Simulink® Release Notes

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink® Release Notes

Trademarks

Patents

Summary by Version

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

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V7.5 (R2010a)	Yes Details	Yes Summary	Bug Reports Includes fixes	Printable Release Notes: PDF Current product documentation
V7.4 (R2009b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V7.3 (R2009a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V7.2 (R2008b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V7.1.1 (R2008a+)	No	No	Bug Reports Includes fixes	No
V7.1 (R2008a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V7.0.1 (R2007b+)	No	No	Bug Reports Includes fixes	No
V7.0 (R2007b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V6.6.1 (R2007a+)	No	No	Bug Reports Includes fixes	No
V6.6 (R2007a)	Yes Details	Yes Summary	Bug Reports Includes fixes	No
V6.5 (R2006b)	Yes Details	Yes Summary	Bug Reports Includes fixes	No

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
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
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

Using Release Notes

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

- New features

- Changes

- Potential impact on your existing files and practices

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

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

What Is in the Release Notes

New Features and Changes

- New functionality

- Changes to existing functionality

Version Compatibility Considerations

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

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

Fixed Bugs and Known Problems

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

Access Bug Reports using your MathWorks Account.

Version 7.5 (R2010a) Simulink Software

This table summarizes what's new in V7.5 (R2010a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports Includes fixes	Printable Release Notes: PDF Current product documentation

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

- "Simulation Performance" on page 5
- "Component-Based Modeling" on page 6
- "Embedded MATLAB Function Blocks" on page 8
- "Simulink Data Management" on page 10
- "Simulink Signal Management" on page 14
- "Block Enhancements" on page 17
- "User Interface Enhancements" on page 30
- "S-Functions" on page 34
- "Documentation Improvements" on page 35

Simulation Performance

Computation of Sparse and Analytical Jacobian for Implicit Simulink Solvers

The implicit Simulink solvers now support numerical and analytical methods for computing the Jacobian matrix in one of the following representations: sparse perturbation, full perturbation, sparse analytical, and full analytical.

The sparse methods attempt to improve performance by taking advantage of sparsity information associated with the Jacobian matrix. Similarly, the analytical methods attempt to improve performance by computing the Jacobian using analytical equations rather than the perturbation equations.

Since the applicability of these representations is highly model dependent, an `auto` option directs Simulink to use a heuristic to choose an appropriate representation for your model. In the case of a model that has a large number of states and for which the Jacobian is computed in sparse analytical form, the performance improvement may be substantial. In general, the performance improvement achieved varies from model to model.

Sparse Perturbation Support for RSim and Rapid Accelerator Mode

For implicit Simulink solvers, the numerical sparse perturbation method for solving the Jacobian supports both RSim and Rapid Accelerator mode.

Increased Accuracy in Detecting Zero-Crossing Events

The zero-crossing bracketing algorithm now uses a smaller tolerance for defining the interval in which an event occurs. The resulting increased accuracy of locating an event means that existing models may exhibit slightly different numerical results.

Saving Code Generated by Accelerating Models to slprj Folder

In Accelerator mode and in Rapid Accelerator mode, a build has historically resulted in the creation of generated code, respectfully, in the *modelname*_accel_rtw and the *modelname*_raccel_rtw folders in the current working folder. However, in order to be more consistent with other builds, in R2010a and future releases, these files will be created in the slprj/accel/*modelname* and the slprj/raccel/*modelname* folders.

Component-Based Modeling

For Each Subsystem Block

The For Each Subsystem block is very useful for modeling scenarios where you need to repeat the same algorithm on individual elements (or submatrices)

of an input signal. The set of blocks within the subsystem represent the algorithm that is to be applied to a single element (or submatrix) of the original signal. You can configure the inputs of the subsystem to decompose the corresponding inputs into elements (or submatrices), and configure the outputs to suitably concatenate the processed results. Additionally, each block that has states inside this subsystem maintains separate sets of states for each element or submatrix it processes. Consequently, the operation of this subsystem is akin to copying the contents of the subsystem as many times as the number of elements in the original input signal, and then processing each element through its respective subsystem copy.

An additional benefit of this subsystem is that it may be utilized to improve code reuse in Real-Time Workshop generated code for certain models. Consider a model containing two reusable Atomic Subsystems with the same scalar algorithm applied to each element of the signal. If the input signal dimensions for these subsystems are different, you will find that two distinct functions are produced in the code generated by Real-Time Workshop for this model. Now, if you were to convert the two subsystems to For Each Subsystems such that the contents of each processes a single scalar element, then you will find that the two subsystems produce a single function in the code generated by Real-Time Workshop. This function is parameterized by the number of elements to be processed.

New Function-Call Split Block

A new Function-Call Split block allows you to branch periodic and asynchronous function-call signals and connect them to multiple function-call subsystems (or models). These subsystems (or models) are guaranteed to execute in the order determined by their data dependencies. If a deterministic order cannot be computed, the model produces an error.

To test the validity of your function-call connections, use the Model Advisor diagnostic, **Check usage of function-call connections**. This diagnostic determines if:

- **Configurations > Diagnostics > Connectivity > Invalid function-call connection** is set to `error`

- **Configuration Parameters > Diagnostics > Connectivity > Context-dependent inputs** is set to `Enable All`

Trigger Port Enhancements

You can use trigger ports, which you define with a Trigger block, in new ways:

- Place edge-based (rising, falling, or either), as well as function-call, trigger ports at the root level of a model. Before R2010a, to place a trigger port in a root-level model, you had to set the trigger type to `function-call`.

- Place triggered ports in models referenced by a Model block. See "Defining Triggered Models".

- Lock down the data type, port dimension, and trigger signal sample time. To specify these values, use the new **Signal Attributes** pane of the Block Parameters dialog box of the Trigger block. Specifying these attributes is useful for unit testing and running standalone simulation of a subsystem or referenced model that has an edge-based trigger port. See "Simulating a Triggered Model".

Compatibility Considerations. When you add a trigger port to a root-level model, if you use the **File > Save As** option to specify a release before R2010a, Simulink replaces the trigger port with an empty subsystem.

Model Reference Support for Custom Code

Select the new `ModelRefHonorSimTargetCustomCode` model parameter to have SIM target Accelerator code generation include Stateflow® and Embedded MATLAB® custom code for a referenced model. The default setting for this parameter is `off`.

Embedded MATLAB Function Blocks

New Ability to Use Global Data

Embedded MATLAB Function blocks are now able to use global data within a Simulink model and across multiple models.

This feature provides these benefits:

- Allows you to share data between Embedded MATLAB Function blocks and other Simulink blocks without introducing additional input and output

wires in your model. This reduces unnecessary clutter and improves the readability of your model.

- Provides a means of scoping the visibility of data within your model.

For more information, see "Using Global Data with the Embedded MATLAB Function Block" in the Simulink documentation.

Support for Logical Indexing

Embedded MATLAB Function blocks now support logical indexing when variable sizing is enabled. Embedded MATLAB supports variable-size data by default for MEX and C/C++ code generation.

For more information about logical indexing, see "Using Logicals in Array Indexing" in the MATLAB documentation.

Support for Variable-Size Matrices in Buses

Embedded MATLAB Function blocks now support Simulink buses containing variable-size matrices as inputs and outputs.

Support for Tunable Structure Parameters

Embedded MATLAB Function blocks now support tunable structure parameters. See "Working with Structure Parameters in Embedded MATLAB Function Blocks".

Check Box for 'Treat as atomic unit' Now Always Selected

In existing models, simulation and code generation for Embedded MATLAB Function blocks always behave as if the **Treat as atomic unit** check box in the Subsystem Parameters dialog box is selected. Starting in R2010a, this check box is always selected for consistency with existing behavior.

Simulink Data Management

New Function Finds Variables Used by Models and Blocks

The new `Simulink.findVars` function returns information about workspace variables and their usage. For example, you can use `Simulink.findVars`, sometimes in conjunction with other Simulink functions, to:

- Identify all workspace variables used by a model or block

- Identify any workspace variables unused by a model or block

- Search a model for all places where a specified variable is referenced

- Subdivide a model, including only necessary variables with each model

See `Simulink.findVars` and the other Simulink functions referenced on that page for more information.

MATLAB Structures as Tunable Structure Parameters

You can create a MATLAB structure that groups base workspace variables into a hierarchy, and dereference the structure fields to provide values in Simulink block parameter expressions. This technique reduces base workspace clutter and allows related workspace variables to be conveniently grouped. However, in previous releases you could not use a MATLAB structure as a masked subsystem or a model reference argument, and no value given by a MATLAB structure field could be tuned. These restrictions limited the usefulness of MATLAB structures for grouping variables used in block parameter expressions.

In R2010a, these restrictions no longer apply to MATLAB structures that contain only numeric data. You can use a numeric structure, or any substructure within it, as a masked subsystem or a model reference argument, thereby passing all values in the structure with a single argument. You can also control MATLAB structure tunability using the same techniques that control MATLAB variable tunability. In R2010a, all values in a given structure must be either tunable or nontunable. See "Using Structure Parameters" for more information.

Simulink.saveVars Documentation Added

The Simulink.saveVars function was added in R2009b but was incompletely documented. See "New Function Exports Workspace Variables and Values" on page 44 for more information.

Custom Floating-Point Types No Longer Supported

Custom floating-point types, float(TotalBits, ExpBits), are no longer supported.

Compatibility Considerations. If you have code that uses custom floating-point types, modify this code using one of these methods:

- Replace calls to float(TotalBits, ExpBits) with calls to fixdt('double') or fixdt('single') as appropriate.

- Create your own custom float replacement function.

  Write a MATLAB function custom_float_user_replacement and place the file on your MATLAB path. This function must take TotalBits and ExpBits as input arguments and return a supported numerictype object, such as fixdt('double') or fixdt('single').

  For example,

  ```
  function DataType = custom_float_user_replacement( TotalBits, ExpBits

  if (TotalBits <= 32) && (ExpBits <= 8)
      DataType = numerictype('single');
  else
      DataType = numerictype('double');
  end
  ```

  In R2010a and future releases, if the file custom_float_user_replacement.m is on your MATLAB path, calls to float(TotalBits, ExpBits) automatically call custom_float_user_replacement(TotalBits, ExpBits).

Data Store Logging

You can log the values of a local or global data store data variable for all the steps in a simulation. Data store logging is useful for:

- Model debugging – view the order of all data store writes

- Confirming a model modification – use the logged data to establish a baseline for comparing results to identify the impact of a model modification

To log a local data store that you create with a Data Store Memory block:

- Use the new **Logging** pane of the Block Parameters dialog box for the Data Store Memory block.

- Enable data store logging with the new **Configuration Parameters > Data Import/Export > Data stores** parameter.

To log a data store defined by a `Simulink.Signal` object, from the MATLAB command line, set `DataLogging` (which is a property of the `LoggingInfo` property of `Simulink.Signal`) to `1`.

For details, see "Logging Data Stores". To see an example of logging a global data store, run the `sldemo_mdlref_dsm` demo.

Models with No States Now Return Empty Variables

Simulink creates empty variables for state logging (`xout`) or final state logging (`xfinal`), if both of these conditions apply:

- A model has no states.

- In the **Configuration Parameters > Data Import/Export** pane, you enable the **States**, **Final States**, or both parameters (the default is `off`).

Compatibility Considerations. If you configure your model to return empty variables when it has no states, then a possible result is that Simulink creates more variables than it did in previous releases.

Using model variants, running different models in batch mode, tuning models, or reconfiguring models can produce unexpected results based on the state values. For example, if you simulate a model that produces a state value, and then run a model variant that produces no state, Simulink overwrites the state value with an empty variable. If your model depends on the first state value not being overwritten if no state is returned in a subsequent simulation (which was the case in previous releases), then you get unexpected results.

To File Block Enhancements

The To File block now supports:

- Saving very large data sets that may be too large to fit in RAM

- Saving logged data up until the point of a premature ending of simulation processing. Previously, if the simulation processing did not complete, then To File did not store any logged data for that simulation.

- A new **Save format** parameter to control whether the block uses Timeseries or array format for data.

  - Use Timeseries format for writing multidimensional, real, or complex inputs, with different data types, (for example, built-in data types, including Boolean; enumerated (enum) data and fixed-point data with a word length of up to 32 bits.

  - Use Array format only for one-dimensional, double, noncomplex inputs. Time values are saved in the first row. Additional rows correspond to input elements.

Compatibility Considerations. For data saved using MAT file versions prior to 7.3, the From File block can only load two-dimensional arrays consisting of one-dimensional, double, noncomplex samples. To load data of any other type, complexity, or dimension, use a Timeseries object and save the file using MAT file version 7.3 or later. For example, use 'save file_name -v7.3 timeseries_object':

```
save file_name -v7.3 timeseries_object
```

From File Block Enhancements

The From File block now supports:

- Incremental loading of very large data sets that may be too large to fit in RAM

- Built-in data types, including Boolean

- Fixed-point data with a word length of up to 32 bits

- Complex data

- Multidimensional data

Root Inport Support for Fixed-Point Data Contained in a Structure

You can now use a root (top-level) Inport block to supply fixed-point data that is contained in a structure.

In releases before R2010a, you had to use a Simulink.Timeseries object instead of a structure.

Simulink Signal Management

Enhanced Support for Proper Use of Bus Signals

To improve model reliability and robustness, avoid mixing Mux blocks and bus signals. To help you use Mux blocks and bus signals properly, R2010a adds these enhancements:

- When Simulink detects Mux block and bus signal mixtures, the "Mux blocks used to create bus signals" diagnostic now generates:
  - A warning when all the following conditions apply:
    - You load a model created in a release before R2010a.
    - The diagnostic is set to 'None'.
    - Simulink detects improper Mux block usage.
  - An error for new models
- Two new diagnostics in the **Configuration Parameters > Diagnostics > Connectivity** pane:
  - The "Non-bus signals treated as bus signals" diagnostic detects when Simulink implicitly converts a non-bus signal to a bus signal to support connecting the signal to a Bus Assignment or Bus Selector block.
  - The "Repair bus selections" diagnostic repairs broken selections in the Bus Selector and Bus Assignment block parameters dialog boxes that are due to upstream bus hierarchy changes.

Compatibility Considerations. In R2010a, if you load a model created in a prior release, you might get warning messages that you did not get before. To avoid getting Mux block-related warnings for existing models that you want to load in R2010a, use the `slreplace_mux` function to substitute Bus Creator blocks for any Mux blocks used to create buses signals.

Bus Initialization

In releases before R2010a:

- For *virtual* buses, you could specify a non-zero scalar or vector initial condition (IC) value that applies to all elements of the bus. You could use a vector value only if all bus elements use the same data type.

- For *nonvirtual* buses, the only value you could specify was zero.

In R2010a, you can create a MATLAB structure for an IC. You can:

- Specify ICs for all or a subset of the bus elements.

- Use the new `Simulink.Bus.createMATLABStruct` helper method to create a full IC structure.

- Use the new Model Advisor Simulink check, **Check for partial structure parameter usage with bus signals**, to detect when structure parameters are not consistent in shape with the associated bus signal.

Using IC structures helps you to:

- Specify nonzero initial conditions

- Specify initial conditions for mixed-dimension signals

- Apply a different IC for each signal in the bus

- Specify ICs for a subset of signals in a bus without specifying ICs for all the signals

- Use the same ICs for multiple blocks, signals, or models

For information about creating and using initial condition structures, see "Specifying Initial Conditions for Bus Signals".

S-Functions for Working with Buses

The following S-functions provide a programmatic interface for working with buses:

S-function	Description
ssGetBusElementComplexSignal	Get the signal complexity for a bus element.
ssGetBusElementDataType	Get the data type identifier for a bus element.
ssGetBusElementDimensions	Get the dimensions of a bus element.
ssGetBusElementName	Get the name of a bus element.
ssGetBusElementNumDimensions	Get the number of dimensions for a bus element.
ssGetBusElementOffset	Get the offset from the start of the bus data type to a bus element.
ssGetNumBusElements	Get the number of elements in a bus signal.
ssGetSFcnParamName	Get the value of a block parameter for an S-function block.
ssIsDataTypeABus	Determine whether a data type identifier represents a bus signal.
ssRegisterTypeFromParameter	Register a data type that a parameter in the Simulink data type table specifies.
ssSetBusInputAsStruct	Specify whether to convert the input bus signal for an S-function from virtual to nonvirtual.
ssSetBusOutputAsStruct	Specify whether the output bus signal from an S-function must be virtual or nonvirtual.
ssSetBusOutputObjectName	Specify the name of the bus object that defines the structure and type of the output bus signal.

Command Line API for Accessing Information About Bus Signals

You can use two new signal property parameters to get information about the type and hierarchy of a signal programmatically:

- CompiledBusType

  - Returns information about whether the signal connected to a port is a bus, and if so, whether it is a virtual or nonvirtual bus

- SignalHierarchy

  - Returns the signal name of the signal. If the signal is a bus, the parameter also returns the hierarchy and names of the bus signal.

See "Model Parameters" and "Getting Information about Buses".

Signal Name Propagation for Bus Selector Block

The new SignalNameFromLabel port parameter supports signal name propagation for Bus Creator block input signals whenever you change the name of an input signal programmatically. You can set this parameter with the set_param command, specifying either a port or line handle and the signal name to propagate.

See "Model Parameters".

Block Enhancements

New Square Root Block

You can use the new Sqrt block to perform square-root calculations. This block includes the following functions:

Function	Icon
sqrt	$\sqrt{u}$
signedSqrt	$\pm\sqrt{\lvert u\rvert}$
rSqrt	$1/\sqrt{u}$

Compatibility Considerations. The sqrt and 1/sqrt functions no longer appear in the Math Function block. For backward compatibility, models with a Math Function block that uses one of these two functions continue to work. However, consider running the slupdate function on your model. slupdate replaces any Math Function block that uses sqrt or 1/sqrt with an equivalent Sqrt block that ensures the same behavior.

New Second-Order Integrator Block

You can use the new Second-Order Integrator block to model second-order systems that have bounds on their states. This block is useful for modeling physical systems, for example, systems that use Newton's Second Law and have constraints on their motion.

Benefits of using this block include:

- Highly accurate results
- Efficient detection of zero crossings
- Prevention of direct feedthrough and algebraic loops

New Find Nonzero Elements Block

You can use the new Find block to locate all nonzero elements of an input signal. This block outputs the indices of nonzero elements in linear indexing or subscript form and provides these benefits:

When you use the block to...	You can...
Convert logical indexing to linear indexing	Use the linear indices you get from processing a logical indexing signal as the input to a Selector or Assignment block
Extract subscripts of nonzero values	Use the subscript of matrices for 2-D or higher-dimensional signal arrays to aid with image processing
Represent sparse signals	Use indices and values as a compact representation of sparse signals

PauseFcn and ContinueFcn Callback Support for Blocks and Block Diagrams

The new PauseFcn and ContinueFcn callbacks detect clicking of the **Pause** and **Continue** buttons during simulation. You can set these callbacks using the set_param command or the **Callbacks** tab of the Model Properties dialog box. Both the PauseFcn and ContinueFcn callbacks support Normal and Accelerator simulation modes.

Gain Block Can Inherit Parameter Data Type from Gain Value

The Gain block now supports the **Parameter data type** setting of Inherit: Inherit from 'Gain'. This enhancement provides the benefit of inheriting the parameter data type directly from the **Gain** parameter. For example:

If you set Gain to...	The parameter data type inherits...
2	double
single(2)	single
int8(2)	int8

Direct Lookup Table (n-D) Block Enhancements
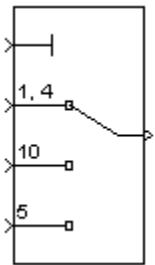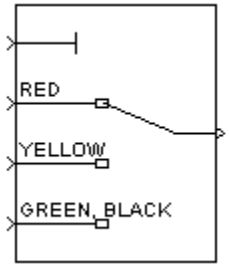
The Direct Lookup Table (n-D) block now supports:

• Multidimensional signals for the table input port

- Fixed-point data types for the table input port

- Explicit specification of the table data type in the block dialog box

Multiport Switch Block Allows Explicit Specification of Data Port Indices

The icon for the Multiport Switch block now shows the values of indices on data port labels. This enhancement helps you identify the data inputs without having to open the block dialog box:

Block Parameter Settings	Block Icon
Data port order: Specify indices Data port indices (e.g. {1,[2,3]}): {[1,4],10,5}	
Data port order: Specify indices Data port indices (e.g. {1,[2,3]}): {myColors.RED, myColors.YELLOW, [myColors.GREEN, myColors.BLACK]}	

When you load existing models that contain the Multiport Switch block, the following parameter mapping occurs:

Block Parameter Settings of a Model from R2009b or Earlier	Block Parameter Settings When You Load the Model in R2010a
Number of inputs: [3] ☐ Use zero-based indexing	Data port order: [One-based contiguous] Number of data ports: [3]
Number of inputs: [3] ☑ Use zero-based indexing	Data port order: [Zero-based contiguous] Number of data ports: [3]

The following command-line parameter mapping applies:

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of inputs	Number of data ports	Inputs	Same
Use zero-based indexing	Data port order	zeroidx	DataPortOrder

The parameter mapping in R2010a ensures that you get the same block behavior as in previous releases.

