double` values. The Transport Delay, Unit Delay, Variable Transport Delay, Memory, Merge, and Outport blocks can specify nonzero initial conditions when operating on fixed-point signals.

Simulink Block Library Reorganization

The Simulink Block Library contains a new Subsystems sublibrary. The new library contains most of the new control flow blocks as well as subsystem and subsystem-related blocks that used to reside in the Signals & Systems library. The subsystems in the new library each contain the minimum set of blocks needed to create a functioning subsystem, e.g., an input port and an output port.

Compatibility Considerations. The Simulink Block Library contains a new Subsystems sublibrary. The new library contains most of the new control flow blocks as well as subsystem and subsystem-related blocks that used to reside in the Signals & Systems library. The subsystems in the new library each contain the minimum set of blocks needed to create a functioning subsystem, e.g., an input port and an output port.

Scope Enhancements

The Scope block includes the following enhancements:

- A floating version of the Scope added to the Sinks block library
- Floating Scope saves the signals selected for display in the model file
- The Scope's toolbar buttons for toggling between floating/nonfloating mode, restoring saved axes, locking/unlocking axes, and displaying the **Signal Selector**

S-Functions Sorted Like Built-In Blocks

Compatibility Considerations. When sorting blocks, Simulink now treats S-function blocks the way it treats built-in blocks. This means that S-functions now work correctly in nonvirtual subsystems when there is a direct feedback connection (in Simulink 4.0 and prior, this wasn't the case). It also means that models compile (update diagram) faster. As a side effect, the execution order for S-functions that incorrectly set the direct feedthrough flag differs from that used in previous versions of Simulink. Consequently, models that contain invalid S-functions may produce different answers in this version of Simulink.

Triggered Subsystems

This section describes features and changes to the Simulink triggered subsystems.

Added Latched Triggered Subsystems

Now triggered subsystems enable you to implement software triggering, hardware triggering, or a combination of the two. Software triggering is defined as

```
if (trigger_signal_edge_detected) {
 out(t) = f(in(t));
}
```

Hardware triggering is defined as

```
if (trigger_signal_edge_detected) {
 out(t) = f(in(t-h));   // h == last step size
}
```

Compatibility Considerations. Previous to this version, triggered subsystems provided software triggering and a form of hardware triggering when a cycle involving triggered subsystems existed. Now, you must explicitly specify whether or not you'd like software or hardware triggering. This is done by selecting `'Latch (buffer) input'` on the Inport blocks in a triggered subsystem.

Each input port of a triggered subsystem configures whether or not the input should be latched. A latched input provides the hardware-triggering semantics for that input port. Type `sl_subsys_semantics` at the MATLAB prompt for more information.

Self-Triggering Subsystems Are No Longer Allowed

Compatibility Considerations. Before this version, you could define the output of a triggered subsystem to directly feed back into the trigger port of the subsystem (with potentially other additive signals). This resulted in an implicit delay. Now you must explicitly define the delay by inserting a memory block.

Running Simulink 4.1 Models in Simulink 4.0

Simulink 4.0 can run models created or saved by Simulink 4.1, with the provisions outlined in the following.

Compatibility Considerations

Simulink 4.0 can run models created or saved by Simulink 4.1 as long as the models do not use features introduced in the new version, including new block types and block parameters. In particular, you should not attempt to

use Simulink 4.0 to simulate or even open models that use the new Simulink control flow blocks. Opening such models cause Simulink 4.0 to crash.

Direct Feedthrough Compensation Deprecated

If an S-function needs the current value of its input to compute its output, it must set its direct feedthrough flag to true.

Compatibility Considerations

Previously, if a direct feedthrough S-function failed to do this, Simulink tried to provide a valid signal to the S-function's `mdlOutput` (M-file `flag=3`) or `mdlGetTimeOfNextVarHit` (M-file `flag=4`) methods. This special compensation mode for S-functions was flawed. For this reason, the current version deprecates the mode, though making it available as an option. In this version, by default, if an S-function sets its direct feedthrough flag to false during initialization, Simulink sets the S-function's input signal to `NULL` (or a `NaN` signal for M-file S-functions) during the `mdlOutput` or `mdlGetTimeOfNextVarHit` methods. Thus, in this version, models with S-function(s) may produce segmentation violations. See *`matlabroot`*`/simulink/src/sfuntmpl_directfeed.txt` for more information.