Compatibility Considerations. In R2010a, a warning appears at compile time when your model contains a Multiport Switch block with the following configuration:

- The control port uses an enumerated data type.

- The data port order is contiguous.

During edit time, the block icon cannot show the mapping of each data port to an enumerated value. This configuration can also lead to unused ports during simulation and unused code during Real-Time Workshop® code generation.

Run the slupdate function on your model to replace each Multiport Switch block of this configuration with a block that explicitly specifies data port indices. Otherwise, your model might not work in a future release.

In R2010a, the following Multiport Switch block configuration also produces a warning at compile time:

- The control port uses a fixed-point or built-in data type.
- The data port order is contiguous.
- At least one of the contiguous data port indices is not representable with the data type of the control port.

The warning alerts you to unused ports during simulation and unused code during Real-Time Workshop code generation.

Trigonometric Function Block Supports CORDIC Algorithm and Fixed-Point Data Types

When you select sin, cos, or sincos for the Trigonometric Function block, additional parameters are available.

New Block Parameter	Purpose	Benefit
Approximation method	Specify the type of approximation the block uses to compute output: None or CORDIC.	Enables you to use a faster method of computing block output for fixed-point and HDL applications.
Number of iterations	For the CORDIC algorithm, specify how many iterations to use for computing block output.	Enables you to adjust the precision of your block output.

This block now supports fixed-point data types when you select `sin`, `cos`, or `sincos` and set **Approximation method** to `CORDIC`.

Enhanced Block Support for Enumerated Data Types

The following Simulink blocks now support enumerated data types:

- Data Type Conversion Inherited
- Data Type Duplicate
- Interval Test
- Interval Test Dynamic
- Probe (input only)
- Relay (output only)
- Unit Delay Enabled
- Unit Delay Enabled Resettable
- Unit Delay Resettable
- Unit Delay With Preview Enabled
- Unit Delay With Preview Enabled Resettable
- Unit Delay With Preview Enabled Resettable External RV
- Unit Delay With Preview Resettable
- Unit Delay With Preview Resettable External RV

For more information, see "Using Enumerated Data" in the *Simulink User's Guide*.

Lookup Table Dynamic Block Supports Direct Selection of Built-In Data Types for Outputs

In R2010a, you can select the following data types directly for the **Output data type** parameter of the Lookup Table Dynamic block:

- `double`
- `single`

- int8

- uint8

- int16

- uint16

- int32

- uint32

- boolean

Previously, you had to enter an expression for **Output data type** to specify a built-in data type.

Compare To Zero and Wrap To Zero Blocks Now Support Parameter Overflow Diagnostic

If the input data type to a Compare To Zero or Wrap To Zero block cannot represent zero, detection of this parameter overflow occurs. In the **Diagnostics > Data Validity** pane of the Configuration Parameters dialog box, set **Parameters > Detect overflow** to warning or error.

Data Type Duplicate Block Enhancement

The Data Type Duplicate block is now a built-in block. Previously, this block was a masked S-Function. The read-only **BlockType** parameter has changed from S-Function to DataTypeDuplicate.

Compatibility Considerations. In R2010a, signal propagation might behave differently from previous releases. As a result, your model might not compile under these conditions:

- Your model contains a Data Type Duplicate block in a source loop.

- Your model has underspecified signal data types.

If your model does not compile, set data types for signals that are not fully specified.

Lookup Table and Lookup Table (2-D) Blocks To Be Deprecated in a Future Release

In a future release, the Lookup Table and Lookup Table (2-D) blocks will no longer appear in the Simulink Library Browser. Consider replacing instances of those two blocks by using 1-D and 2-D versions of the Lookup Table (n-D) block. Among other enhancements, the Lookup Table (n-D) block supports the following features that the other two blocks do not:

- Specification of parameter data types different from input or output signal types

- Reduced memory use and faster code execution for evenly spaced breakpoints that are nontunable

- Fixed-point data types with word lengths up to 128 bits

- Specification of index search method

- Specification of action for out-of-range inputs

To upgrade your model:

Step	Description	Reason
1	Run the Simulink Model Advisor check for "Check model, local libraries, and referenced models for known upgrade issues".	Identify blocks that do not have compatible settings with the Lookup Table (n-D) block.
2	For each block that does not have compatible settings with the Lookup Table (n-D) block: • Decide how to address each warning. • Adjust block parameters as needed.	Modify each Lookup Table or Lookup Table (2-D) block to make them compatible.
3	Repeat steps 1 and 2 until you are satisfied with the results of the Model Advisor check.	Ensure that block replacement works for the entire model.
4	Run the slupdate function on your model.	Perform block replacement with the Lookup Table (n-D) block.

Compatibility Considerations. The Model Advisor check groups all Lookup Table and Lookup Table (2-D) blocks into three categories:

• Blocks that have compatible settings with the Lookup Table (n-D) block

• Blocks that have incompatible settings with the Lookup Table (n-D) block

• Blocks that have repeated breakpoints

Blocks with Compatible Settings

When a block has compatible parameter settings with the Lookup Table (n-D) block, automatic block replacement can occur without backward incompatibilities.

Lookup Method in the Lookup Table or Lookup Table (2-D) Block	Parameter Settings in the Lookup Table (n-D) Block After Block Replacement	
	Interpolation	Extrapolation
Interpolation-Extrapolation	Linear	Linear
Interpolation-Use End Values	Linear	None-Clip
Use Input Below	None-Flat	Not applicable

Depending on breakpoint characteristics, the Lookup Table (n-D) block uses one of two index search methods.

Breakpoint Characteristics in the Lookup Table or Lookup Table (2-D) Block	Index Search Method in the Lookup Table (n-D) Block After Block Replacement
Not evenly spaced	Binary search
Evenly spaced and tunable	A prompt appears, asking you to select Binary search or Evenly spaced points.
Evenly spaced and nontunable	

The Lookup Table (n-D) block also adopts other parameter settings from the Lookup Table or Lookup Table (2-D) block. For parameters that exist only in the Lookup Table (n-D) block, the following default settings apply after block replacement:

Lookup Table (n-D) Block Parameter	Default Setting After Block Replacement
Breakpoint data type	Inherit: Same as corresponding input
Action for out-of-range input	None

Blocks with Incompatible Settings

When a block has incompatible parameter settings with the Lookup Table (n-D) block, the Model Advisor shows a warning and a recommended action, if applicable.

• If you perform the recommended action, you can avoid incompatibility during block replacement.

- If you use automatic block replacement without performing the recommended action, you might see numerical differences in your results.

Incompatibility Warning	Recommended Action	What Happens for Automatic Block Replacement
The **Lookup Method** is `Use Input Nearest` or `Use Input Above`. The Lookup Table (n-D) block does not support these lookup methods.	Change the lookup method to one of the following: • `Interpolation - Extrapolation` • `Interpolation - Use End Values` • `Use Input Below`	The **Lookup Method** changes to `Interpolation - Use End Values`. In the Lookup Table (n-D) block, this setting corresponds to: • **Interpolation** set to `Linear` • **Extrapolation** set to `None-Clip` You also see a message that explains possible numerical differences.
The **Lookup Method** is `Interpolation - Extrapolation`, but the input and output are not the same floating-point type. The Lookup Table (n-D) block supports linear extrapolation only when all inputs and outputs are the same floating-point type.	Change the extrapolation method or the port data types of the block.	
The block uses small fixed-point word lengths, so that interpolation uses only one rounding operation. The Lookup Table (n-D) block uses two rounding operations for interpolation.	None	You see a message that explains possible numerical differences.

Blocks with Repeated Breakpoints

When a block has repeated breakpoints, the Model Advisor recommends that you change the breakpoint data and rerun the check. You cannot perform automatic block replacement for blocks with repeated breakpoints.

Elementary Math Block Now Obsolete

The Elementary Math block is now obsolete. You can replace any instance of this obsolete block in your model by using one of these blocks in the Math Operations library:

- Math Function
- Rounding Function
- Trigonometric Function

Compatibility Considerations. If you open a model that contains an Elementary Math block, a warning message appears. This message suggests running `slupdate` on your model to replace each instance of the obsolete block with an appropriate substitute.

If you try to start simulation or generate code for a model that contains this obsolete block, an error message appears.

DocBlock Block RTF File Compression

In R2010a, when you add or modify a DocBlock block that uses Microsoft® RTF format and you save the model, Simulink compresses the RTF file. The saved RTF files with images are much smaller than in previous releases.

Compatibility Considerations. In R2010a, if you use `slupdate` or save a model that includes a DocBlock block that uses RTF format, you cannot run the model in an earlier version of Simulink.

To run a model that has a compressed RTF file in an earlier version of Simulink, use **Save As** to save the model in the format of the earlier release.

Simulink Extras PID Controller Blocks Deprecated

In R2010a, the PID Controller (with Approximate Derivative) and PID Controller blocks of the Simulink Extras library no longer appear in the Simulink Library Browser. For models created using R2009b or earlier, consider using the `slupdate` function to replace these blocks with the new PID Controller block of the Simulink/Continuous or Simulink/Discrete library. Among other enhancements, the new PID Controller block supports:

- Continuous-time and discrete-time modeling

- Ideal and Parallel controller forms

- Automatic PID tuning (requires a Simulink® Control Design™ license)

For more information, see the PID Controller and PID Controller (2 DOF) block reference pages.

Compatibility Considerations. For backward compatibility, simulation and code generation of models that contain the deprecated PID Controller (with Approximate Derivative) or PID Controller block continue to work.

User Interface Enhancements

Model Explorer Column Views

The Model Explorer now supports column views, which specify sets of property columns to display in the **Contents** pane. The Model Explorer displays only the properties that are defined for the current column view. The Model Explorer does not add new properties dynamically as you add objects to the **Contents** pane. Using a defined subset of properties to display streamlines the task of exploring and editing model object properties and increases the density of the data displayed.

Model Explorer provides several standard column views with common property sets. You can:

- Select the column view based on the task you are performing

- Customize the standard column views

- Create your own column views

- Export and import column views saved in MAT-files, which you can share with other users

See "Controlling the Set of Displayed Properties".

Compatibility Considerations. Column views replace the **Customize Contents** option provided in previous releases.

In R2010a, the Model Explorer provides a different interface for performing some of the tasks that you previously performed using **View** menu items. The following table summarizes differences between R2009b and R2010a.

R2009b View Menu Item	R2010a Model Explorer Interface Change
Dialog View	Replaced by **Show Dialog Pane**
Customize Contents	Replaced by **Column View > Show Details**
Show Properties	Eliminated; select **Column View > Show Details** to specify properties to display
Mark Nonexistent Properties	Replaced by **Show Nonexistent Properties as -**
Library Browser	Eliminated (you can access the Library Browser from the Simulink Editor **View** menu)
List View Options	Replaced by **Row Filter**

Model Explorer Display of Masked Subsystems and Linked Library Subsystems

The Model Explorer now contains global options for specifying whether the Model Explorer displays the contents of library links and masked subsystems. These options also control whether the **Model Hierarchy** pane displays linked or masked subsystems. See "Displaying Masked Subsystems"and "Displaying Linked Library Subsystems".

Compatibility Considerations. In R2010a, when you select a masked subsystem node in the **Model Hierarchy** pane, the **Contents** pane displays the objects of the subsystem, reflecting the global setting to display masked subsystems. In prior releases, if you selected a masked subsystem node, you needed to right-click the node and select **Look Under Mask** to view the subsystem objects in the **Contents** pane.

In R2010a, the search results reflect the **Show Library Links** and **Show Masked Subsystems** settings. In previous releases, you specified the **Look**

Inside Masked Subsystems and **Look Inside Linked Subsystems** options as part of the search options. R2010a does not include those search options.

Model Explorer Object Count

The top-right section of the **Contents** pane displays a count of objects found for the currently selected nodes in the **Model Hierarchy** pane. The count indicates the number of objects displayed in the **Contents** pane, compared to the total number of objects in the currently selected nodes. The number of displayed objects is less than the total number of objects in scope when you filter some objects by using **View > Row Filter** options. See .

Model Explorer Search Option for Variable Usage

You can use the new `for Variable Usage` search type to search for blocks that use a variable that is defined in the base or model workspaces. See "Search Bar Controls".

Model Explorer Display of Signal Logging and Storage Class Properties

The Model Explorer **Contents** pane displays the following additional properties for signal lines:

- Signal logging-related properties (such as `DataLogging`)
- Storage class properties, including properties associated with custom storage classes for signals

Displaying these properties in the **Contents** pane enables batch editing. Prior to R2010a, you could edit these properties only in the Signal Properties dialog box.

Model Explorer Column Insertion Options

In R2010a, right-clicking on a column heading in the Contents pane provides two new column insertion options:

- **Insert Path** – adds the `Path` property column to the right of the selected column.

- **Insert Recently Hidden Columns** – selects a property from a list of columns you recently hid, to add that property column to the right of the selected column

See "Adding Property Columns".

Diagnostics for Data Store Memory Blocks

The Model Advisor 'By Task' folder now contains a Data Store Memory Blocks subfolder. This subfolder contains checks relating to Data Store Memory blocks that examine your model for:

- Multitasking, strong typing, and shadowing issues
- An enabled status of the read/write diagnostics
- Read/write issues

New Command-Line Option for RSim Targets

A new h command-line option allows you to print a summary of the available options for RSim executable targets.

Simulink.SimulationOutput.get Method for Obtaining Simulation Results

The Simulink.SimulationOutput class now has a `get` method. After simulating your model, you can use this method to access simulation results from the Simulink.SimulationOutput object.

Simulink.SimState.ModelSimState Class has New snapshotTime Property

The Simulink.SimState.ModelSimState class has a new `snapshotTime` property. You can use this property to access the exact time at which Simulink took a "snapshot" of the simulation state (SimState) of your model.

Simulink.ConfigSet.saveAs to Save Configuration Sets

The `saveAs` method is added to the `Simulink.ConfigSet` class to allow you to easily save the settings of configuration sets as MATLAB functions or scripts. Using the MATLAB function or script, you can share and archive

model configuration sets. You can also compare the settings in different configuration sets by comparing the MATLAB functions or scripts of the configuration sets.

For details, see "Saving Configuration Sets" in the *Simulink User's Guide*.

S-Functions

Building C MEX-Files from Ada and an Example Ada Wrapper

In an R2008b release note, the MathWorks announced that support for Ada S-functions in Simulink would be removed in a future release and a migration strategy would be forthcoming. In this release, the addition of a technical note facilitates your incorporating Ada code into Simulink without using Ada S-functions support.

This technical note demonstrates:

- how to build a C MEX S-function from Ada code without using the `mex ada` command

- an example Ada wrapper around a C MEX S-Function API

New S-Function API Checks for Branched Function-Calls

A new S-function API, `ssGetCallSystemNumFcnCallDestinations`, allows you to determine the number of function-call blocks that your S-function calls. Based on this returned number, you can then deduce whether or not your S-function calls a branched function-call.

You can call this SimStruct function from `mdlSetWorkWidths` or later in your S-function.

New C MEX S-Function API and M-File S-Function Flag for Compliance with For Each Subsystem

To allow a C MEX S-function to reside inside of a For Each Subsystem block, you must call the new `ssSupportsMultipleExecInstances` API and set the flag to true in the `mdlSetWorkWidths` method.

As for M-file S-functions, you must set the new flag `block.SupportsMultipleExecInstances` to true in the Setup section.

Legacy Code Tool Enhanced to Support Enumerated Data Types and Structured Tunable Parameters

The Legacy Code Tool has been enhanced to support

- Enumerated data types for input, output, parameters, and work vectors
- Structured tunable parameters

For more information about data types that the Legacy Code Tool supports, see "Supported Data Types". For more information about the Legacy Code Tool, see

- "Integrating Existing C Functions into Simulink Models with the Legacy Code Tool" in the Writing S-Functions documentation
- `legacy_code` function reference page

Compatibility Considerations. For enumerated data type support:

- If you upgrade from R2008b or later release, you can continue to compile the S-function source code and continue to use the compiled output from an earlier release without recompiling the code.
- If you upgrade from R2008a or earlier release, you cannot use enumerated types; the Simulink engine will display an error during simulation.

You cannot use tunable structured parameters with Legacy Code Tool in a release prior to R2010a.

Documentation Improvements

Modeling Guidelines for High-Integrity Systems

The MathWorks intends the "Modeling Guidelines for High-Integrity Systems" document for engineers developing models and generating code for high-integrity systems using Model-Based Design with MathWorks products. This document describes creating Simulink models that are complete,

unambiguous, statistically deterministic, robust, and verifiable. The document focus is on model settings, block usage, and block parameters that impact simulation behavior or code generated by the Real-Time Workshop® Embedded Coder™ product.

These guidelines do not assume that you use a particular safety or certification standard. The guidelines reference some safety standards where applicable, including DO-178B, IEC 61508, and MISRA C®.

You can use the Model Advisor to support adhering to these guidelines. Each guideline lists the checks that are applicable to that guideline.

For more information, see "Modeling Guidelines for High-Integrity Systems" in the Simulink documentation.

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Included in Help

The MathWorks Automotive Advisory Board (MAAB) involves major automotive original equipment manufacturers (OEMs) and suppliers in the process of evolving MathWorks controls, simulation, and code generation products, including the Simulink, Stateflow, and Real-Time Workshop products. An important result of the MAAB has been the "MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow." Help for the Simulink product now includes these guidelines. The MAAB guidelines link to relevant Model Advisor MAAB check help and MAAB check help links to relevant MAAB guidelines.

For more information, see "MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow" in the Simulink documentation.

Version 7.4 (R2009b) Simulink Software

This table summarizes what's new in V7.4 (R2009b):

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

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

- "Simulation Performance" on page 37
- "Component-Based Modeling" on page 39
- "Variable-Size Signals" on page 42
- "Embedded MATLAB Function Blocks" on page 43
- "Simulink Data Management" on page 44
- "Removing Support for Custom Floating-Point Types in Future Release" on page 46
- "Simulink File Management" on page 47
- "Block Enhancements" on page 47
- "User Interface Enhancements" on page 57

Simulation Performance

Single-Output `sim` Syntax

An enhanced `sim` command provides for greater ease of use and for greater compatibility with `parfor` loops. Since the command now saves all simulation results to a single object, the management of output variables is straightforward for all cases, including parallel computing.

Expanded Support by Rapid Accelerator

Simulink Rapid Accelerator mode now supports root inputs of enumerated data type and fixed-point parameters of any word length.

SimState Support in Accelerator Mode

Simulink Accelerator mode now supports the SimState feature. You can therefore save the simulation state and later resume the simulation from the exact save time.

Integer Arithmetic Applied to Sample Hit Computations

For fixed-step simulations, Simulink now computes sample time hits using integer arithmetic. This modification improves the timing resolution of sample hits of multirate models.

Compatibility Considerations. Previously, if an S-function had two rates, and if (ssIsSampleHit(S, idx1) == true && ssIsSampleHit(S,idx2) == true, then Simulink would adjust the task times to be evaluated as ssGetTaskTime(S, idx1) == ssGetTaskTime(S, idx2). Simulink no longer forces this equality; instead, Simulink now leaves the individual task times to be integer multiples of their corresponding periods. Consequently, existing code with logic that relies upon the equality of the task times needs to be updated.

In addition, the behavior of the command get_param(model, 'SimulationTime') is now different. Instead of returning the time of the next known sample hit at the bottom of the current step, this command now returns the current time.

Improved Accuracy of Variable-Step Discrete Solver

For variable-step discrete simulation of purely discrete models, where the fundamental step size is the same as the fastest discrete rate, Simulink now uses the specified start and stop times.