Improved Invalid Model Configuration Diagnostics

This version of Simulink does a better job of detecting and flagging invalid modeling constructs in Simulink models. The changes include:

- Direct feedthrough compensation no longer occurs by default for S-functions (see "Direct Feedthrough Compensation Deprecated" on page 117).
- S-functions are now sorted like built-in blocks (see "S-Functions Sorted Like Built-In Blocks" on page 115).
- Simulink no longer inserts implicit latches in triggered subsystems that directly or indirectly trigger themselves (see "Self-Triggering Subsystems Are No Longer Allowed" on page 116, above). Instead it signals an error when it detects a triggered subsystem loop with unlatched inputs. To avoid the error, you must select the **Latch** option on the triggered subsystem's input ports.

- Simulink now signals an error when it detects invalid configurations of function-call subsystems. See the Subsystem Examples block in the Subsystems library for examples of illegal modeling constructs involving function-call subsystems. You can disable this diagnostic by setting the `Invalid FcnCall Connection` parameter on the **Diagnostics** pane of the **Simulation Parameters** dialog box to `none` or `warning`.

Compatibility Considerations

Consequently models that ran in previous versions of Simulink (sometimes producing incorrect results) may not run in the current release.

Bug Fixes

This section lists fixes to bugs that occurred in the previous version of Simulink.

Variable sample time S-functions

Simulink no longer crashes when an S-function with variable sample time is placed in an atomic subsystem.

Bus selector detection of duplicated names

A bug related to the detection of a duplicated name in a bus that was feeding a Bus Selector block was fixed.

Optimize block memory use

In Simulink 4.0, the Continuous and Discrete Transfer Function blocks and the Discrete Filter block used more memory than they needed to, particularly for the case of many poles. They now use an optimal amount of memory.

Miscellaneous fixes to the model loader

Miscellaneous bug fixes have been performed on the model loader:

- The loader and saver now retain any comment lines (i.e., lines that begin with #) that are found at the top of the model file.
- The loader does not crash on Windows NT when file sizes are integer multiples of 4096.

- The loader does not hang on corrupt models in which blocks with duplicate names are found.

Profiler fixes

The Simulink profiler now saves its files in the temporary directory. See the MATLAB command `tempdir`. The help was also updated.

Chirp block fix

The Chirp block now sweeps through frequencies correctly from the initial frequency at the simulation start time to the target frequency at the target time.

Function-call subsystem bug fixes

This version fixes several bugs related to the execution orders of function-call subsystems.

Sorting bug fix

Previous versions incorrectly computed the direct feedthrough setting for nonvirtual subsystems in triggered/function-call subsystems. This resulted in incorrect execution (sorting) orders. Now all nonvirtual subsystems within triggered subsystems have their direct feedthrough (needs input) flags set for all input ports. This is needed because a nonvirtual subsystem with a triggered sample time executes both its output and update methods together within the context of the model's output method.

Fixed handling of grounded/unconnected inputs feeding certain blocks

Simulink 4.0 incorrectly handled grounded or unconnected inputs to level-1 and level-2 S-functions requiring contiguous inputs and to some Matrix blocks. This has been fixed in Simulink 4.1.

Version 4.0 (R12) Simulink

This table summarizes what's new in V4.0 (R12):

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

New features and changes introduced in this version are organized by these topics:

Simulink Editor

This section describes enhancements to the Simulink Editor.

Preferences

The Simulink **Preferences** dialog box allows you to specify default settings for many options (see "Working with Simulink Preferences" in *Getting Started with Simulink*).

Text Alignment

Simulink 4.0 allows you to choose various alignments for annotation text. To choose an alignment for an annotation, select the annotation and then select

Text Alignment from the editor menu bar or context (right-click) menu (see "Annotating Diagrams").

UNIX Context Menus

The UNIX version of Simulink 4.0 now has context menus for block diagrams. Click the right button on your mouse to display the menu.

Library Link Enhancements

Simulink 4.0 optionally displays an arrow in each block that represents a library link in a model. Simulink 4.0 also allows you to modify a link in a model and propagate the changes back to the library (see "Modifying a Linked Subsystem" in Using Simulink).

Note Simulink displays "Parameterized Link" on the parameter dialog box of a masked subsystem whose parameters differ from the library reference block to which the masked subsystem is linked. This feature, which is not documented in Using Simulink, allows you to determine quickly whether a library link differs from its reference.

Find Dialog Box

The **Find** dialog box enables you to search Simulink models and Stateflow charts for objects that satisfy specified search criteria. You can use the dialog box to find annotations, blocks, signals, states, state transitions, etc. To invoke the **Find** dialog, select **Find** from the Simulink **Edit** menu (see "The Finder").