Compatibility Considerations. Previously, if the fundamental step size was equal to the fastest discrete rate, the Simulink simulation did not uniformly honor the user-specified start and stop times. Specifically, if the start and stop times were not exact multiples of the fundamental step size, then the start time was adjusted to the time of the first sample time hit and the simulation stopped at the sample time hit just before the specified stop time. However, if the simulation was required to hit certain time points (either by specifying `TSPAN` in the `sim` command such as 'sim('Model_A',[0 10])', or via the `OutputTimes` parameter), then the start and stop times were not adjusted

Now Simulink variable-step simulation of purely discrete models consistently honors the user-specified start and stop times, irrespective of whether the fastest discrete sample time is the GCD of all of the other sample times

Component-Based Modeling

Enhanced Library Link Management
In R2009b, improved library link management facilitates visualizing and restoring edited library links. See "Working with Library Links" for more information.

Enhanced Mask Editor Provides Tabs and Signal Attributes
You can use the R2009b Mask Editor to create a mask that has tabbed panes, and define the same signal attribute specifications in a mask that built-in Simulink blocks provide. See "Working with Block Masks" , "Simulink Mask Editor"and "Mask Icon Drawing Commands"for more information.

Model Reference Variants
Model reference variants allow you to configure any Model block to select its referenced model from a set of candidate models. The selection occurs when you compile the model that contains the Model block, and depends on the values of one or more MATLAB variables or Simulink parameters in the base workspace. To configure a Model block to select the model that it references, you:

• Provide a set of Boolean expressions that reference base workspace values.

- Associate each expression with one of the models that the block could reference.

When you compile the model, Simulink evaluates all the expressions. Each Model block that uses model reference variants then selects the candidate model whose associated expression is `true`, and ignores all the other models. Compilation then proceeds exactly as if you had entered the name of the selected model literally in the Model block's **Model name** field.

You can nest Model blocks that use variants to any level, allowing you to define any number of arbitrarily complex customized models within a single framework. No matter how many simulation environments you define, selecting one requires only setting variable or parameter values appropriately in the base workspace. See "Using Model Reference Variants" for more information.

Protected Referenced Models

A *protected model* is a referenced model from which all block and line information has been eliminated. Protecting a model does not use encryption technology. A protected model can be distributed without revealing the intellectual property that it embodies. The model is said to run in Protected mode, and gives the same results that its source model does when run in Accelerator mode.

You can use a protected model much as you could any referenced model that executes in Accelerator mode. Simulink tools work with protected models to the extent possible given that the model's contents are obscured. For example, the Model Explorer and the Model Dependency Viewer show the hierarchy under an ordinary referenced model, but not under a protected model. Signals in a protected model cannot be logged, because the log could reveal information about the protected model's contents.

When a referenced model requires object definitions or tunable parameters that are defined in the MATLAB base workspace, the protected version of the model may need some or all of those same definitions when it executes as part of a third-party model. Simulink provides techniques for identifying and obtaining the needed data. You can use the Simulink Manifest Tools or other techniques to package the model and any data for delivery.

Protecting a model requires a Real-Time Workshop license, which makes code generation capabilities available for use internally when creating the protected version of the model. The receiver of a protected model does not need a Real-Time Workshop license to use the model, and cannot use Real-Time Workshop to generate code for the model or any model that references it.

To accommodate protected models, the Model block now accepts a suffix in the **Model name** field. This suffix can be `.mdl` for an unprotected model or `.mdlp` for a protected model. If the suffix is omitted, Model block first searches the MATLAB path for a block with the specified name and the suffix `.mdl`. If that search fails, the block searches the path for a model with the suffix `.mdlp`.

The Model block now has a field named `ProtectedModel`, a boolean that indicates whether the referenced model is protected, and three fields for representing the name of the referenced model in different formats: `ModelNameDialog`, `ModelName`, and `ModelFile`. See the Model block parameters in "Ports & Subsystems Library Block Parameters" for information about these parameters. For more information about protecting models, see "Protecting Referenced Models".

Simulink Manifest Tools

Enhanced Simulink Manifest Tools now discover and analyze model variants, protected models, and Simscape™ files.

New manifest analysis options for controlling whether to report file dependency locations for user files, all files, or no files. For example, you may not want to view the file locations of all the dependencies on MathWorks products. This is typical if your main use of Simulink Manifest Tools is to discover and package all the required files for your model. By not analyzing file locations, you speed up report creation, and the report is smaller and easier to navigate. If you need to trace all dependencies to understand why a particular file or toolbox is required by a model, you can always regenerate the full report of all files.

The manifest report is enhanced with sortable columns, and now MATLAB Programs as well as P-files are reported in the manifest if both exist.

For more information, see "Model Dependencies" in the Simulink User's Guide.

S-Function Builder

The S-Function Builder has been enhanced to support bus signals for managing complex signal interfaces. See *Developing S-Functions* for more information.

Variable-Size Signals

New capability that allows signal sizes to change during execution facilitates modeling of systems with varying environments, resources, and constraints. For Simulink models that demonstrate using variable-size signals, see "Working with Variable-Size Signals"

Simulink Support

- Referenced Model
- Simulink Accelerator and Rapid Accelerator
- Bus Signals
- C-mex S-function
- Level-2 M-file S-function
- Simulink Debugger
- Signal Logging and Loading
- Block Run-Time Object

Simulink Block Support

Support for variable-size signal inputs and outputs in over 40 Simulink blocks including many blocks from the Math Operations library. For a list of Simulink blocks, see "Simulink Block Support for Variable-Size Signals"

Embedded MATLAB Function Blocks

Support for Variable-Size Arrays and Matrices

Embedded MATLAB Function blocks now support variable-size arrays and matrices with known upper bounds. With this feature, you can define inputs, outputs, and local variables to represent data that varies in size at runtime.

Change in Text and Visibility of Parameter Prompt for Easier Use with Fixed-Point Advisor and Fixed-Point Tool

The **Lock output scaling against changes by the autoscaling tool** check box is now **Lock data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This enhancement enables you to lock the current data type settings on the dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

New Compilation Report for Embedded MATLAB Function Blocks

The new compilation report provides compile-time type information for the variables and expressions in your Embedded MATLAB functions. This information helps you find the sources of error messages and understand type propagation issues, particularly for fixed-point data types. For more information, see "Working with Compilation Reports" in the *Simulink User's Guide*.

Compatibility Considerations. The new compilation report is not supported by the MATLAB internal browser on Sun™ Solaris™ 64-bit platforms. To view the compilation report on Sun Solaris 64-bit platforms, you must configure your MATLAB Web preferences to use an external browser, for example, Mozilla Firefox. To learn how to configure your MATLAB Web preferences, see Web Preferences in the MATLAB documentation.

New Options for Controlling Run-time Checks for Faster Performance

In simulation, the code generated for Embedded MATLAB Function blocks includes various run-time checks. To reduce the size of the generated code,

and potentially improve simulation times, you can use new **Simulation Target** configuration parameters to control whether or not your generated code performs:

- Integrity checks to detect violations of memory integrity in the generated code. For more information, see "Ensure memory integrity" in the *Simulink Graphical User Interface*.

- Responsiveness checks to periodically check for Ctrl+C breaks and refresh graphics. For more information, see "Ensure responsiveness" in the *Simulink Graphical User Interface*.

Embedded MATLAB Function Blocks Improve Size Propagation Behavior

Heuristics for size propagation have improved for underspecified models. During size propagation, Embedded MATLAB Function blocks no longer provide default sizes. Instead, for underspecified models, Simulink gets defaults from other blocks that have more size information.

Compatibility Considerations. Certain underspecified models that previously ran without error may now generate size mismatch errors. Examples of underspecified models include:

- Models that contain a cycle in which no block specifies output size

- Models that do not specify the size of input ports

To eliminate size mismatch errors:

- Specify sizes for the input ports of your subsystem or model.

- Specify sizes of all ports on at least one block in any loop in your model.

Simulink Data Management

New Function Exports Workspace Variables and Values

The new Simulink.saveVars function can save workspace variables and their values into a MATLAB file. The file containing the data is human-readable and can be manually edited. If Simulink cannot generate MATLAB code for a workspace variable, Simulink.saveVars saves that variable into a companion

MAT-file rather than a MATLAB file. Executing the MATLAB file (which also loads any companion MAT file) restores the saved variables and their values to the workspace. See `Simulink.saveVars` for more information.

New Enumerated Constant Block Outputs Enumerated Data

Although the Constant block can output enumerated values, it provides many block parameters that do not apply to enumerated types, such as **Output minimum** and **Output maximum**. In R2009b, the **Sources** library includes the Enumerated Constant block. When you need a block that outputs constant enumerated values, use Enumerated Constant rather than Constant to avoid seeing irrelevant block parameters.

Enhanced Switch Case Block Supports Enumerated Data

The Switch Case block now supports enumerated data types for the input signal and case conditions. For more information, see "Using Enumerated Data" and the Switch Case block documentation.

Code for Multiport Switch Block Shows Enumerated Values

In previous releases, generated code for a Multiport Switch block that uses enumerated data contains the underlying integer for each enumerated value rather than its name. In R2009b, the code contains the name of each enumerated value rather than its underlying integer. This change adds readability and facilitates comparing the code with the model, but has no effect on the behavior of the code. For more information, see "Using Enumerated Data" and Multiport Switch.

Data Class Infrastructure Partially Deprecated

Some classes and properties in the Simulink data class infrastructure have been deprecated in R2009b. See "Working with Data" for information about Simulink data classes.

Compatibility Considerations. If you use any of the deprecated constructs, Simulink posts a warning that identifies the construct and describes one or more techniques for eliminating it. The techniques differ depending on the construct. You can ignore these warnings in R2009b, but The MathWorks recommends making the described changes now because the deprecated constructs may be removed from future releases, upgrading the warnings to errors.

Saving Simulation Results to a Single Object

Enhanced `sim` command that saves all simulation results to a single object for easier management of simulation results.

Simulation Restart in R2009b

In order to restart an R2009a simulation in R2009b, you should first regenerate the initial SimState in R2009b.

Compatibility Considerations. The SimState that Simulink saves from a R2009a simulation might be incompatible with the internal representation of the same model in R2009b. Simulink detects this incompatibility when the R2009a SimState is used to restart a R2009b simulation. If the mismatch resides in the model interface only, then Simulink issues a warning. (You can use the Simulink diagnostic 'SimState interface checksum mismatch' to turn off such warnings or to direct Simulink to report an error.) However, if the mismatch resides in the structural representation of the model, then Simulink reports an error. To avoid these errors and warnings, you need to regenerate the initial SimState in R2009b.

Removing Support for Custom Floating-Point Types in Future Release

Support for custom floating-point types, `float(TotalBits, ExpBits)`, will be removed in a future release.

In R2009b, Simulink continues to process these types.

For more information, see `float`.

Simulink File Management

Removal of Functions

The following functions are no longer available:

- adams.m

- euler.m

- gear.m

- linsim.m

- rk23.m

- rk45.m

Deprecation of SaveAs to R12 and R13

In R2009b, you will no longer be able to use the SaveAs feature to save a model to releases R12 or R13. You will, however, be able to save models to R12 and R13 using the command-line. In R2010a, the command-line capability will also be removed.

Improved Behavior of Save_System

When you use the save_system function to save a model to an earlier release, you will no longer receive a dialog box that indicates that the save was successful.

Block Enhancements

New Turnkey PID Controller Blocks for Convenient Controller Simulation and Tuning

You can implement a continuous- or discrete-time PID controller with just one block by using one of the new PID Controller and PID Controller (2DOF) blocks. With the new blocks, you can:

- Configure your controller in any common controller configuration, including PID, PI, PD, P, and I.

- Tune PID controller gains either manually in the block or automatically in the new PID Tuner. (PID Tuner requires a Simulink Control Design license.)

- Generate code to implement your controller using any Simulink data type, including fixed-point data types (requires a Real-Time Workshop license).

You can set many options in the PID Controller and PID Controller (2DOF) blocks, including:

- Ideal or parallel controller configurations

- Optional output saturation limit with anti-windup circuitry

- Optional signal-tracking mode for bumpless control transfer and multiloop controllers

- Setpoint weighting in the PID Controller (2DOF) block

The blocks are available in the Continuous and Discrete libraries. For more information on using the blocks, see the PID Controller and PID Controller (2DOF) reference pages. For more information on tuning the PID blocks, see Automatic PID Tuning in the Simulink Control Design reference pages.

New Enumerated Constant Block Outputs Enumerated Data

Although the Constant block can output enumerated values, it provides many block parameters that do not apply to enumerated types, such as **Output minimum** and **Output maximum**. In R2009b, the **Sources** library includes the Enumerated Constant block. When you need a block that outputs constant enumerated values, use Enumerated Constant rather than Constant to avoid seeing irrelevant block parameters.

Enhanced Switch Case Block Supports Enumerated Data

The Switch Case block now supports enumerated data types for the input signal and case conditions. For more information, see "Using Enumerated Data" and the Switch Case block documentation.

Code for Multiport Switch Block Shows Enumerated Values

In previous releases, generated code for a Multiport Switch block that uses enumerated data contains the underlying integer for each enumerated

value rather than its name. In R2009b, the code contains the name of each enumerated value rather than its underlying integer. This change adds readability and facilitates comparing the code with the model, but has no effect on the behavior of the code. For more information, see "Using Enumerated Data" and Multiport Switch.

Discrete Transfer Fcn Block Has Performance, Data Type, Dimension, and Complexity Enhancements

The following enhancements apply to the Discrete Transfer Fcn block:

- Improved numerics and run-time performance of outputs and states by reducing the number of divide operations in the filter to one

- Support for signed fixed-point and signed integer data types

- Support for vector and matrix inputs

- Support for input and coefficients with mixed complexity

- A new **Initial states** parameter for entering nonzero initial states

- A new **Optimize by skipping divide by leading denominator coefficient (a0)** parameter that provides more efficient implementation by eliminating all divides when the leading denominator coefficient is one. This enhancement provides optimized block performance.

Compatibility Considerations. Due to these enhancements, you might encounter the following compatibility issues:

- **Realization parameter removed**

  The Real-Time Workshop software `realization` parameter has been removed from this block. You can no longer use the `set_param` and `get_param` functions on this block parameter. The generated code for this block has been improved to be similar to the former `'sparse'` realization when the **Optimize by skipping divide by leading denominator coefficient (a0)** parameter is selected, while maintaining tunability as in the former `'general'` realization when the parameter is not selected.

- **State changes**

Due to the reduction in the number of divide operations that the block performs, you might notice that your logged states have changed when the leading denominator coefficient is not one.

Lookup Table (n-D) Block Supports Parameter Data Types Different from Signal Data Types

The Lookup Table (n-D) block supports breakpoint data types that differ from input data types. This enhancement provides these benefits:

- Lower memory requirement for storing breakpoint data that uses a smaller type than the input signal

- Sharing of prescaled breakpoint data between two Lookup Table (n-D) blocks with different input data types

- Sharing of custom storage breakpoint data in Real-Time Workshop generated code for blocks with different input data types

The Lookup Table (n-D) block supports table data types that differ from output data types. This enhancement provides these benefits:

- Lower memory requirement for storing table data that uses a smaller type than the output signal

- Sharing of prescaled table data between two Lookup Table (n-D) blocks with different output data types

- Sharing of custom storage table data in Real-Time Workshop generated code for blocks with different output data types

The Lookup Table (n-D) block also supports separate data type specification for intermediate results. This enhancement enables use of a higher precision for internal computations than for table data or output data.

For consistency with other lookup table blocks, the **Process out-of-range input** parameter prompt is now **Action for out-of-range input**. Similarly, the command-line parameter is now `ActionForOutOfRangeInput`. For backward compatibility, the old command-line parameter `ProcessOutOfRangeInput` continues to work. The parameter settings also remain the same: `None`, `Warning`, or `Error`.

Reduced Memory Use and More Efficient Code for Evenly Spaced Breakpoints in Prelookup and Lookup Table (n-D) Blocks

For the Prelookup and Lookup Table (n-D) blocks, Real-Time Workshop generated code now stores only the first breakpoint, spacing, and number of breakpoints when:

- The breakpoint data is nontunable.

- The index search method is `Evenly spaced points`.

This enhancement reduces memory use and provides faster code execution. Previously, the code stored all breakpoint values in a set, regardless of the tunability or spacing of the breakpoints.

The following enhancements also provide more efficient code for the two blocks:

Block	Enhancement for Code Efficiency
Lookup Table (n-D)	Removal of unnecessary bit shifts for calculating the fraction
Prelookup and Lookup Table (n-D)	Use of simple division instead of computationally expensive function calls for calculating the index and fraction

Math Function Block Computes Reciprocal of Square Root

The Math Function block now supports a new function for computing the reciprocal of a square root: `1/sqrt`. You can use one block instead of two separate blocks for this computation, resulting in smaller block diagrams.

You can select one of two methods for computing the reciprocal of a square root: `Exact` or `Newton-Raphson`. Both methods support real input and output signals. When you use the `Newton-Raphson` method, you can also specify the number of iterations to perform the algorithm.

Math Function Block Enhancements for Real-Time Workshop Code Generation

The Math Function block now supports Real-Time Workshop code generation in these cases:

- Complex input and output signals for the `pow` function, for use with floating-point data types

- Fixed-point data types with fractional slope and nonzero bias for the `magnitude^2`, `square`, and `reciprocal` functions

Relational Operator Block Detects Signals That Are Infinite, NaN, or Finite

The Relational Operator block now includes `isInf`, `isNaN`, and `isFinite` functions to detect signals that are infinite, `NaN`, or finite. These new functions support real and complex input signals. If you select one of these functions, the block changes automatically to one-input mode.

Changes in Text and Visibility of Dialog Box Prompts for Easier Use with Fixed-Point Advisor and Fixed-Point Tool

The **Lock output scaling against changes by the autoscaling tool** check box is now **Lock output data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type for the output, such as `fixdt(1,16,0)`. This check box is now visible for any output data type specification. This enhancement helps you lock the current data type settings on a dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

This enhancement applies to the following blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference

- Discrete Derivative

- Discrete-Time Integrator

- Divide

- Dot Product

- Fixed-Point State-Space

- Gain

- Inport

- Lookup Table

- Lookup Table (2-D)

- Lookup Table Dynamic

- Math Function

- MinMax

- Multiport Switch

- Outport

- Prelookup

- Product

- Product of Elements

- Relay

- Repeating Sequence Interpolated

- Repeating Sequence Stair

- Saturation

- Saturation Dynamic

- Signal Specification

- Switch

The **Lock scaling against changes by the autoscaling tool** check box is now **Lock data type settings against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a

fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This enhancement helps you lock the current data type settings on a dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

This enhancement applies to the following blocks:

- Discrete FIR Filter
- Interpolation Using Prelookup
- Lookup Table (n-D)
- Sum
- Sum of Elements

Direct Lookup Table (n-D) Block Enhancements

The Direct Lookup Table (n-D) block now supports:

- Direct entry of **Number of table dimensions**
- Entry of **Table data** using the Lookup Table Editor

Previously, entering an integer greater than 4 for the **Number of table dimensions** required editing **Explicit number of table dimensions**. This extra parameter no longer appears on the block dialog box. For backward compatibility, scripts that contain `explicitNumDims` continue to work.

The other parameters for the block have changed as follows. For backward compatibility, the old command-line parameters continue to work.

Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of table dimensions	`maskTabDims`	`NumberOfTableDimensions`
Inputs select this object from table	`outDims`	`InputsSelectThisObjectFromTable`
Make table an input	`tabIsInput`	`TableIsInput`

Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Table data	`mxTable`	`Table`
Action for out-of-range input	`clipFlag`	`ActionForOutOfRangeInput`
Sample time	`samptime`	`SampleTime`

The read-only **BlockType** parameter has also changed from `S-Function` to `LookupNDDirect`.

Compatibility Considerations. In R2009b, signal dimension propagation can behave differently from previous releases. Your model might not compile under these conditions:

- A Direct Lookup Table (n-D) block is in a source loop.

- Underspecified signal dimensions exist.

If your model does not compile, set dimensions explicitly for underspecified signals.

Unary Minus Block Enhancements

Conversion of the Unary Minus block from a masked S-Function to a core block enables more efficient simulation of the block.

You can now specify sample time for the block. The **Saturate to max or min when overflows occur** check box is now **Saturate on integer overflow**, and the command-line parameter is now `SaturateOnIntegerOverflow`. For backward compatibility, the old command-line parameter `DoSatur` continues to work.

The read-only **BlockType** parameter has also changed from `S-Function` to `UnaryMinus`.

Weighted Sample Time Block Enhancements

Conversions of the Weighted Sample Time and Weighted Sample Time Math blocks from masked S-Functions to core blocks enable more efficient simulation of the blocks.

The following parameter changes apply to both blocks. For backward compatibility, the old command-line parameters continue to work.

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Output data type mode	Output data type	`OutputDataType ScalingMode`	`OutDataTypeStr`
Saturate to max or min when overflows occur	Saturate on integer overflow	`DoSatur`	`SaturateOnIntegerOverflow`

The read-only **BlockType** parameter has also changed from `S-Function` to `SampleTimeMath`.

Switch Case Block Parameter Change

For the Switch Case block, the command-line parameter for the **Show default case** check box is now `ShowDefaultCase`. For backward compatibility, the old command-line parameter `CaseShowDefault` continues to work.

Signal Conversion Block Parameter Change

For the Signal Conversion block, the parameter prompt for the **Override optimizations and always copy signal** check box is now **Exclude this block from 'Block reduction' optimization**.

Compare To Constant and Compare To Zero Blocks Use New Default Setting for Zero-Crossing Detection

The **Enable zero-crossing detection** parameter is now on by default for the Compare To Constant and Compare To Zero blocks. This change provides consistency with other blocks that support zero-crossing detection.

Signal Builder Block Change

You can no longer see the system under the Signal Builder block mask. In previous releases, you could right-click this block and select **Look Under Mask**.

In the Model Explorer, the Signal Builder block no longer appears in the Model Hierarchy view. In previous releases, this view was visible.

User Interface Enhancements

Context-Sensitive Help for Simulink Blocks in the Continuous Library

R2009b introduces context-sensitive help for parameters that appear in Simulink blocks of the Continuous library. This feature provides quick access to a detailed description of the block parameters.

To use the context-sensitive help:

1 Place your pointer over the label of a parameter and right-click.

2 A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu that appears after right-clicking the **Enable zero-crossing detection** parameter for the PID Controller block.
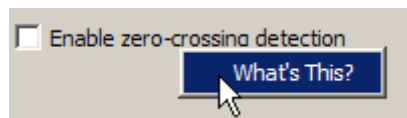
3 Click **What's This?** A window appears showing a description of the parameter.

Adding Blocks from a Most Frequently Used Blocks List

If you are using the same block repeatedly in a model, then you can save time by using the:

- **Most Frequently Used Blocks** tab in the Library Browser
- **Most Frequently Used Blocks** context menu option in the Model Editor

These features provide quick access to blocks you have added to models frequently. For details, see "Adding Frequently Used Blocks".

Highlighting for Duplicate Inport Blocks

The **Highlight to Destination** option for a signal provides more information now for duplicate inport blocks. Applying this option to a signal of an inport block that has duplicate blocks highlights:

- The signal and destination block for that signal
- The signals and destination blocks of the duplicate blocks at the currently opened level in the model

Using the Model Explorer to Add a Simulink.NumericType Object

You can add a Simulink.NumericType object to the model workspace using the Model Explorer, provided you do not enable the **Is alias** option.

An example of when you might use this feature is when you:

- Want to define user-defined data types together in the model
- Do not need to preserve the data type name in the model or in the generated code

Block Output Display Dialog Has OK and Cancel Buttons

The **Block Output Display** dialog now includes **OK** and **Cancel** buttons to specify whether or not to apply your option settings.

Improved Definition of Hybrid Sample Time

Historically, you could not use the hybrid sample time to effectively identify a multirate subsystem or block. A subsystem was marked as "hybrid" and colored in yellow whether it contained two discrete sample times or one discrete sample time and one or more blocks with constant sample time [inf, 0]. Now, in R2009b, the check for the hybrid attribute no longer includes constant sample times, thereby improving the usefulness of the hybrid sample time color in identifying subsystems (and blocks) that are truly multirate.

Find Option in the Model Advisor

In R2009b, the Model Advisor includes a **Find** option to help you find checks. The find option, accessible through the **Edit** menu, allows you to find checks and folders more easily by searching names and analysis descriptions.

For more information, see "Overview of the Model Advisor Window".

Version 7.3 (R2009a) Simulink Software

This table summarizes what's new in V7.3 (R2009a):

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

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

- "Simulation Performance" on page 60
- "Component-Based Modeling" on page 61
- "Embedded MATLAB Function Blocks" on page 63
- "Data Management" on page 63
- "Simulink File Management" on page 65
- "Block Enhancements" on page 65
- "User Interface Enhancements" on page 75
- "S-Functions" on page 76
- "Removal of Lookup Table Designer from the Lookup Table Editor" on page 77

Simulation Performance

Saving and Restoring the Complete SimState

Use the new SimState feature to save the complete simulation state. Unlike the final states stored in earlier versions of Simulink, the SimState contains the complete simulation state of the model (including block states that are

logged). You can then restore the state at a later time and continue simulation from the exact instant at which you stopped the simulation.

Save Simulink Profiler Results

Save the results of the Simulink Profiler and later regenerate reports for review or for comparison.

Component-Based Modeling

Port Value Displays in Referenced Models
In R2009a, port value displays can appear for blocks in a Normal mode referenced model. To control port value displays, choose **View > Port Values** in the model window. For complete information about port value displays, see "Displaying Block Outputs".

Parallel Builds Enable Faster Diagram Updates for Large Model Reference Hierarchies In Accelerator Mode
R2009a provides potentially faster diagram updates for models containing large model reference hierarchies by building referenced models that are configured in Accelerator mode in parallel whenever possible. For example, updating of each model block can be distributed across the cores of a multicore host computer.

To take advantage of this feature, Parallel Computing Toolbox™ software must be licensed and installed in your development environment. If Parallel Computing Toolbox software is available, updating a model diagram rebuilds referenced models configured in Accelerator mode in parallel whenever possible.

For example, to use parallel building for updating a large model reference hierarchy on a desktop machine with four cores, you could perform the following steps:

1 Issue the MATLAB command 'matlabpool 4' to set up a pool of four MATLAB workers, one for each core, in the Parallel Computing Toolbox environment.

2 Open your model and make sure that the referenced models are configured in Accelerator mode.

3 Optionally, inspect the model reference hierarchy. For example, you can use the Model Dependency Viewer from the **Tools** menu of Model Explorer to determine, based on model dependencies, which models will be built in parallel.

4 Update your model. Messages in the MATLAB command window record when each parallel or serial build starts and finishes.

The performance gain realized by using parallel builds for updating referenced models depends on several factors, including how many models can be built in parallel for a given model referencing hierarchy, the size of the referenced models, and host machine attributes such as amount of RAM and number of cores.

The following notes apply to using parallel builds for updating model reference hierarchies:

- Parallel builds of referenced models support only local MATLAB workers. They do not support remote workers in MATLAB® Distributed Computing Server™ configurations.

- The host machine should have an appropriate amount of RAM available for supporting the number of local workers (MATLAB sessions) that you plan to use. For example, setting `matlabpool` to 4 results in five MATLAB sessions on your machine, each using approximately 120 MB of memory at startup.

- The same MATLAB environment must be set up in each MATLAB worker session as in the MATLAB client session — for example, the same base workspace variables, MATLAB path settings, and so forth. You can do this using the `PreLoadFcn` callback of the top model. Since the top model is loaded with each MATLAB worker session, its preload function can be used for any MATLAB worker session setup.

Embedded MATLAB Function Blocks

Support for Enumerated Types

Embedded MATLAB Function blocks now support Simulink enumerated types and generate C code for enumerated data. See "Using Enumerated Data in Embedded MATLAB Function Blocks" in the Simulink documentation.

Use of Basic Linear Algebra Subprograms (BLAS) Libraries for Speed

Embedded MATLAB Function blocks now use BLAS libraries to speed up low-level matrix operations during simulation. See "Speeding Up Simulation with the Basic Linear Algebra Subprograms (BLAS) Library" in the Simulink documentation.

Data Management

Signal Can Resolve to at Most One Signal Object

You can resolve a named signal to a signal object. The object can then specify or validate properties of the signal. For more information, see `Simulink.Signal`, "Using Signal Objects to Initialize Signals and Discrete States", "Using Signal Objects to Tune Initial Values", and "Applying CSCs to Parameters and Signals".

In previous releases, you could associate a signal with multiple signal objects, provided that the multiple objects specified compatible signal attributes. In R2009a, a signal can be associated with at most one signal object. The signal can reference the object more than once, but every reference must resolve to exactly the same object. A different signal object that has exactly the same properties will not meet the requirement. See "Multiple Signal Objects" for more information.

Compatibility Considerations. A compile-time error occurs in R2009a if a model associates more than one signal object with any signal. To prevent the error, decide which object the signal will use, and delete or reconfigure all references to any other signal objects so that all remaining references resolve to the chosen signal object. See "Displaying Signal Sources and Destinations" for a description of techniques that you can use to trace the full extent of a signal.

"Signed" Renamed to "Signedness" in the Simulink.NumericType class

In previous releases, the Property dialog of a `Simulink.NumericType` object whose **Data type mode** was any `Fixed-point` mode showed a property named **Signed**, which was a checkbox. Selecting the checkbox specified a signed type; clearing it specified an unsigned type. The API equivalent of **Signed** was `Signed`, a Boolean whose values could be `1` (signed) or `0` (unsigned).

In R2009a, a property named **Signedness** replaces **Signed** in the Property dialog of a `Simulink.NumericType` object. You can set **Signedness** to `Signed` (the default), `Unsigned`, or `Auto`, which specifies that the object inherits its **Signedness**. The API equivalent of **Signedness** is `Signedness`, which can be 1 (signed), 0 (unsigned), or `Auto`.

For compatibility with existing models, the property `Signed` remains available in R2009a. Setting `Signed` in R2009a sets `Signedness` accordingly. Accessing `Signed` in R2009a returns the value of `Signedness` if that value is 0 or 1, or generates an error if the value of `Signedness` is `Auto`, because that is not a legal value for `Signed`.

Do not use the `Signed` with `Simulink.NumericType` in new models; use `Signedness` instead. See `Simulink.NumericType` for more information.

"Sign" Renamed to "Signedness" in the Data Type Assistant

For blocks and classes that support fixed-point data types, the property **Sign** previously appeared in the Data Type Assistant when the **Mode** was `Fixed point`. In R2009a, this property appears in the Data Type Assistant as **Signedness**. Only the GUI label of the property differs; its behavior and API are unchanged in all contexts.

Tab Completion for Enumerated Data Types

Tab completion now works for enumerated data types in the same way that it does for other MATLAB classes. See "Tab Completion" in "Instantiating an Enumerated Type in MATLAB" for details.

Simulink File Management

Model Dependencies Tools

Enhanced file dependency analysis has the following new features:

- Files in the Simulink manifest are now recorded relative to a project root folder making manifests easier to share, compare and read. See "Generating Manifests" and "Editing Manifests".

- Command-line dependency analysis can now report toolbox dependencies, and when discovering file dependencies you can optionally generate a manifest file. See "Command-Line Dependency Analysis"

Block Enhancements

Prelookup and Interpolation Using Prelookup Blocks Support Parameter Data Types Different from Signal Data Types

The Prelookup block supports breakpoint data types that differ from input data types. This enhancement provides these benefits:

- Enables lower memory requirement for storing breakpoint data that uses a smaller type than the input signal

- Enables sharing of prescaled breakpoint data between two Prelookup blocks with different input data types

- Enables sharing of custom storage breakpoint data in Real-Time Workshop generated code for blocks with different input data types

The Interpolation Using Prelookup block supports table data types that differ from output data types. This enhancement provides these benefits:

- Enables lower memory requirement for storing table data that uses a smaller type than the output signal

- Enables sharing of prescaled table data between two Interpolation Using Prelookup blocks with different output data types

- Enables sharing of custom storage table data in Real-Time Workshop generated code for blocks with different output data types

The Interpolation Using Prelookup block also supports separate data type specification for intermediate results. This enhancement enables use of a greater precision for internal computations than for table data or output data.

Lookup Table (n-D) and Interpolation Using Prelookup Blocks Perform Efficient Fixed-Point Interpolations

Whenever possible, Lookup Table (n-D) and Interpolation Using Prelookup blocks use a faster overflow-free subtraction algorithm for fixed-point interpolation. To achieve this efficiency, the blocks use a data type of larger container size to perform the overflow-free subtraction, instead of using control-flow branches as in previous releases. Also, Real-Time Workshop generated code for fixed-point interpolation is now smaller.

Compatibility Considerations. Due to the change in the overflow-free subtraction algorithm, fixed-point interpolation in Lookup Table (n-D) and Interpolation Using Prelookup blocks might, in a few cases, introduce different rounding results from previous releases. Both simulation and code generation use the new overflow-free algorithm, so they have the same rounding behavior and provide bit-true consistency.

Expanded Support for Simplest Rounding Mode to Maximize Block Efficiency

In R2009a, support for the `Simplest` rounding mode has been expanded to enable more blocks to handle mixed floating-point and fixed-point data types:

- Abs

- Data Type Conversion Inherited

- Difference

- Discrete Derivative

- Discrete FIR Filter

- Discrete-Time Integrator
- Dot Product
- Fixed-Point State-Space
- Gain
- Index Vector
- Lookup Table (n-D)
- Math Function (for the `magnitude^2`, `reciprocal`, `square`, and `sqrt` functions)
- MinMax
- Multiport Switch
- Saturation
- Saturation Dynamic
- Sum
- Switch
- Transfer Fcn Direct Form II
- Transfer Fcn Direct Form II Time Varying
- Transfer Fcn First Order
- Transfer Fcn Lead or Lag
- Transfer Fcn Real Zero
- Weighted Sample Time
- Weighted Sample Time Math

For more information, see "Rounding Mode: Simplest" in the *Simulink® Fixed Point™ User's Guide*.

New Rounding Modes Added to Multiple Blocks
For the following Simulink blocks, the dialog box now displays `Convergent` and `Round` as possible rounding modes. These modes enable numerical agreement with embedded hardware and MATLAB results.

- Abs
- Data Type Conversion
- Data Type Conversion Inherited
- Difference
- Discrete Derivative
- Discrete FIR Filter
- Discrete-Time Integrator
- Divide
- Dot Product
- Fixed-Point State-Space
- Gain
- Index Vector
- Interpolation Using Prelookup
- Lookup Table
- Lookup Table (2-D)
- Lookup Table (n-D)
- Lookup Table Dynamic
- Math Function (for the `magnitude^2`, `reciprocal`, `square`, and `sqrt` functions)
- MinMax
- Multiport Switch
- Prelookup
- Product
- Product of Elements
- Saturation
- Saturation Dynamic
- Sum

- Switch

- Transfer Fcn Direct Form II

- Transfer Fcn Direct Form II Time Varying

- Transfer Fcn First Order

- Transfer Fcn Lead or Lag

- Transfer Fcn Real Zero

- Weighted Sample Time

- Weighted Sample Time Math

In the dialog box for these blocks, the field **Round integer calculations toward** has been renamed **Integer rounding mode**. The command-line parameter remains the same.

For more information, see "Rounding Mode: Convergent" and "Rounding Mode: Round" in the *Simulink Fixed Point User's Guide*.

Compatibility Considerations. If you use an earlier version of Simulink software to open a model that uses the Convergent or Round rounding mode, the mode changes automatically to Nearest.

Lookup Table (n-D) Block Performs Faster Calculation of Index and Fraction for Power of 2 Evenly-Spaced Breakpoint Data

For power of 2 evenly-spaced breakpoint data, the Lookup Table (n-D) block uses bit shifts to calculate the index and fraction, instead of division. This enhancement provides these benefits:

- Faster calculation of index and fraction for power of 2 evenly-spaced breakpoint data

- Smaller size of Real-Time Workshop generated code for the Lookup Table (n-D) block

Discrete FIR Filter Block Supports More Filter Structures

The following filter structures have been added to the Discrete FIR Filter block:

- Direct form symmetric

- Direct form antisymmetric

- Direct form transposed

- Lattice MA

Running a model with these filter structures requires a Signal Processing Blockset™ license.

Discrete Filter Block Performance, Data Type, Dimension, and Complexity Enhancements

The following enhancements have been made to the Discrete Filter block:

- Improved numerics and run-time performance of outputs and states by reducing the number of divide operations in the filter to at most one

- Support for signed fixed-point and integer data types

- Support for vector and matrix inputs

- Support for complex inputs and filter coefficients, where inputs and coefficients can each be real or complex, independently of the other

- A new **Initial states** parameter allows you to enter non-zero initial states

- A new **Leading denominator coefficient equals 1** parameter provides a more efficient implementation by eliminating all divides when the leading denominator coefficient is one

Compatibility Considerations. Due to these enhancements, you might encounter the compatibility issues in the following sections.

Realization parameter removed. The Real-Time Workshop software `realization` parameter has been removed from this block. You can no longer use the `set_param` and `get_param` functions on this block parameter. The generated code for this block has been improved to be similar to the former `'sparse'` realization, while maintaining tunability as in the former `'general'` realization.

State changes. Due to the reduction in the number of divide operations performed by the block, you might notice that your logged states have changed when the leading denominator coefficient is not one.

MinMax Block Performs More Efficient and Accurate Comparison Operations

For multiple inputs with mixed floating-point and fixed-point data types, the MinMax block selects an appropriate data type for performing comparison operations, instead of using the output data type for all comparisons, as in previous releases. This enhancement provides these benefits:

- Faster comparison operations, with fewer fixed-point overflows

- Smaller size of Real-Time Workshop generated code for the MinMax block

Logical Operator Block Supports NXOR Boolean Operator

In R2009a, the Logical Operator block has been enhanced with a new NXOR Boolean operator. When you select this operator, the block returns TRUE when an even number of inputs are TRUE. Similarly, the block returns FALSE when an even number of inputs are FALSE.

Use NXOR to replace serial XOR and NOT operations in a model.

Discrete-Time Integrator Block Uses Efficient Integration-Limiting Algorithm for Forward Euler Method

When you select the **Limit output** check box for the Forward Euler method, the Discrete-Time Integrator block uses only one saturation when a second saturation is unnecessary. This change in the integration-limiting algorithm provides these benefits:

- Faster integration

- Smaller size of Real-Time Workshop generated code for the Discrete-Time Integrator block

Dot Product Block Converted from S-Function to Core Block

Conversion of the Dot Product block from a masked S-Function to a core block enables more efficient simulation and better handling of the block in Simulink models.

Due to this conversion, you can specify sample time and values for the output minimum and maximum for the Dot Product block. The read-only **BlockType** parameter has also changed from S-Function to DotProduct.

Compatibility Considerations. In R2009a, signal dimension propagation might behave differently from previous releases. As a result, your model might not compile under these conditions:

• Your model contains a Dot Product block in a source loop.

• Your model has underspecified signal dimensions.

If your model does not compile, set dimensions for signals that are not fully specified.

For example, your model might not compile in this case:

• Your model contains a Transfer Fcn Direct Form II Time Varying block, which is a masked S-Function with a Dot Product block in a source loop.

• The second and third input ports of the Transfer Fcn Direct Form II Time Varying block are unconnected, which results in underspecified signal dimensions.

To ensure that your model compiles in this case, connect Constant blocks to the second and third input ports of the Transfer Fcn Direct Form II Time Varying block and specify the signal dimensions for both ports explicitly.

Pulse Generator Block Uses New Default Values for Period and Pulse Width

For the Pulse Generator block, the default **Period** value has changed from 2 to 10, and the default **Pulse Width** value has changed from 50 to 5. These changes enable easier transitions between time-based and sample-based mode for the pulse type.

Random Number and Unit Delay Blocks Use New Default Values for Sample Time

The default **Sample time** values for the Random Number and Unit Delay blocks have changed:

- The default **Sample time** value for the Random Number block has changed from 0 to 0.1.

- The default **Sample time** value for the Unit Delay block has changed from 1 to -1.

Trigonometric Function Block Provides Better Support of Accelerator Mode

The Trigonometric Function block now supports Accelerator mode for all cases with real inputs and Normal mode support. For more information about simulation modes, see "Accelerating Models" in the *Simulink User's Guide*.

Reshape Block Enhanced with New Input Port

The Reshape block **Output dimensionality** parameter has a new option, `Derive from reference input port`. This option creates a second input port, Ref, on the block and derives the dimensions of the output signal from the dimensions of the signal input to the Ref input port. Similarly, the Reshape block command-line parameter, `OutputDimensionality`, has the new option, `Derive from reference input port`.

Multidimensional Signals in Simulink Blocks

The following blocks were updated to support multidimensional signals. For more information, see "Signal Dimensions" in the *Simulink User's Guide*.

- Assertion

- Extract Bits

- Check Discrete Gradient

- Check Dynamic Gap

- Check Dynamic Lower Bound

- Check Dynamic Range

- Check Dynamic Upper Bound

- Check Input Resolution

- Check Static Gap

- Check Static Lower Bound

- Check Static Range

- Check Static Upper Bound

- Data Type Scaling Strip

- Wrap to Zero

Subsystem Blocks Enhanced with Read-Only Property That Indicates Virtual Status

The following subsystem blocks now have the property, `IsSubsystemVirtual`. This read-only property returns a Boolean value, `on` or `off`, to indicate if a subsystem is virtual.