Model Browser

The Model Browser's toolbar includes the following new buttons:

- Show Library Links

  Shows library links as nodes in the browser tree.

- Look Under Masks

  Shows the contents of masked blocks as nodes in the browser tree.

Single Window Mode

Simulink now provides two modes for opening subsystems. In multiwindow mode, Simulink opens each subsystem in a new window. In single-window mode, Simulink closes the parent and opens the subsystem (see "Window Reuse" in *Using Simulink*).

Keyboard Navigation

Simulink 4.0 provides the following new keyboard shortcuts.

Key	Action
Tab	Selects the next block in the block diagram.
Shift+Tab	Selects the previous block in the block diagram.
Ctrl+Tab	Cycles between the browser tree pane and the diagram pane when the model browser is enabled.
Enter	Opens the currently selected subsystem.
Esc	Opens the parent of the current subsystem.

Enhanced Library Browser

The Library Browser incorporates the following new features:

- Blocks no longer appear as browser tree nodes. Instead, they appear as icons in the preview pane.
- The preview pane has moved from beneath the library tree pane to beside the tree pane. You can create instances of blocks displayed in the preview pane by dragging them from the preview pane and dropping them in a model.
- Splitter bars now divide the browser's panes, allowing the panes to be independently resized.

- Double-clicking a block's icon opens the block's parameter dialog box with all fields disabled. This allows you to inspect, but not modify, a library block's parameters.
- Double-clicking a library block opens the library in the preview pane.
- You can now insert a block in the topmost model on your screen by right-clicking the block in the preview pane and selecting **Insert in...** from the context menu that appears. If no model is open or the topmost model is a locked library, the Library Browser offers to create a model in which to insert the block.
- The browser now contains a menu with **File**, **Edit**, and **Help** options.
- The block help text pane has moved from the bottom of the Library Browser to the top.
- Selecting **Find** from the Library Browser's **Edit** menu displays a modeless **Find** dialog box.
- The browser's search feature is much faster and supports regular expressions.

Help Menus

Simulink 4.0 adds a Help menu to the menu bar on model and library windows. The help item on a block context menu displays a help page for the block. The help item on the model context menu displays the first page of the Using Simulink book.

Modeling Enhancements

Hierarchical Variable Scoping

This release extends the ability of Simulink to resolve references to variables in masked subsystems. Previously Simulink could resolve references only to variables in a block's local workspace. With this release, Simulink will resolve references to variables located anywhere within the workspace hierarchy containing the block (see "The Mask Workspace" in *Using Simulink*).

Note In some cases, hierarchical scoping will cause some models to behave differently in the current release than in previous releases of Simulink.

Matrix Signals

Many Simulink blocks can now accept or output matrix signals. A matrix signal is a two-dimensional array of signal elements represented by a matrix. Each matrix element represents the value of the corresponding signal element at the current time step. In addition to matrix signals, Simulink also supports scalar (dimensionless) signals and vector signals (one-dimensional arrays of signals). Simulink can optionally thicken (select **Wide Lines** from the **Format** menu) and display the dimensions of lines (select **Line Dimensions** from the **Format** menu) that carry vector or matrix signals. When you select the **Line Dimensions** option, Simulink displays a label of the form `[r x c]` above a matrix signal line, where `r` is the number of rows and `c` is the number of columns. For example, the label `[2 x 3]` indicates that the line carries a two-row by three-column matrix signal.

You can use Simulink source blocks, such as a Sine Wave or a Constant block, to generate matrix signals. For example, to create a time-invariant matrix signal, insert a Constant block in your model and set its Constant Value parameter to any MATLAB expression that evaluates to a matrix, e.g., `[1 2; 3 4]`, that represents the desired signal. See "Working with Signals" in Using Simulink for more information.

Simulink Data Objects

Simulink data objects allow a model to capture user-defined information about parameters and signals, such as minimum and maximum values, units, and so on (see "Working with Data Objects" in Using Simulink).

Block Execution Order

Simulink now optionally displays the execution order of each block on the model's block diagram (see "Displaying Block Execution Order" in Using Simulink).

Simulink Debugger

This section describes enhancements to the Simulink debugger.

GUI Debugger Interface

Simulink 4.0 introduces a graphical user interface (GUI) for the Simulink Debugger. For more information, see "Simulink Debugger" in the online help for Simulink (see "Simulink Debugger" in Using Simulink).