- Atomic Subsystem

- Code Reuse Subsystem

- Configurable Subsystem

- Enabled and Triggered Subsystem

- Enabled Subsystem

- For Iterator Subsystem

- Function-Call Subsystem

- If Action Subsystem

- Subsystem

- Switch Case Action Subsystem

- Triggered Subsystem

- While Iterator Subsystem

User Interface Enhancements

Port Value Displays in Referenced Models
In R2009a, port value displays can appear for blocks in a Normal mode referenced model. To control port value displays, choose **View > Port Values** in the model window. For complete information about port value displays, see "Displaying Block Outputs".

Print Sample Time Legend
Print the Sample Time Legend either as an option of the block diagram print dialog box or directly from the legend. In either case, the legend will print on a separate sheet of paper. For more information, see "Print Sample Time Legend".

M-API for Access to Compiled Sample Time Information
New MATLAB API provides access to the compiled sample time data, color, and annotations for a specific block or the entire block diagram directly from M code.

Model Advisor Report Enhancements
In R2009a, the Model Advisor report is enhanced with:

- The ability to save the report to a location that you specify.
- Improved readability, including the ability to:
  - Filter the report to view results according to the result status. For example, you can now filter the report to show errors and warnings only.
  - Collapse and expand the folder view in the report.
  - View a summary of results for each folder in the report.

See "Consulting the Model Advisor" in the Simulink User's Guide.

Counterclockwise Block Rotation
This release lets you rotate blocks counterclockwise as well as clockwise (see "How to Rotate a Block" for more information).

Physical Port Rotation for Masked Blocks

This release lets you specify that the ports of a masked block not be repositioned after a clockwise rotation to maintain a left-to-right and top-to-bottom numbering of the ports. This enhancement facilitates use of masked blocks in mechanical systems, hydraulic systems, and other modeling applications where block diagrams do not have a preferred orientation (see "Port Rotation Type" for more information.)

Smart Guides

In R2009a, when you drag a block, Simulink draws lines, called smart guides, that indicate when the block's ports, center, and edges align with the ports, centers, and edges of other blocks in the same diagram. This helps you create well-laid-out diagrams (see "Smart Guides" for more information).

Customizing the Library Browser's User Interface

Release 2009a lets you customize the Library Browser's user interface. You can change the order in which libraries appear in the Library Browser, disable or hide libraries, sublibraries, and blocks, and add, disable, or hide items on the Library Browser's menus. See "Customizing the Library Browser" for more information.

Subsystem Creation Command

This release adds a command, `Simulink.BlockDiagram.createSubSystem`, that creates a subsystem from a specified group of blocks.

S-Functions

Level-1 Fortran S-Functions

In this release, if you attempt to compile or simulate a model with a Level-1 Fortran S-function, you will receive an error due to the use of the newly deprecated function 'MXCREATEFULL' within the Fortran S-function wrapper 'simulink.F'. If your S-function does not explicitly use 'MXCREATEFULL', simply recompile the S-function. If your S-function uses 'MXCREATEFULL', replace each instance with 'MXCREATEDOUBLEMATRIX' and recompile the S-function.

Removal of Lookup Table Designer from the Lookup Table Editor

In R2009a, the Lookup Table Designer is no longer available in the Lookup Table Editor.

Compatibility Considerations

Previously, you could select **Edit > Design Table** in the Lookup Table Editor to launch the Lookup Table Designer. In R2009a, this menu item is no longer available.

Version 7.2 (R2008b) Simulink Software

This table summarizes what's new in V7.2 (R2008b):

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

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

Simulation Performance

Parallel Simulations in Rapid Accelerator Mode
Simulink now has the capability to run parallel simulations in Rapid Accelerator mode using `parfor` on prebuilt Simulink models.

You can now run parallel simulations in Rapid Accelerator mode with different external inputs and tunable parameters. The `sim` command can be called from a `parfor` loop if the model does not require a rebuild.

For more information, see "Running a Simulation Programmatically".

Improved Rebuild Mechanism in Rapid Accelerator Mode

Simulink now has enhanced tuning of the solver and logging parameters in Rapid Accelerator mode without requiring a rebuild.

An improved rebuild mechanism ensures that the model does not rebuild when you change block diagram parameters (e.g., stop time, solver tolerances, etc.). This enhancement significantly decreases the time for simulation in Rapid Accelerator mode.

Data Type Size Limit on Accelerated Simulation Removed

In previous releases, accelerated simulation was not supported for models that use integer or fixed-point data types greater than 32 bits in length. In this release, the acceleration limit on integer and fixed-point data type size has increased to 128 bits, the same as the limit for normal-mode, i.e., unaccelerated simulation.

New Initialization Behavior in Conditional, Action, and Iterator Subsystems

For releases prior to 2008b, at the simulation start time, Simulink initializes all blocks unconditionally and subsystems cannot reset the states. Release 2008b introduces behavior that mirrors the behavior of Real-Time Workshop. For normal simulation mode, the Simulink block initialization method (mdlInitializeConditions) can be called more than once at the start time if:

- The block is contained within a Conditional, Action, or Iterator subsystem.

- The subsystem is configured to reset states when enabled (or triggered); and the subsystem is enabled (or triggered) at the start time.

This new initialization behavior has the following effect on S-functions:

- If you need to ensure that the initialization code in the mdlInitializeConditions function runs only once, then move this initialization code into the mdlStart method. The MathWorks recommends this code change as a best practice.

- The change to the block initialization method, as described above, exposed a bug in the S-function macro ssIsFirstInitCond for applications involving an S-function within a Conditional, Action or Iterator subsystem. This bug has been fixed in R2008b.

  To determine if you consequently need to update your Simulink S-functions for compatibility, compare the simulation results from R2007b or an earlier release with those of R2008b. If they differ at the start time, ssIsFirstInitCond is running more than once and you must regenerate and recompile the appropriate Simulink S-functions.

  For Real-Time Workshop, you must regenerate and recompile all S-function targets and any Real-Time Workshop target for which the absolute time is turned on. (If a third-party vendor developed your S-functions, have the vendor regenerate and recompile them for you. The vendor can use the SLDiagnostics feature to identify all S-functions in a model.)

Component-Based Modeling

Processor-in-the-Loop Mode in Model Block

In R2008b, Simulink has a new Model block simulation mode for processor-in-the-loop (PIL) verification of generated code. This feature requires Real-Time Workshop Embedded Coder software. The feature lets you test the automatically generated and cross-compiled object code on your embedded processor by easily switching between Normal, Accelerator, and PIL simulation modes in your original model. You can reuse test suites, resulting in faster iteration between model development and generated code verification. For more information, see "Referenced Model Simulation Modes".

Conditionally Executed Subsystem Initial Conditions

R2008b of Simulink includes enhanced handling of initial conditions for conditionally executed subsystems, Merge blocks, and Discrete-Time Integrator blocks, improving consistency of simulation results.

This feature allows you to select simplified initialization mode for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks. The simplified initialization improves the consistency of simulation results, especially for models that do not specify initial conditions for conditional subsystem output ports, and for models that have conditionally executed subsystem output ports connected to S-functions.

Note To use the new simplified initialization mode, you must activate this feature.

Activating This Feature for New Models. For new models, you can activate this feature as follows:

1 In the model window, select **Simulation > Configuration Parameters**.

   The Configuration Parameters dialog box opens.

2 Select **Diagnostics > Data Validity**.

   The Data Validity Diagnostics pane opens.

3 In the Model Initialization section, set **Underspecified initialization detection** to Simplified.

4 Select **Diagnostics > Connectivity**.

   The Connectivity Diagnostics pane opens.

5 Set **Mux blocks used to create bus signals** to error.

6 Set **Bus signal treated as vector** to error.

7 Click **OK**.

For more information, see "Underspecified initialization detection".

Migrating Existing Models. For existing models, The MathWorks recommends using the Model Advisor to migrate your model to the new simplified initialization mode settings.

To migrate an existing model:

1 In the model window, select **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box opens.

2 Select **Diagnostics > Data Validity**.

The Data Validity Diagnostics pane opens.

3 In the Merge Block section, set **Detect multiple driving blocks executing at the same time step** to error.

4 Click **OK**.

5 Simulate the model and ensure that it runs without errors.

6 Select **Tools > Model Advisor**.

The Model Advisor opens.

7 In the Model Advisor Task Manager, select **By Product > Simulink**.

8 Run **Check for proper bus usage** in the Model Advisor.

9 Run **Check consistency of initialization parameters for Outport and Merge blocks** in the Model Advisor.

10 After you have resolved any errors identified by this check, click **Proceed** to migrate your model to simplified initialization mode.

For information on using the Model Advisor, see "Consulting the Model Advisor" in the *Simulink User's Guide*.

For information on the Model Advisor checks, see "Check consistency of initialization parameters for Outport and Merge blocks" in the *Simulink Reference*.

Compatibility Considerations. Activating this feature can cause differences in simulation results, when compared to previous versions. Since you must opt-in to this feature before any changes are made, there are no issues for existing models. However, The MathWorks recommends that you backup existing models before you migrate them, in case you want to return to the original behavior.

Model Block Input Enhancement

Model block inputs can now be local and reusable. This capability reduces global data usage and data copying when interfacing with code from a referenced model, which can reduce memory usage during simulation and increase the efficiency of generated code. This enhancement is always relevant, so no configuration parameter is necessary or provided to control it.

One Parameter Controls Accelerator Mode Build Verbosity

In previous releases, the `ModelReferenceSimTargetVerbose` parameter controlled verbosity when a referenced model was built for execution in Accelerator mode, as specified by the Model block's Simulation mode parameter. The `ModelReferenceSimTargetVerbose` had no GUI equivalent. See "Referenced Model Simulation Modes" and the Model block documentation for more information.

A different parameter, `AccelVerboseBuild`, controls the verbosity when a model is built in Simulink Accelerator mode or Rapid Accelerator mode, as specified in the **Simulation** menu. See "Accelerating Models" for more information. The GUI equivalent of the `AccelVerboseBuild` parameter is **Configuration Parameters > Optimization > Verbose accelerator builds**. See "Verbose accelerator builds" for more information.

All types of accelerated simulation entail code generation (though the code is not visible to the user) and the two verbosity parameters control whether a detailed account of the code generation process appears in the MATLAB Command Window. However, providing separate verbosity parameters for the two cases was unnecessary.

In R2008b, the `ModelReferenceSimTargetVerbose` parameter is deprecated and has no effect. The `AccelVerboseBuild` parameter (**Configuration Parameters > Optimization > Verbose accelerator builds**) now controls

the verbosity for Simulink Accelerator mode, referenced model Accelerator mode, and Rapid Accelerator mode.

Another parameter, `RTWVerbose` (**Configuration Parameters > Real-Time Workshop > Debug > Verbose build**) controls the verbosity of Real-Time Workshop code generation. This parameter is unaffected by the changes to `ModelReferenceSimTargetVerbose` and `AccelVerboseBuild`.

Compatibility Considerations. In R2008b, trying to set `ModelReferenceSimTargetVerbose` generates a warning message and has no effect on verbosity. The warning says to use `AccelVerboseBuild` instead. The default for `AccelVerboseBuild` is `'off'`.

A model saved in R2008b will not include the `ModelReferenceSimTargetVerbose` parameter. An R2008b model saved to an earlier Simulink version that supports `ModelReferenceSimTargetVerbose` will include that parameter, giving it the same value that `AccelVerboseBuild` has in the R2008b version.

The effect of loading a model from an earlier Simulink version into R2008b depends on the source version:

- **Prior to R14:** Neither parameter exists, so no compatibility consideration arises.
- **R14 – R2006b:** Only `ModelReferenceSimTargetVerbose` exists. Copy its value to `AccelVerboseBuild`.
- **R2007a:** Both parameters exist but neither has a GUI equivalent. Ignore the value of `ModelReferenceSimTargetVerbose` and post no warning.
- **R2007b – R2008a:** Both parameters exist and `AccelVerboseBuild` and has a GUI equivalent. If `ModelReferenceSimTargetVerbose` is `'on'`, post a warning to use `AccelVerboseBuild` instead.

Embedded MATLAB Function Blocks

Support for Fixed-Point Word Lengths Up to 128 Bits

Embedded MATLAB Function blocks now support up to 128 bits of fixed-point precision. This increase in maximum precision from 32 to 128 bits supports

generating efficient code for targets with non-standard word sizes and allows
Embedded MATLAB Function blocks to work with large fixed-point signals.

Enhanced Simulation and Code Generation Options for Embedded MATLAB Function Blocks

You can now specify embeddable code generation options from the Embedded
MATLAB Editor using a new menu item: **Tools > Open RTW Target**.
Simulation options continue to be available from **Tools > Open Simulation
Target**.

In addition, simulation and embeddable code generation options now appear
in a single dialog box. For details, see "Unified Simulation and Embeddable
Code Generation Options" on page 95.

Data Type Override Now Works Consistently on Outputs

When you enable data type override for Embedded MATLAB Function blocks,
outputs with explicit and inherited types are converted to the override type.
For example, if you set data type override to `true singles`, the Embedded
MATLAB Function block converts all outputs to `single` type and propagates
the override type to downstream blocks.

In previous releases, Embedded MATLAB Function blocks did not apply data
type override to outputs with inherited types. Instead, the inherited type
was preserved even if it did not match the override type, sometimes causing
errors during simulation.

Compatibility Consideration. Applying data type override rules to outputs
with inherited types may introduce the following compatibility issues:

• Downstream Embedded MATLAB Function blocks must be able to accept
  the propagated override type. Therefore, you must allow data type override
  for downstream blocks for which you set output type explicitly. Otherwise,
  you may not be able to simulate your model.

• You might get unexpected simulation results if the propagated type uses
  less precision than the original type.

Improperly-Scaled Fixed-Point Relational Operators Now Match MATLAB Results

When evaluating relational operators, Embedded MATLAB Function blocks compute a common type that encompasses both input operands. In previous releases, if the common type required more than 32 bits, Embedded MATLAB Function blocks may have given different answers from MATLAB. Now, Embedded MATLAB Function blocks give the same answers as MATLAB.

Compatibility Consideration. Some relational operators generate multi-word code even if one of the fixed-point operands is not a multi-word value. To work around this issue, cast both operands to the same fixed-point type (using the same scaling method and properties).

Data Management

Support for Enumerated Data Types

Simulink models now support enumerated data types. For details, see:

- "Using Enumerated Data" in the Simulink User's Guide

- "Using Enumerated Data in Stateflow Charts" in the Stateflow User's Guide

- "Enumerated Data Type Considerations" in the Real-Time Workshop User's Guide

Simulink Bus Editor Enhancements

The Simulink Bus Editor can now filter displayed bus objects by either name or relationship. See "Filtering Displayed Bus Objects" for details.

You can now fully customize the export and import capabilities of the Simulink Bus Editor. See "Customizing Bus Object Import and Export" for details.

New Model Advisor Check for Proper Data Store Memory Usage

A new Model Advisor check posts advice and warnings about the proper use of Data Store Memory, Data Store Read, and Data Store Write blocks.

See "Check Data Store Memory blocks for multitasking, strong typing, and shadowing issues" for details.

Simulink File Management

Model Dependencies Tools

Enhanced file dependency analysis can now:

- Find system target files

- Analyze `STF_make_rtw_hook` functions

- Analyze all configuration sets, not just the active set.

See "Scope of Dependency Analysis" in the Simulink User's Guide.

Block Enhancements

Trigonometric Function Block

R2008b provides an enhanced Trigonometric Function block to:

- Support sincos

- Provide greater floating-point consistency

Math Function Block

In Simulink 2008b, an enhanced Math Function block provides greater floating-point consistency.

Merge Block

R2008b provides enhanced handling of initial conditions for the Merge block and thus improves the consistency of simulation results.

For more information, see "Conditionally Executed Subsystem Initial Conditions" on page 80.

Discrete-Time Integrator Block

R2008b provides an enhanced handling of initial conditions for the Discrete-Time Integrator block and thereby improves the consistency of simulation results.

For more information, see "Conditionally Executed Subsystem Initial Conditions" on page 80.

Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors

In this release, Simulink software can signal an error if a block callback function, e.g., `CopyFcn`, modifies a link to a library block. For example, an error occurs if you attempt to copy a library link to a self-modifying masked subsystem whose `CopyFcn` deletes a block contained by the subsystem. This change means that you cannot use block callback functions to create self-modifying library blocks. Mask initialization code for a library block is the only code allowed to modify the block.

Compatibility Consideration. Previous releases allowed use of block callback functions to create self-modifying library blocks. Opening, editing, or running models that contain links to such blocks can cause illegal modification errors in the current release. As a temporary work around, you can break any links in your model to a library block that uses callback functions to modify itself. The best long-term solution is to move the self-modification code to the block's mask initialization section.

Random Number Block

In the dialog box for the Random Number block, the field **Initial Seed** has been renamed **Seed**. The command-line parameter remains the same.

Signal Generator Block

The Signal Generator block now supports multidimensional signals. For a list of blocks that support multidimensional signals, see "Signal Dimensions" in the *Simulink User's Guide*.

Sum Block

The accumulator of the Sum block now applies for all input signals of any data type (for example, double, single, integer, and fixed-point). In previous releases, the accumulator of this block was limited to inputs and outputs of only integer or fixed-point data types.

Switch Block

The Switch block now supports the immediate back propagation of a known output data type to the first and third input ports. This occurs when you set the **Output data type** parameter to `Inherit:  Inherit via internal rule` and select the **Require all data port inputs to have the same data type** check box. In previous releases, this back propagation did not occur immediately.

Uniform Random Number Block

In the dialog box for the Uniform Random Number block, the field **Initial Seed** has been renamed **Seed**. The command-line parameter remains the same.

User Interface Enhancements

Sample Time

The display of sample time information has been expanded to include:

- Signal lines labeling with new color-independent **Annotations**

- A new **Sample Time Legend** maps the sample time **Colors** and **Annotations** to sample times.

- A distinct color for indicating that a block and signal are asynchronous.

The section "Modeling and Simulation of Discrete Systems" has been renamed "Working with Sample Times" and has been significantly expanded to provide a comprehensive review of sample times and a discussion on the new Sample Time Legend and Sample Time Display features. For more information, see "Working with Sample Times".

Model Advisor

In R2008b, the Model Advisor is enhanced with:

- A model and data restore point that provides you with the ability to revert changes made in response to advice from the Model Advisor

- Context-sensitive help available for Model Advisor checks

- Tristate check boxes that visually indicate selected and cleared checks in folders

- A system selector for choosing the system level that the Model Advisor checks

See "Consulting the Model Advisor" in the Simulink User's Guide.

"What's This?" Context-Sensitive Help for Commonly Used Blocks

R2008b introduces context-sensitive help for parameters that appear in the following commonly used blocks in Simulink:

Bus Creator
Bus Selector
Constant
Data Type Conversion
Demux
Discrete-Time Integrator
Gain
Inport
Integrator
Logical Operator
Mux
Outport
Product
Relational Operator
Saturation
Subsystem
Sum
Switch

Terminator
Unit Delay

This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

1 Place your cursor over the label of a parameter.

2 Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after right-clicking the **Multiplication** parameter for the Gain block.
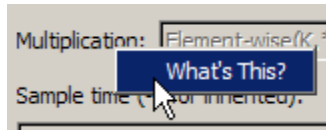
3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

Compact Icon Option Displays More Blocks in Library Browser

This release introduces a compact icon option that maximizes the number of blocks and libraries visible in the Library Browser's **Library** pane without scrolling (see "Library Pane").

Signal Logging and Test Points Are Controlled Independently

In previous releases, a signal could be logged only if it was also a test point. Therefore, selecting **Log signal data** in the Signal Properties dialog box automatically selected **Test point**, and disabled it so that it could not be cleared. However, a signal can be a test point without being logged, so clearing **Log signal data** did not automatically clear **Test point**. The same asymmetric behavior occurred programmatically with the underlying DataLogging and TestPoint parameters.

In R2008b, no connection exists between enabling logging for a signal and making the signal a test point. Either, both, or neither capability can be enabled for any signal. Selecting and clearing **Log signal data** therefore has no effect on the setting of **Test point**, and similarly for the underlying parameters. See "Logging Signals" and "Working with Test Points" for more information.