Block Library

This section describes enhancements to the Simulink block libraries.

Product Block

The Product block now supports both element-by-element and matrix multiplication and inversion of inputs. The block's parameter dialog includes a new **Multiplication** parameter that allows you to specify whether the block should multiply or invert inputs element-by-element or matrix-by-matrix.

Gain Block

The Gain block now supports matrix as well as element-wise multiplication of the input signal by a gain factor. Both input signals and gain factors can be matrices. The block's parameter dialog includes a new **Multiplication** parameter that allows you to choose the following options:

- `K.*u` (element-wise product)
- `K*u` (matrix product with the gain as the left operand)
- `u*K` (matrix product with the gain as the right operand)

Math Function Block

The Math Function block adds two new matrix-specific functions: transpose and Hermitian. The first function outputs the transpose of the input matrix. The second function outputs the complex conjugate transpose (Hermitian) of the input matrix.

Reshape Block

Simulink 4.0 introduces the Reshape block, which changes the dimensionality of its input signals, based on an **Output dimensionality** parameter that you specify. For example, the block can change an n-element vector to a 1-by-N or N-by-1 matrix signal and vice versa. You can find the Reshape block in the Simulink Signals & Systems library.

Multiplexing Matrix Signals

The Simulink Mux, Demux, and Bus Selector blocks have been enhanced to support multiplexing of matrix signals.

Function Call Iteration Parameter

Simulink 4.0 adds a **Number of iterations** parameter to the Function Call Generator block. This parameter allows you to specify the number of times the target block is called per time step.

Probing Signal Dimensionality

The Probe block now optionally outputs the dimensionality of the signal connected to its input.

Configurable Subsystem

The Configurable Subsystem block has been reimplemented to make it easier to use. The configurable subsystem block now has a **Blocks** menu that allows you to choose which block the subsystem represents. To display the menu, select the configurable subsystem and then **Blocks** from the Simulink editor's **Edit** or context (right click) menu.

Look-Up Table Blocks

This release provides four new Look-Up Table (LUT) blocks.

- Direct Look-Up Table (n-D)
- Look-Up Table (n-D)
- PreLookup Index Search (Obsolete)
- Interpolation (n-D) Using PreLookup (Obsolete)

The blocks reside in the Simulink Functions and Tables block library.

Polynomial Block

The Polynomial block outputs a polynomial function of its input. The block resides in the Simulink Functions and Tables block library.

Signal Specification

The Signal Specification block allows you to specify the attributes that the input signal must satisfy. If the input signal does not meet the specification, the block generates an error.

ADA S-Functions

Simulink now supports S-functions coded in ADA. See "Creating Ada S-Functions" in *Writing S-Functions* for more information.

Bitwise Logical Operator Block

The Bitwise Operator block is a new block that logically masks, inverts, or shifts the bits of an unsigned integer signal. See the online Simulink documentation for details.

Atomic Subsystems

Simulink 4.0 allows you to designate subsystems as *atomic* as opposed to *virtual*. An *atomic subsystem* is a true subsystem. When simulating a model, Simulink executes all blocks contained by an atomic subsystem block before executing the next block of the containing model (or atomic subsystem).

By declaring a subsystem atomic, you guarantee that Simulink completes execution of the subsystem before executing any other blocks at the same level in the model hierarchy. See "Atomic Subsystems" in *Using Simulink* for more information.

Note Conditionally executed subsystems are inherently atomic. Simulink does not allow you to specify them as atomic or virtual.

SB2SL

SB2SL Extends Code Generation Support

SB2SL, which is included as part of Simulink, allows you to translate SystemBuild SuperBlocks to Simulink models.

For Release 12, SB2SL 2.1 has been enhanced to provide more complete support for use with the Real-Time Workshop. If you use the Real-Time Workshop 4.0 to generate code for models you have converted from SystemBuild to Simulink (using SB2SL), then code is generated for most translated blocks in the model.

The blocks that do *not* support code generation through the Real-Time Workshop 4.0 are:

- ConditionBlock
- Decoder
- Encoder
- GainScheduler
- Interp Table (Archive library)
- ShiftRegister

Note SB2SL 2.1 also includes a number of important bug fixes.

Port Name Property

In the current release, a port's name property refers to the port's (and line's) name, which, in the current release, can differ from the line's label.

Compatibility Considerations

In previous releases, the name property of ports and lines referred to the label of the line connected to the port. If you need to get the line's label, invoke

```
get_param(p, 'label')
```

where `p` is the handle of the port.

Compatibility and Limitations Summary for Simulink

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

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version V6.6.1 (R2007a+)	None
V6.6 (R2007a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "GNU Compiler Upgrade" on page 11 • "Change to Simulink.ModelAdvisor.getModelAdvisor Method" on page 14 • "Legacy Code Tool Enhancements" on page 16 • "Using & and | Operators in Embedded MATLAB Function Blocks" on page 21 • "Calling get Function from Embedded MATLAB Function Blocks" on page 22 • "Default for Signal Resolution Parameter Has Changed" on page 24 • "Port Parameter Evaluation Has Changed" on page 27 • "Referencing Configuration Sets" on page 25 • "Change to PaperPositionMode Parameter" on page 29 • "Change in Version 6.5 (R2006b) Introduced Incompatibility" on page 30

Version (Release)	New Features and Changes with Version Compatibility Impact
V6.5 (R2006b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Enhanced Lookup Table Blocks" on page 32 • "Parameter Objects Can Now Be Used to Specify Model Configuration Parameters" on page 36 • "New Requirement for Calling MATLAB Functions from Embedded MATLAB" on page 41 • "Type and Size Mismatch of Values Returned from MATLAB Functions Generates Error" on page 42 • "Changes to Integrator Block's Level Reset Options" on page 37 • "Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error" on page 37 • "Embedded MATLAB Function Blocks Cannot Output Character Data" on page 42
V6.4.1 (R2006a+)	None

Version (Release)	New Features and Changes with Version Compatibility Impact
V6.4 (R2006a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Range-Checking for Parameter and Signal Object Values" on page 45 • "Concatenate Block" on page 46 • "Built-in Block's Initial Appearance Reflects Parameter Settings" on page 47 • "Setting FIMATH Cast Before Sum to False No Longer Supported in Embedded MATLAB Function Blocks" on page 53 • "Type Mismatch of Scalar Output Data in Embedded MATLAB Function Blocks Generates Error" on page 54 • "Implicit Parameter Type Conversions No Longer Supported in Embedded MATLAB Function Blocks" on page 54
V6.3 (R14SP3)	See the **Compatibility Considerations** and/or **Limitations** subheading for each of these changes: • "Model Referencing" on page 56 • "MEX-Files on Windows" on page 64 • "Fixed-Point Functions No Longer Supported for Use in Signal Objects" on page 64 • "Parameter Object Expressions No Longer Supported in Dialog Boxes" on page 64
V6.2 (R14SP2)	See the **Compatibility Considerations** and/or **Limitations** subheadings for each of these changes: • "Rootlevel Input Ports" on page 66

Version (Release)	New Features and Changes with Version Compatibility Impact
V6.1 (R14SP1)	See the **Compatibility Considerations** for this change: • "Model Load Warnings" on page 69
V6.0 (R14)	See the **Compatibility Considerations** and **Limitations** subheadings for each of these changes: • "Model Referencing" on page 71 • "MATLAB Data Type Conversions" on page 80 • "Signal Object Resolution Changes" on page 80 • "Loading Models Containing Non-ASCII Characters" on page 81 • "Change in Sample Time Behavior of Unary Minus Block" on page 82 • "Initial Output of Conditionally Executed Subsystems" on page 82 • "Execution Context Default Changes" on page 82 • "Internal Signal Structures Revamped" on page 78
V5.0.1 (R13.0.1)	See the **Compatibility Considerations** subheading for this change: • "Tunable Parameters for Unified Fixed-Point Blocks" on page 94
V5.0 (R13)	See the **Compatibility Considerations** subheadings for each of these changes: • "Enhanced Diagnostic Viewer" on page 103 • "Enhanced Mask Editor" on page 104 • "Model Discretizer" on page 105

Version (Release)	New Features and Changes with Version Compatibility Impact
V4.1 (R12+)	See the **Compatibility Considerations** subheadings for each of these changes: • "Simulink Block Library Reorganization" on page 114 • "S-Functions Sorted Like Built-In Blocks" on page 115 • "Added Latched Triggered Subsystems" on page 115 • "Self-Triggering Subsystems Are No Longer Allowed" on page 116 • "Running Simulink 4.1 Models in Simulink 4.0" on page 116 • "Direct Feedthrough Compensation Deprecated" on page 117 • "Improved Invalid Model Configuration Diagnostics" on page 117
V4.0 (R12)	See the **Compatibility Considerations** for this change: • "Port Name Property" on page 129