To reflect the independence of logging and test points, the command **Test Point Indicators** in the Simulink **Format > Port/Signal Displays** menu has been renamed **Testpoint/Logging Indicators**. The effect of the command, the graphical indicators displayed, and the meaning of the underlying parameter ShowTestPointIcons, are all unchanged.

Compatibility Considerations. Scripts and practices that relied on **Log signal data** to automatically set a test point must be changed to set the test point explicitly. The relevant set_param commands are:

```
set_param(PortHandle(n),'DataLogging','on')
set_param(PortHandle(n),'TestPoint','on')
```

To disable either capability, set the relevant parameter to 'off'. See "Enabling Signal Logging Programmatically" for an example.

Signal Logging Consistently Retains Duplicate Signal Regions

A *virtual signal* is a signal that graphically represents other signals or parts of other signals. Virtual signals are purely graphical entities; they have no functional or mathematical significance. The nonvirtual components of a virtual signal are called *regions*. For example, if Mux block (which is a virtual block) inputs two nonvirtual signals, the block outputs a virtual signal that has two regions. See "Virtual Signals" and "Mux Signals" for more information.

In previous releases, when a virtual signal contains duplicate regions, signal logging excluded all but one of the duplicates in some contexts, but included all of the duplicates in other contexts, giving inconsistent results. For example, if the same nonvirtual signal is connected to two input ports of a Mux block, that one signal is the source of two regions in the Mux block output. Previously, if that output was being logged in Normal mode simulation, the log object would contain data for only one of the regions, because the other was eliminated as a duplicate.

In R2008a, Simulink no longer eliminates duplicate regions when logging the output of virtual blocks like Mux or Selector blocks. Simulink now logs all regions, which appear in a `Simulink.TsArray` object. The duplicate regions have unique names as follows:

```
<signal_name>_reg<#counter>
```

This change affects signal logs and all capabilities that depend on signal logging, such as scopes and signal viewers.

Compatibility Considerations. In cases where signal logging previously omitted duplicate regions, signal logs will now be larger, and scopes and signal viewers will now show more data. This change could give the impression that the results of simulation have changed, but actually only the logging of those results has changed. No action is needed unless:

- A dependency exists on the exact size of a log or the details of its contents.

- The size and details have changed due to the inclusion of previously omitted signals.

In such a case, make changes as needed to accept the changed logging behavior. See "Logging Signals" for more information.

Simulink Configuration Parameters

In R2008b, the following Simulink configuration parameters are updated:

Note The command-line parameter name is not changing for these parameters.

Location	Previous Parameter	New Parameter
Solver	**States shape preservation** / `ShapePreserveControl`	**Shape preservation** / `ShapePreserveControl`
Solver	**Consecutive min step size violations** / `MaxConsecutiveMinStep`	**Number of consecutive min steps** / `MaxConsecutiveMinStep`

Location	Previous Parameter	New Parameter
Solver	**Consecutive zero crossings relative tolerance** / `ConsecutiveZCsStepRelTol`	**Time tolerance** / `ConsecutiveZCsStepRelTol`
Solver	**Zero crossing location algorithm** / `ZeroCrosAlgorithm`	**Algorithm** / `ZeroCrosAlgorithm`
Solver	**Zero crossing location threshold** / `ZCThreshold`	**Signal threshold**/ `ZCThreshold`
Solver	**Number of consecutive zero crossings allowed** / `MaxConsecutiveZCs`	**Number of consecutive zero crossings** / `MaxConsecutiveZCs`
Optimization	**Eliminate superfluous temporary variables (Expression folding)** / `ExpressionFolding`	**Eliminate superfluous local variables (Expression folding)**/ `ExpressionFolding`
Optimization	**Remove internal state zero initialization** / `ZeroInternalMemoryAtStartup`	**Remove internal data zero initialization** / `ZeroInternalMemoryAtStartup`

In R2008b, the following Simulink configuration parameters have moved:

Note The command-line parameter name is not changing for these parameters.

Parameter	Old Location	New Location
Check undefined subsystem initial output	**Diagnostics > Compatibility**	**Diagnostics > Data Validity**
Check preactivation output of execution context	**Diagnostics > Compatibility**	**Diagnostics > Data Validity**
Check runtime output of execution context	**Diagnostics > Compatibility**	**Diagnostics > Data Validity**

In R2008b, the **Optimization > Minimize array reads using temporary variables** parameter has been obsoleted.

Model Help Menu Update

The Simulink model **Help** menu now includes links to block support tables for the following products, if they are installed.

- Simulink product
- Communications Blockset™
- Signal Processing Blockset
- Video and Image Processing Blockset™

To obtain the block support tables for all of these products that are installed, select **Help > Block Support Table > All Tables**.

In previous releases, **Help > Block Support Table** provided such tables only for the main Simulink library.

Unified Simulation and Embeddable Code Generation Options

You can now specify both simulation and embeddable code generation options in the Configuration Parameters dialog box. The simulation options apply only to Embedded MATLAB Function blocks, Stateflow charts, and Truth Table blocks.
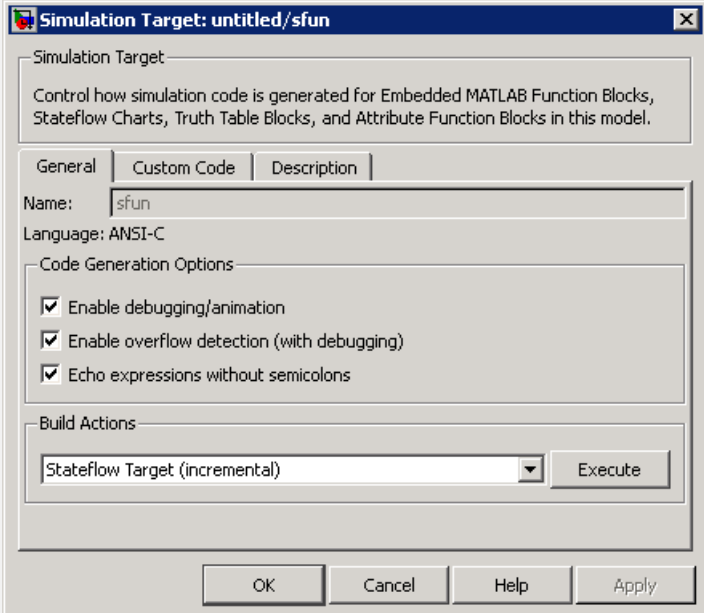
The following table summarizes changes that apply for Embedded MATLAB Function blocks:
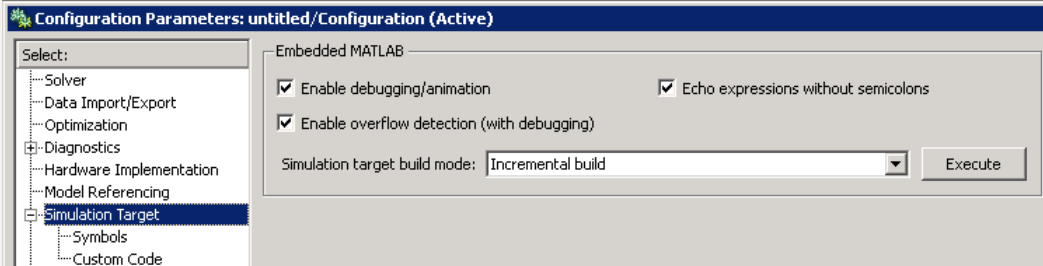
Type of Model	Simulation Options	Embeddable Code Generation Options
Nonlibrary	Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box. See: • "Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box" on page 97 • "Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box" on page 99 • "Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box" on page 100	New menu item in the Embedded MATLAB Editor for specifying code generation options for nonlibrary models: **Tools > Open RTW Target** New options in the **Real-Time Workshop** pane of the Configuration Parameters dialog box. See: • "Nonlibrary Models: Enhancement for the **Real-Time Workshop**: **Symbols** Pane of the Configuration Parameters Dialog Box" on page 109 • "Nonlibrary Models: Enhancement for the **Real-Time Workshop**: **Custom Code** Pane of the Configuration Parameters Dialog Box" on page 110
Library	Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box. See: • "Library Models: Changes for the General Pane of the Simulation Target Dialog Box" on page 104 • "Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box" on page 105 • "Library Models: Changes for the Description Pane of the Simulation Target Dialog Box" on page 106	New menu item in Embedded MATLAB Editor for specifying custom code generation options for library models: **Tools > Open RTW Target** For a description of these options, see "Library Models: Support for Specifying Custom Code Options in the Real-Time Workshop Pane of the Configuration Parameters Dialog Box" on page 110.

For details about the new options, see "Configuration Parameters Dialog Box" in the *Simulink Graphical User Interface* documentation. For compatibility information, see "Compatibility Considerations" on page 116.

For changes specific to Stateflow, see "Unified Simulation and Embeddable Code Generation Options for Stateflow Charts and Truth Table Blocks" in the Stateflow and Stateflow® Coder™ release notes.
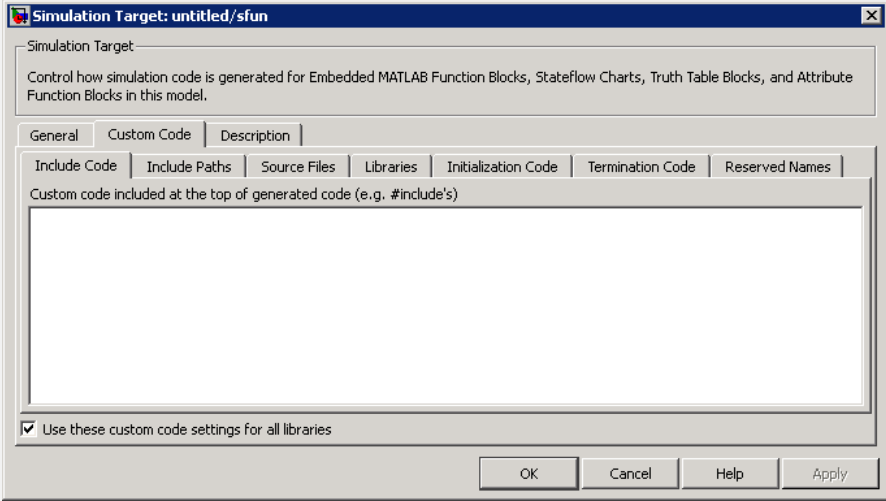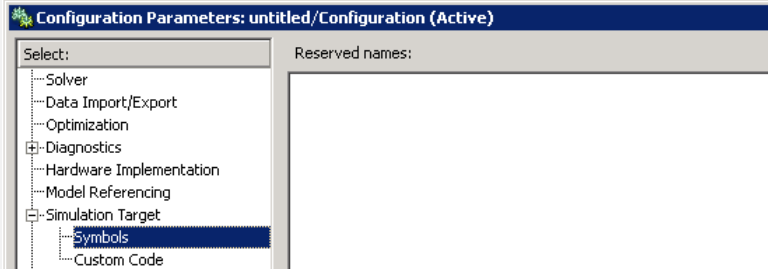
Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box. The following sections describe changes in the panes of the Simulation Target dialog box for nonlibrary models.
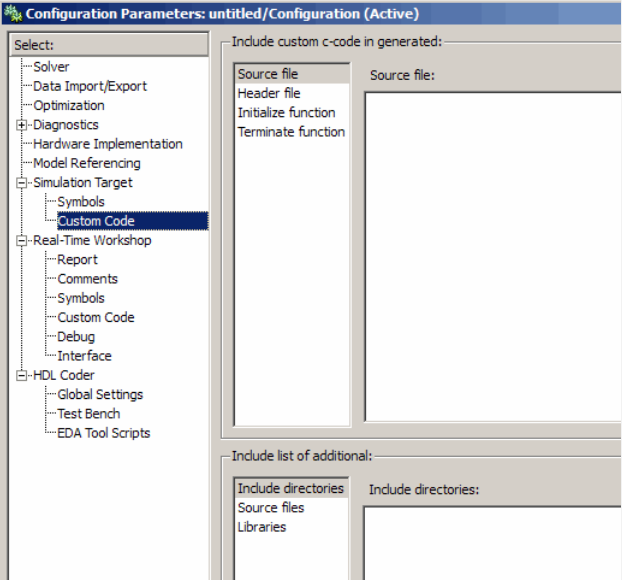
Release	Appearance
Previous	**General** pane of the Simulation Target dialog box

Release	Appearance
New	**Simulation Target** pane of the Configuration Parameters dialog box

For details, see "Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box" on page 101.

Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box.

Release	Appearance
Previous	**Custom Code** pane of the Simulation Target dialog box
New	**Simulation Target > Symbols** pane of the Configuration Parameters dialog box

Release	Appearance
New	**Simulation Target > Custom Code** pane of the Configuration Parameters dialog box

For details, see "Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box" on page 101.

Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box. In previous releases, the **Description** pane of the Simulation Target dialog box appeared as follows.

In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is now accessible only in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.

Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box.

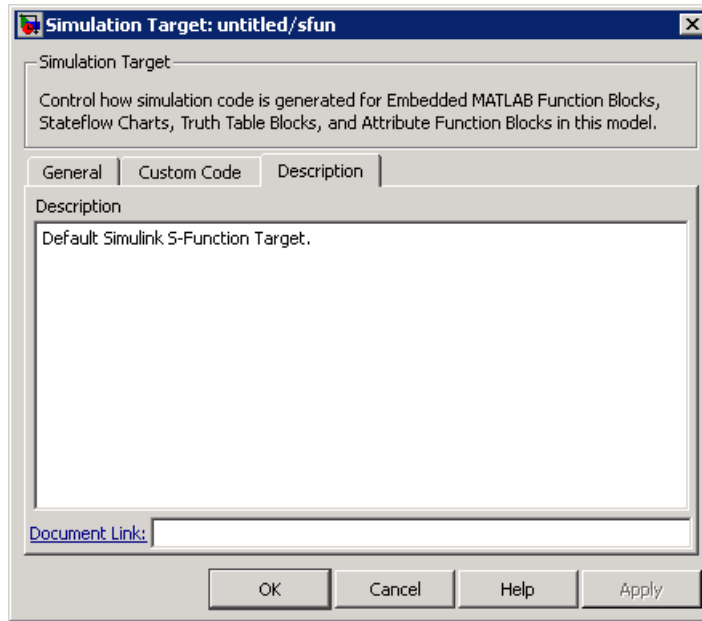For nonlibrary models, the following table maps each GUI option in the Simulation Target dialog box to the equivalent in the Configuration Parameters dialog box. The options are listed in order of appearance in the Simulation Target dialog box.

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	**Simulation Target > Enable debugging / animation**	on
General > Enable overflow detection (with debugging)	**Simulation Target > Enable overflow detection (with debugging)**	on
General > Echo expressions without semicolons	**Simulation Target > Echo expressions without semicolons**	on
General > Build Actions	**Simulation Target > Simulation target build mode**	Incremental build
None	**Simulation Target > Custom Code > Source file**	''
Custom Code > Include Code	**Simulation Target > Custom Code > Header file**	''
Custom Code > Include Paths	**Simulation Target > Custom Code > Include directories**	''
Custom Code > Source Files	**Simulation Target > Custom Code > Source files**	''
Custom Code > Libraries	**Simulation Target > Custom Code > Libraries**	''
Custom Code > Initialization Code	**Simulation Target > Custom Code > Initialize function**	''
Custom Code > Termination Code	**Simulation Target > Custom Code > Terminate function**	''
Custom Code > Reserved Names	**Simulation Target > Symbols > Reserved names**	{}

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
Custom Code > Use these custom code settings for all libraries	None	Not applicable
Description > Description	None **Note** If you load an older model that contained user-specified text in the **Description** field, that text now appears in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.	Not applicable
Description > Document Link	None	Not applicable
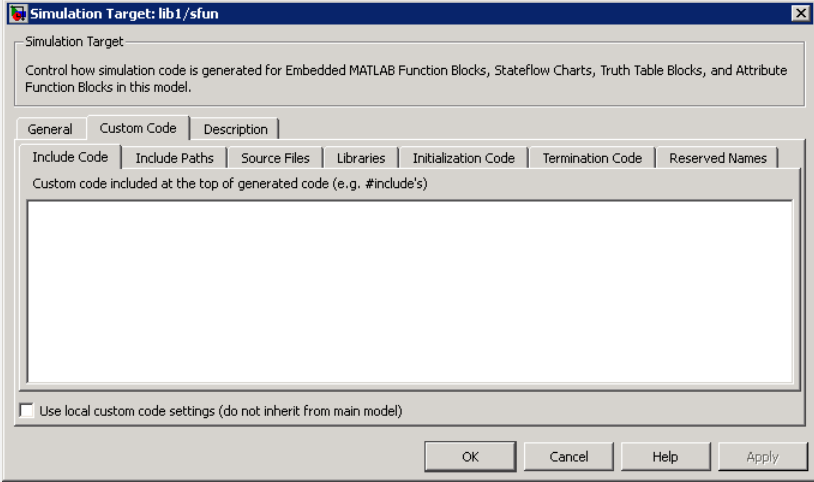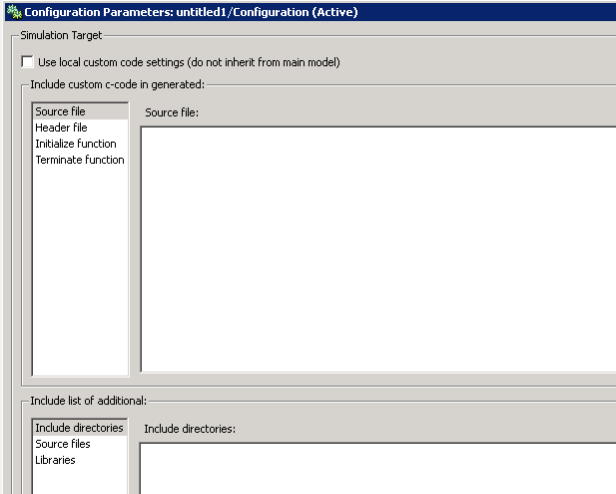
Note For nonlibrary models, **Simulation Target** options in the Configuration Parameters dialog box are also available in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, you can select **Simulation Target** in the **Contents** pane to access the options.

Library Models: Changes for the General Pane of the Simulation Target Dialog Box. In previous releases, the **General** pane of the Simulation Target dialog box for library models appeared as follows.
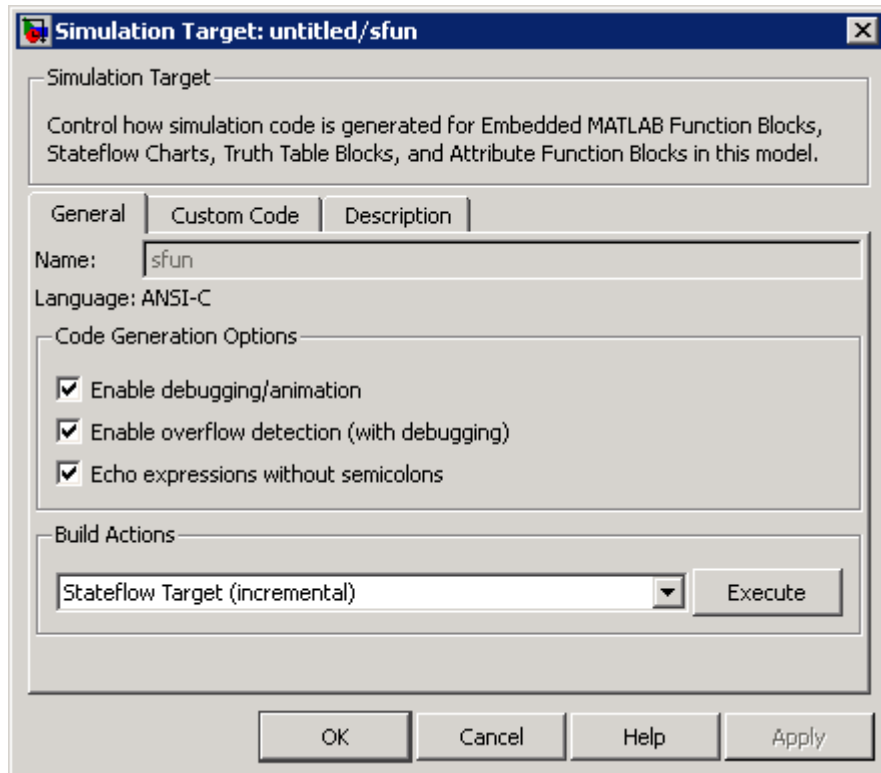
In R2008b, these options are no longer available. All library models inherit these option settings from the main model to which the libraries are linked.

Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box.

Release	Appearance
Previous	**Custom Code** pane of the Simulation Target dialog box
New	**Simulation Target** pane of the Configuration Parameters dialog box

For details, see "Library Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box" on page 107.

Library Models: Changes for the Description Pane of the Simulation Target Dialog Box. In previous releases, the **Description** pane of the Simulation Target dialog box appeared as follows.

In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is discarded.

Library Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box. For library models, the following table maps each GUI option in the Simulation Target dialog box to the equivalent in the Configuration Parameters dialog box. The options are listed in order of appearance in the Simulation Target dialog box.

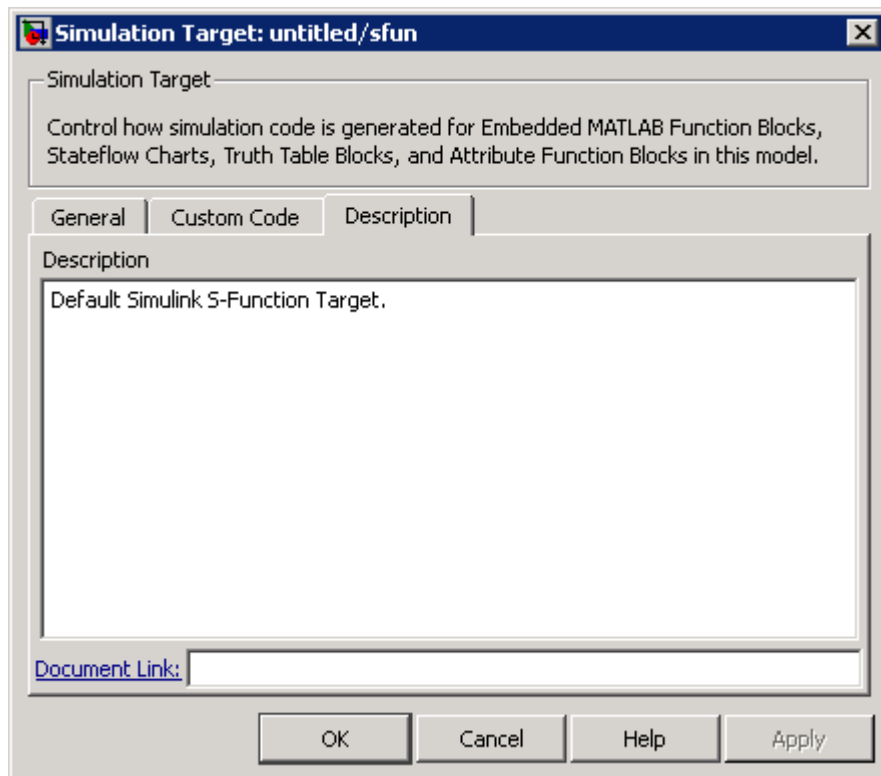| Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	None	Not applicable
General > Enable overflow detection (with debugging)	None	Not applicable
General > Echo expressions without semicolons	None	Not applicable
General > Build Actions	None	Not applicable
None	**Simulation Target > Source file**	' '
Custom Code > Include Code	**Simulation Target > Header file**	' '
Custom Code > Include Paths	**Simulation Target > Include directories**	' '
Custom Code > Source Files	**Simulation Target > Source files**	' '
Custom Code > Libraries	**Simulation Target > Libraries**	' '
Custom Code > Initialization Code	**Simulation Target > Initialize function**	' '
Custom Code > Termination Code	**Simulation Target > Terminate function**	' '
Custom Code > Reserved Names	None	Not applicable

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
Custom Code > Use local custom code settings (do not inherit from main model)	**Simulation Target > Use local custom code settings (do not inherit from main model)**	`off`
Description > Description	None	Not applicable
Description > Document Link	None	Not applicable

Note For library models, **Simulation Target** options in the Configuration Parameters dialog box are not available in the Model Explorer.

Nonlibrary Models: Enhancement for the Real-Time Workshop: Symbols Pane of the Configuration Parameters Dialog Box. In previous releases, the **Real-Time Workshop** > **Symbols** pane of the Configuration Parameters dialog box appeared as follows.
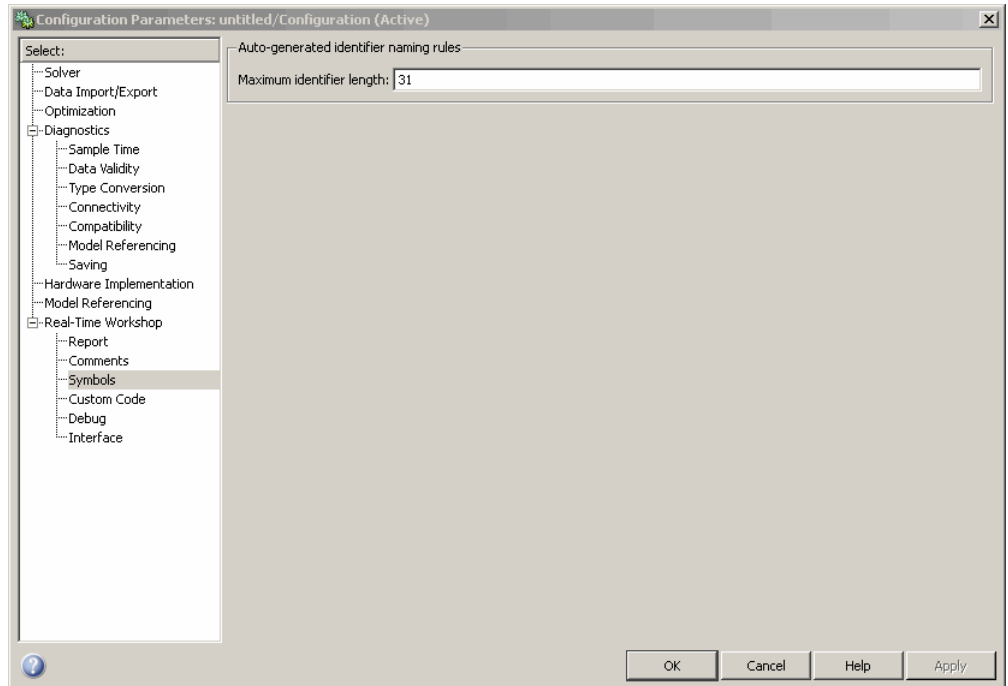
In R2008b, a new option is available in this pane: **Reserved names**. You can use this option to specify a set of keywords that the Real-Time Workshop build process should not use. This action prevents naming conflicts between functions and variables from external environments and identifiers in the generated code.

You can also choose to use the reserved names specified in the **Simulation Target** > **Symbols** pane to avoid entering the same information twice for the nonlibrary model. Select the option **Use the same reserved names as Simulation Target**.

Nonlibrary Models: Enhancement for the Real-Time Workshop: Custom Code Pane of the Configuration Parameters Dialog Box. In previous releases, the **Real-Time Workshop** > **Custom Code** pane of the Configuration Parameters dialog box appeared as follows.
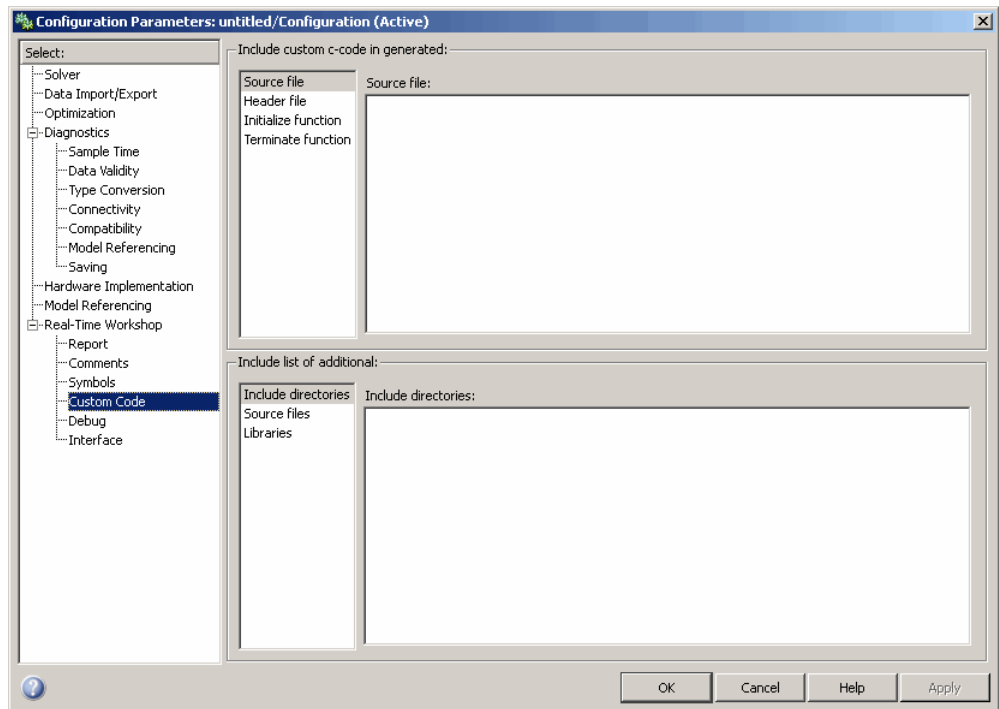
In R2008b, a new option is available in this pane: **Use the same custom code settings as Simulation Target**. You can use this option to copy the custom code settings from the **Simulation Target** > **Custom Code** pane to avoid entering the same information twice for the nonlibrary model.

Library Models: Support for Specifying Custom Code Options in the Real-Time Workshop Pane of the Configuration Parameters Dialog Box. In R2008b, you can specify custom code options in the Configuration Parameters dialog box, as shown:
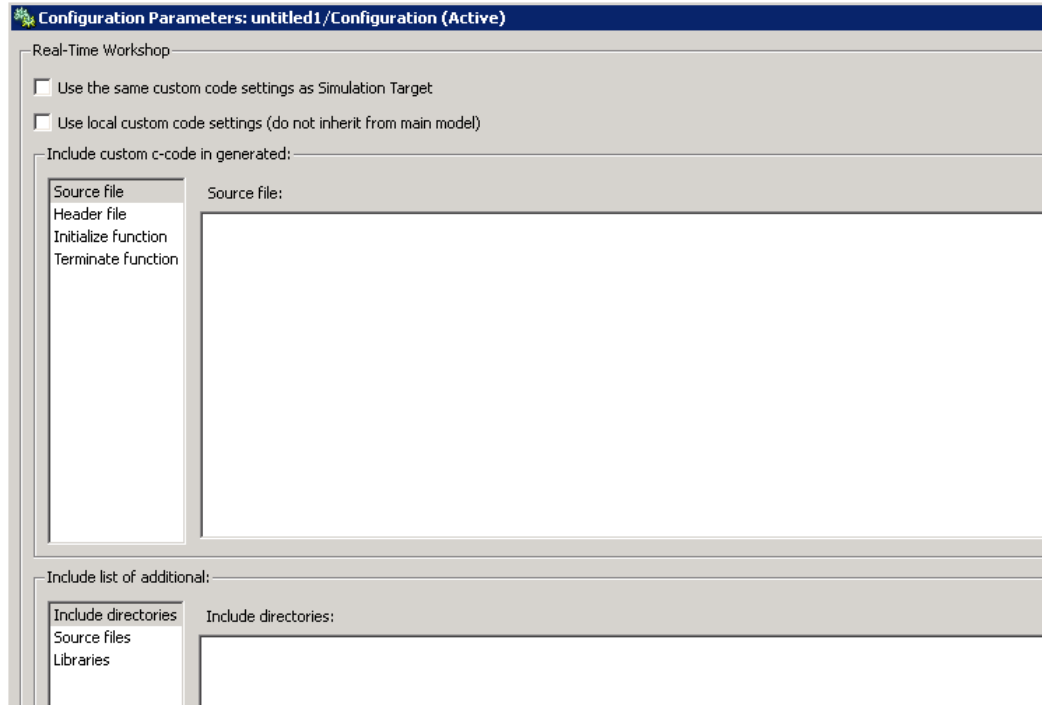
For more information, see "Real-Time Workshop Pane: Custom Code" in the Real-Time Workshop Reference documentation.

Mapping of Target Object Properties to Parameters in the Configuration Parameters Dialog Box

Previously, you could programmatically set options for simulation and embeddable code generation of models containing Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks by accessing the API properties of Target objects `sfun` and `rtw`, respectively. In R2008b, the API properties of Target objects `sfun` and `rtw` are replaced by parameters that you configure using the commands `get_param` and `set_param`.

For compatibility details, see "Compatibility Considerations" on page 116.

Mapping of Object Properties to Simulation Parameters for Nonlibrary Models. The following table maps API properties of the Target object sfun for nonlibrary models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CodeFlagsInfo ('debug')	**General > Enable debugging / animation**	SFSimEnableDebug string - off, **on**	**Simulation Target > Enable debugging / animation**
CodeFlagsInfo ('overflow')	**General > Enable overflow detection (with debugging)**	SFSimOverflowDetection string - off, **on**	**Simulation Target > Enable overflow detection (with debugging)**
CodeFlagsInfo ('echo')	**General > Echo expressions without semicolons**	SFSimEcho string - off, **on**	**Simulation Target > Echo expressions without semicolons**
CustomCode	**Custom Code > Include Code**	SimCustomHeaderCode *string* -	**Simulation Target > Custom Code > Header file**
CustomInitializer	**Custom Code > Initialization Code**	SimCustomInitializer *string* -	**Simulation Target > Custom Code > Initialize function**

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomTerminator	**Custom Code > Termination Code**	SimCustomTerminator *string* -	**Simulation Target > Custom Code > Terminate function**
ReservedNames	**Custom Code > Reserved Names**	SimReservedNameArray *string array* - **{}**	**Simulation Target > Symbols > Reserved names**
UserIncludeDirs	**Custom Code > Include Paths**	SimUserIncludeDirs *string* -	**Simulation Target > Custom Code > Include directories**
UserLibraries	**Custom Code > Libraries**	SimUserLibraries *string* -	**Simulation Target > Custom Code > Libraries**
UserSources	**Custom Code > Source Files**	SimUserSources *string* -	**Simulation Target > Custom Code > Source files**

Mapping of Object Properties to Simulation Parameters for Library Models. The following table maps API properties of the Target object sfun for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

113

Old `sfun` Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	**Custom Code > Include Code**	SimCustomHeaderCode *string* -	**Simulation Target > Header file**
CustomInitializer	**Custom Code > Initialization Code**	SimCustomInitializer *string* -	**Simulation Target > Initialize function**
CustomTerminator	**Custom Code > Termination Code**	SimCustomTerminator *string* -	**Simulation Target > Terminate function**
UseLocalCustomCodeSettings	**Custom Code > Use local custom code settings (do not inherit from main model)**	SimUseLocalCustomCode string - **off**, on	**Simulation Target > Use local custom code settings (do not inherit from main model)**
UserIncludeDirs	**Custom Code > Include Paths**	SimUserIncludeDirs *string* -	**Simulation Target > Include directories**
UserLibraries	**Custom Code > Libraries**	SimUserLibraries *string* -	**Simulation Target > Libraries**
UserSources	**Custom Code > Source Files**	SimUserSources *string* -	**Simulation Target > Source files**

Mapping of Object Properties to Code Generation Parameters for Library Models. The following table maps API properties of the Target object rtw for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old rtw Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	**Custom Code > Include Code**	CustomHeaderCode *string* -	**Real-Time Workshop > Header file**
CustomInitializer	**Custom Code > Initialization Code**	CustomInitializer *string* -	**Real-Time Workshop > Initialize function**
CustomTerminator	**Custom Code > Termination Code**	CustomTerminator *string* -	**Real-Time Workshop > Terminate function**
UseLocalCustomCodeSettings	**Custom Code > Use local custom code settings (do not inherit from main model)**	RTWUseLocalCustomCode string - **off**, on	**Real-Time Workshop > Use local custom code settings (do not inherit from main model)**
UserIncludeDirs	**Custom Code > Include Paths**	CustomInclude *string* -	**Real-Time Workshop > Include directories**

Old rtw Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
UserLibraries	**Custom Code > Libraries**	CustomLibrary *string* -	**Real-Time Workshop > Libraries**
UserSources	**Custom Code > Source Files**	CustomSource *string* -	**Real-Time Workshop > Source files**

Compatibility Considerations. When you load and save older models in R2008b, not all target property settings are preserved.

What Happens When You Load an Older Model in R2008b

When you use R2008b to load a model created in an earlier version, dialog box options and the equivalent object properties for simulation and embeddable code generation targets migrate automatically to the Configuration Parameters dialog box, except in the cases that follow.

For the simulation target (sfun) of a nonlibrary model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the Simulation Target Dialog Box of a Nonlibrary Model	Equivalent Object Property
Custom Code > Use these custom code settings for all libraries	ApplyToAllLibs

Option in the Simulation Target Dialog Box of a Nonlibrary Model	Equivalent Object Property
Description > Description	`Description`<hr>**Note** If you load an older model that contained user-specified text in the **Description** field, that text now appears in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.
Description > Document Link	`Document`

For the simulation target (`sfun`) of a library model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the Simulation Target Dialog Box of a Library Model	Equivalent Object Property
General > Enable debugging / animation	`CodeFlagsInfo('debug')`
General > Enable overflow detection (with debugging)	`CodeFlagsInfo('overflow')`
General > Echo expressions without semicolons	`CodeFlagsInfo('echo')`
General > Build Actions	None
Custom Code > Reserved Names	`ReservedNames`
Description > Description	`Description`
Description > Document Link	`Document`

For the embeddable code generation target (rtw) of a library model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the RTW Target Dialog Box of a Library Model	Equivalent Object Property
General > Comments in generated code	`CodeFlagsInfo('comments')`
General > Use bitsets for storing state configuration	`CodeFlagsInfo('statebitsets')`
General > Use bitsets for storing boolean data	`CodeFlagsInfo('databitsets')`
General > Compact nested if-else using logical AND/OR operators	`CodeFlagsInfo('emitlogicalops')`
General > Recognize if-elseif-else in nested if-else statements	`CodeFlagsInfo('elseifdetection')`
General > Replace constant expressions by a single constant	`CodeFlagsInfo('constantfolding')`
General > Minimize array reads using temporary variables	`CodeFlagsInfo('redundantloadelimination')`
General > Preserve symbol names	`CodeFlagsInfo('preservenames')`
General > Append symbol names with parent names	`CodeFlagsInfo('preservenameswithparent')`
General > Use chart names with no mangling	`CodeFlagsInfo('exportcharts')`
General > Build Actions	None
Custom Code > Reserved Names	`ReservedNames`
Description > Description	`Description`
Description > Document Link	`Document`

What Happens When You Save an Older Model in R2008b

When you use R2008b to save a model created in an earlier version, parameters for simulation and embeddable code generation from the Configuration Parameters dialog box are saved. However, properties of API Target objects sfun and rtw are not saved if those properties do not have an equivalent parameter in the Configuration Parameters dialog box. In R2008b, this behavior applies even if you choose to save the model as an older version (for example, R2007a).

New Parameters in the Configuration Parameters Dialog Box for Simulation and Embeddable Code Generation

In R2008b, new parameters are added to the Configuration Parameters dialog box for simulation and embeddable code generation of models that contain Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

New Simulation Parameters for Nonlibrary Models. The following table lists the new simulation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimBuildMode string — **sf_incremental_build**, sf_nonincremental_build, sf_make, sf_make_clean, sf_make_clean_objects	**Simulation Target > Simulation target build mode**	Specifies how you build the simulation target for a model.
SimCustomSourceCode *string* -	**Simulation Target > Custom Code > Source file**	Enter code lines to appear near the top of a generated source code file.

New Simulation Parameter for Library Models. The following table lists the new simulation parameter that applies to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimCustomSourceCode *string* -	**Simulation Target > Source file**	Enter code lines to appear near the top of a generated source code file.

New Code Generation Parameters for Nonlibrary Models. The following table lists the new code generation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
ReservedNameArray *string array* - **{}**	**Real-Time Workshop > Symbols > Reserved names**	Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code.
RTWUseSimCustomCode string – **off**, on	**Real-Time Workshop > Custom Code > Use the same custom code settings as Simulation Target**	Specify whether to use the same custom code settings as those specified for simulation.
UseSimReservedNames string – **off**, on	**Real-Time Workshop > Symbols > Use the same reserved names as Simulation Target**	Specify whether to use the same reserved names as those specified for simulation.

New Code Generation Parameters for Library Models. The following table lists the new code generation parameters that apply to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
CustomSourceCode *string* –	**Real-Time Workshop > Source file**	Enter code lines to appear near the top of a generated source code file.
RTWUseSimCustomCode string – **off**, on	**Real-Time Workshop > Use the same custom code settings as Simulation Target**	Specify whether to use the same custom code settings as those specified for simulation.

S-Functions

Ada S-Functions

In future releases, Simulink will not have a built-in Ada S-function capability. As a mitigation strategy, call Ada code from Simulink using standard Ada 95 language features and the Simulink C-MEX S-function API. For details of this process, please contact Technical Support at The MathWorks.

Legacy Code Tool Enhancement

The Legacy Code Tool data structure has been enhanced with a new S-function options field, singleCPPMexFile, which when set to true (1).

- Requires you to generate and manage an inlined S-function as only one file (.cpp) instead of two (.c and .tlc)

- Maintains model code style—level of parentheses usage and preservation of operand order in expressions and condition expressions in if statements—as specified by model configuration parameters.

When you choose not to use this option, code generated by the Legacy Code Tool does not reflect code style configuration settings and requires you to manage C-MEX and TLC files.

For more information, see:

- "Integrating Existing C Functions into Simulink Models with the Legacy Code Tool" in the Writing S-Functions documentation

- "Automating Generation of Files for Fully Inlined S-Functions Using Legacy Code Tool" in the Real-Time Workshop documentation

- `legacy_code` function reference page

Compatibility Considerations.

- If you upgrade from an earlier release, you can continue to use S-functions generated from the Legacy Code Tool available in earlier releases. You can continue to compile the S-function source code and you can continue to use the compiled output from an earlier release without recompiling the code.

- If you set the new `singleCPPMexFile` options field to true (1) when creating an S-function, you cannot use that S-function, in source or compiled form, with versions of Simulink earlier than Version 7.2 (R2008b).

MATLAB Changes Affecting Simulink

Changes to MATLAB Startup Options

The `matlab` command line arguments `-memmgr` and `-check_malloc` are deprecated and will be removed in a future release.

For more information, see "Changes to matlab Memory Manager Startup Options" in the *MATLAB Release Notes*.

Handle Graphics Not Supported Under `-nojvm` Startup Option

If you start MATLAB using the command `matlab -nojvm` (which disables Java), you will receive warnings when using many graphical tools, for example, when you create figures, print Simulink models, or view Simulink scopes.

For more information, see Changes to -nojvm Startup Option in the Desktop Tools and Development Environment release notes.

Version 7.1 (R2008a) Simulink Software

This table summarizes what's new in V7.1 (R2008a):

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

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

- "Simulation Performance" on page 123
- "Component-Based Modeling" on page 124
- "Embedded MATLAB Function Blocks" on page 125
- "Data Management" on page 126
- "Simulink File Management" on page 132
- "Block Enhancements" on page 133
- "User Interface Enhancements" on page 135
- ""What's This?" Context-Sensitive Help Available for Simulink Configuration Parameters Dialog" on page 137
- "S-Functions" on page 138

Simulation Performance

Rapid Accelerator
Improved Rapid Accelerator sim-command performance when running long simulations of small models on Microsoft Windows® platforms.

Long Rapid Accelerator mode simulations of small models invoked by the sim command under the Microsoft Windows operating system now run faster.

Additional Zero Crossing Algorithm

A second zero crossing algorithm that is especially useful in systems exhibiting strong chattering behavior has been added for use with variable step solvers.

The new algorithm is selected by choosing Adaptive from the **Zero crossing location algorithm** option in the Solver pane of the Configuration Parameter dialog. The default algorithm is Non-Adaptive, which is the algorithm used prior to this release.

For more information, see "Zero-Crossing Algorithms".

Component-Based Modeling

Efficient Parent Model Rebuilds

In previous releases, changing a referenced model that executed in Accelerator mode or was used for code generation triggered rebuilding every model that directly or indirectly referenced the changed model. The rebuilding occurred even if the change to the referenced model had no effect on its interface to its parent(s).

In R2008a, changing a referenced model that executes in Accelerator mode or is used for code generation triggers rebuilding a parent model only when the change directly affects the referenced model's interface to the parent model. This behavior eliminates unnecessary code regeneration, which can significantly reduce the time needed to update a diagram.

The faster diagram update has no effect on simulation behavior or performance, but may change the messages that appear in the MATLAB Command Window. See "Referencing a Model" for information about model referencing.

Scalar Root Inputs Passed Only by Reference

The **Configuration Parameters > Model Referencing > Pass scalar root inputs by value** option is Off by default, indicating that scalar root inputs

are passed by reference. In previous releases, setting the option to On affected both simulation and generated code, and caused scalar root inputs to be passed by value. In R2008a, the option has no effect on simulation: scalar root inputs are now always passed by reference, regardless of the setting of **Pass scalar root inputs by value**. The effect of the option on code generation is the same as in previous releases. See "Pass fixed-size scalar root inputs by value for Real-Time Workshop" for more information.

Unlimited Referenced Models

In previous releases, Microsoft Windows imposed a limit on the number of models that could be referenced in Accelerator mode in a model hierarchy. This limitation is removed in R2008a. Under Microsoft Windows, as on all other platforms, the number of referenced models that can appear in a model hierarchy is effectively unlimited. See "Referencing a Model" for information about model referencing.

Embedded MATLAB Function Blocks

Nontunable Structure Parameters

Embedded MATLAB Function blocks now support nontunable MATLAB structure parameters. For more information, see "Working with Structure Parameters in Embedded MATLAB Function Blocks".

Bidirectional Traceability

You can navigate between a line of generated code and its corresponding line of source code in Embedded MATLAB Function blocks. For more information, see "Using Traceability in Embedded MATLAB Function Blocks".

Specify Scaling Explicitly for Fixed-Point Data

When you define data of fixed-point type in Embedded MATLAB Function blocks, you must specify the scaling explicitly in the General pane of the Data properties dialog box. For example, you cannot enter an incomplete specification such as fixdt(1,16) in the Type field. If you do not specify scaling explicitly, you will see an error message when you try to simulate your model.

To ensure that the data type definition is valid for fixed-point data, perform one of these steps in the General pane of the Data properties dialog box:

- Use a predefined option in the Type drop-down menu.

- Use the Data Type Assistant to specify the Mode as fixed-point.

Compatibility Considerations. Previously, you could omit scaling in data type definitions for fixed-point data. Such data types were treated as integers with the specified sign and word length. This behavior has changed. Embedded MATLAB Function blocks created in earlier versions may now generate errors if they contain fixed-point data with no scaling specified.

Data Management

Array Format Cannot Be Used to Export Multiple Matrix Signals

When you export signals to a workspace in Array format from more than one outport, none of the signals can be a matrix signal. In previous releases, violating this rule did not always cause an error, but the matrix data was not exported correctly. In R2008a, violating the rule always causes an error, and no data export occurs. See "Array" for complete information about Array format requirements.

When exporting data to a workspace in Array format from multiple outports, use a Reshape block to convert any matrix signal to a one-dimensional (1-D) array. This restriction applies only to Array format. If you specify either Structure or Structure with time format, you can export matrix signals to a workspace from multiple outports without first converting the signals to vectors.

Compatibility Considerations. The more stringent error checking in R2008a can cause models that export data in Array format from multiple outports to generate errors rather than silently exporting matrix data incorrectly. To eliminate such errors, use a Reshape block to convert any matrix signal to a vector, or switch to Structure or Structure with time format. See "Exporting Data to the MATLAB Base Workspace" for information about data export.

Bus Editor Upgraded

The Simulink Bus Editor has been reimplemented to provide a GUI interface similar to that of the Model Explorer, and to provide several new capabilities, including importing/exporting data from MAT-files and M-files, defining bus objects and elements with the Data Type Assistant, and creating and viewing bus hierarchies (nested bus objects). See "Using the Bus Editor" for details.

Changing Nontunable Values Does Not Affect the Current Simulation

In previous releases, changing the value of any variable or parameter during simulation took effect immediately. In R2008a, only changes to tunable variables and parameters take effect immediately. Other changes have no effect until the next simulation begins. This modification causes simulation behavior to match generated code behavior when the values of nontunable variables and parameters change, and it improves efficiency by eliminating unnecessary re-evaluation. For information about parameter tuning, see "Tunable Parameters" and "Using Tunable Parameters".

Compatibility Considerations. In R2008a, simulation behavior will differ if the behavior in a previous release depended on changing any nontunable variable or parameter during simulation. To regain the previous behavior, define as tunable any nontunable variable or parameter that you want to change during simulation for the purpose of affecting simulation immediately.

Detection of Illegal Rate Transitions

Illegal rate transitions between a block and a triggered subsystems or function call subsystems are now detected when the block is connected to a Unit Delay or Zero Hold block inside a triggered subsystem.

Compatibility Considerations. In this release, Simulink detects illegal rate transition errors when the block sample time is different from the triggered subsystem sample time in those models where the block is connected to a Unit Delay or Zero Hold block inside the triggered subsystem.

Explicit Scaling Required for Fixed-Point Data

In R2008a, when you define a fixed-point data type in a Simulink model, you must explicitly specify the scaling unless the block supports either integer

127

scaling mode or best-precision scaling mode. If a block supports neither of these modes, you cannot define an incomplete fixed-point data type like `fixdt(1,16)`, which specifies no scaling. See "Specifying a Fixed-Point Data Type" and "Showing Fixed-Point Details" for information about defining fixed-point data types.

Compatibility Considerations. In previous releases, you could define a fixed-point data type that specified no scaling in a block that supported neither integer scaling mode nor best-precision scaling mode. The Simulink software posted no warning, and treated fixed-point data type as an integer data type with the specified word length. For example, `fixdt(1,16)` was treated as `fixdt(1,16,0)`.

In R2008a, a fixed-point data type definition that specifies no scaling generates an error unless the block supports either integer scaling mode or best-precision scaling mode. If such an error occurs when you compile a model from an earlier Simulink version, redefine the incomplete fixed-point data type to be an integer type if nothing more is needed, or to be scaled appropriately for its value range.

Fixed-Point Details Display Available

The Data Type Assistant can now display the status and details of fixed-point data types. See "Specifying a Fixed-Point Data Type" and "Showing Fixed-Point Details" for more information.

More than 2GB of Simulation Data Can be Logged on 64-Bit Platforms

When you log time, states, final states, and signals on a 64-bit platform, you can now save more simulation data in the MATLAB base workspace than was previously possible.

- When you log data using the `Structure`, `Structure with time`, or `Timeseries` format, you can now save up to $2^{48}-1$ bytes in each field that contains logged data.

- When you log data using `Array` format, you can now save up to $2^{48}-1$ bytes in each array that contains logged data.

Previously the limit was 2^31-1 bytes in each field or array containing logged data. See "Logging Signals" and "Data Import/Export Pane" for information about logging data.

Order of Simulink and MPT Parameter and Signal Fields Changed

The order of the fields in the `Simulink.Parameter` and `Simulink.Signal` classes, and in their respective subclasses `mpt.Parameter` and `mpt.Signal`, has changed in R2008a.

The order for `Simulink.Parameter` (and `mpt.Parameter`) is now:

```
Simulink.Parameter (handle)
         Value: []
        RTWInfo: [1x1 Simulink.ParamRTWInfo]
    Description: ''
       DataType: 'auto'
            Min: -Inf
            Max: Inf
       DocUnits: ''
     Complexity: 'real'
     Dimensions: [O O]
```

The order for `Simulink.Signal` (and `mpt.Signal`) is now:

```
Simulink.Signal (handle)
         RTWInfo: [1x1 Simulink.SignalRTWInfo]
     Description: ''
        DataType: 'auto'
             Min: -Inf
             Max: Inf
        DocUnits: ''
      Dimensions: -1
      Complexity: 'auto'
      SampleTime: -1
    SamplingMode: 'auto'
    InitialValue: ''
```

Loading a model that uses any `Simulink.Parameter` or `mpt.Parameter` objects, and was saved in a release prior to R2008a, may post an Inconsistent Data warning in the MATLAB Command Window. This message does not indicate a problem with the model, which need not be changed. Resave the model in R2008a to update it to use the new parameter class definitions. The warning will not appear when you reopen the model.

Range Checking for Complex Numbers

Previous releases did not provide range checking for complex numbers, and attempting it generated an error. In R2008a, you can specify a minimum and/or maximum value for a complex number wherever range checking is available and a complex number is a legal value.

The specified minimum and maximum values apply separately to the real part and to the imaginary part of the complex number. If the value of either part of the number is less than the minimum, or greater than the maximum, the complex number is outside the specified range.

No range checking occurs against any combination of the real and imaginary parts, such as (`sqrt(a^2+b^2)`). See "Checking Parameter Values" and "Checking Signal Ranges" for information about range checking.

Rate Transition Blocks Needed on Virtual Buses

In this release, Simulink never automatically inserts a Rate Transition block into a virtual bus, even if `Automatically handle rate transfer` is selected. Instead, an error is displayed indicating that a Rate Transition block must be manually inserted.

Compatibility Considerations. Some models that worked in previous releases, but were dependent on automatic Rate Transition block insertion, will now report an error and will no longer run. An error will be reported if all of these apply:

- The `Automatically handle rate transfer` option is enabled

- The model is multirate

- The model has a virtual bus, all of the elements on the bus have the same data type, and the sample time changes

- A bus selector block is not present on the virtual bus at a point after the sample time changes

- The only way to address the rate transition problem is to insert a rate transition block

Sample Times for Virtual Blocks

In models with asynchronous function calls, some virtual blocks now correctly assign generic sample times instead of triggered sample times.

Compatibility Considerations. The CompiledSampleTime parameter now reports the compiled sample time as generic sample time (that is, [-1, -inf]) rather than triggered sample time ([-1,-1]) for virtual blocks for which all of the flowing is true:

- The virtual block is downstream from an asynchronous source

- The virtual block is not inside a triggered subsystem

- The virtual block had a triggered ([-1,-1]) sample time in previous releases

The simulation results, code generation, and sample time colors are not affected by this change.

Signals Needing Resolution Are Graphically Indicated

In R2008a, the Simulink Editor by default graphically indicates signals that must resolve to signal objects. For any labeled signal whose **Signal name must resolve to signal object** property is enabled, a signal resolution icon appears to the left of the signal name. The icon looks like this:

A signal resolution icon indicates only that a signal's **Signal name must resolve to signal object** property is enabled. The icon does not indicate whether the signal is actually resolved, and does not appear on a signal that is implicitly resolved without its **Signal name must resolve to signal object** property being enabled. See "Signal Resolution Indicators" for more information.

Simulink File Management

Autosave

New Autosave option automatically creates a backup copy of models before updating or simulating. If you open or load a model which has a more recent

autosave copy available, a dialog appears where you can choose to overwrite the original model file with the autosave copy.

You can set the Autosave option in the Simulink Preferences Window. See Autosave in the Simulink Graphical User Interface documentation.

Old Version Notification

New option to notify when loading a model saved in a previous version of Simulink software.

You can set this option in the Simulink Preferences Window. See "Simulink Preferences Window: Main Pane" in the Simulink Graphical User Interface documentation.

Model Dependencies Tools

Enhanced file dependency analysis now also detects:

- TLC files required by S-functions.
- `.fig` files created by GUIDE.
- Files referenced by common data loading functions. File names passed to `xlsread`, `importdata`, `dlmread`, `csvread`, `wk1read`, and `textread` are now identified, in addition to the existing capability for `load`, `fopen` and `imread`.

See "Scope of Dependency Analysis" in the Using Simulink documentation.

Block Enhancements

New Discrete FIR Filter Block Replaces Weighted Moving Average Block

The Discrete FIR Filter block in the Discrete library is new for this release. This block independently filters each channel of the input signal with the specified digital FIR filter. The Discrete FIR Filter block replaces the Weighted Moving Average block.

Compatibility Considerations. You should replace Weighted Moving Average blocks in your existing models with the Discrete FIR Filter block. To do this, run the `slupdate` command on your models.

Rate Transition Block Enhancements

Support for Rate Transition blocks has been enhanced in the following ways:

- Rate Transition block output port sample time now can be specified as a multiple of input port sample time, using the new Rate Transition block parameters **Output port sample time options** and **Sample time multiple (>0)**. See the Rate Transition block documentation in the *Simulink Reference* for more information.

- In previous releases, auto-insertion of Rate Transition blocks was selected for a model using the option **Automatically handle data transfers between tasks** on the **Solver** pane of the Configuration Parameters dialog box. When selected, auto-insertion always ensured data transfer determinism for periodic tasks.

  This release allows you to control the level of data transfer determinism when auto-insertion of Rate Transition blocks is selected for your model. The **Solver** pane option for selecting auto-insertion has been renamed to **Automatically handle rate transition for data transfer**. Selecting auto-insertion now enables a new option, **Deterministic data transfer**. Selecting `Never (minimum delay)` or `Whenever possible` for this option can provide reduced latency for models that do not require determinism. See the "Solver Pane" section in the *Simulink Graphical User Interface* documentation for more information.

- Auto-insertion of Rate Transition blocks is now supported for additional rate transitions, such as sample times with nonzero offset, and between non-integer-multiple sample times.

Enhanced Lookup Table (n-D) Block

The Lookup Table (n-D) block now supports all data types, complex table data, and nonscalar inputs. See the Lookup Table (n-D) block documentation in the *Simulink Reference* for more information.

New Accumulator Parameter on Sum Block

The Sum block dialog box displays a new parameter for specifying the data type of its accumulator. See the Sum block documentation in the *Simulink Reference* for more information.

User Interface Enhancements

Simulink Library Browser

A new version of the Simulink Library browser has the following enhancements:

- Now available on all platforms supported by Simulink software.

- Improved performance for browsing and searching of libraries, by allowing these operations to proceed without actually loading the libraries.

- Enhanced search finds all blocks and displays search results in a separate tab.

- New option to display library blocks in a compact grid layout that conserves screen space.

Simulink Preferences Window

New unified Simulink Preferences window for configuring default settings. The new Preferences window allows you to configure file change notifications, autosave options, fonts, display options, and model configuration defaults.

See "Simulink Preferences Window: Main Pane" in the Simulink Graphical User Interface documentation.

Model Advisor

In R2008a, the Model Advisor tool is enhanced with improved GUI navigation, check analysis, and reports including:

- Reset option that reverts the status of all checks to **Not Run** while keeping the current check selection.

- Model Advisor Result Explorer to make changes to your model.

- **Input Parameters** to provide inputs to checks.

- Check results reported in the same order as the Model Advisor tree.

- The ability to generate reports for any folder.

- Timestamps in reports indicating when checks run at different times.

See "Consulting the Model Advisor" in the Simulink User's Guide.

Solver Controls

Enhanced controls in the Solver pane of the Configuration Parameters dialog. The Solver pane of the Configuration Parameters dialog has been changed as follows:

- The **Solver diagnostic controls** pane has been removed and two new panes have been added (**Tasking and sample time options**, and **Zero crossing options**)

- The `Automatically handle data transfers between tasks` control has been moved to the **Tasking and sample time options** pane, and has been renamed `Automatically handle rate transition for data transfer`

- The `Higher priority value indicates higher task priority` control has been moved to the **Tasking and sample time options** pane

- The `Number of consecutive min step size violations allowed` control has been moved to the **Solver options** pane, and has been renamed `Consecutive min step size violations allowed`

- The `States shape preservation` control has been added to the **Solver options** pane

- The `Consecutive zero crossings relative tolerance` control has been moved to the **Zero crossing options** pane

- The `Number of consecutive zero crossings allowed` control has been moved to the **Zero crossing options** pane

- The `Zero crossing control` control has been moved to the **Zero crossing options** pane

- The `Zero crossing location algorithm` control has been added to the **Zero crossing options** pane

- The `Zero crossing location threshold` control has been added to the **Zero crossing options** pane

- Options that in previous releases were only visible when enabled are now always visible. They are grayed when not enabled.

For more information on the Configuration parameters solver pane, see "Solver Pane".

Compatibility Considerations. The Solver pane of the Configuration Parameter dialog has been restructured, and many parameters have moved or been renamed. Please refer to the list of changes above for information on specific parameters.

"What's This?" Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

R2008a introduces "What's This?" context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

1 Place your cursor over the label of a parameter.

2 Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.
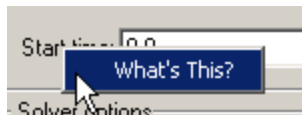
3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

S-Functions

Simplified Level-2 M-File S-Function Template

New basic version of the Level-2 M-file S-function template
`msfuntmpl_basic.m` simplifies creating Level-2 M-file S-functions. See
"Writing Level-2 M-File S-Functions" in *Writing S-Functions* for more
information.

Compatibility Considerations

MATLAB V7.6 (R2008a) on Linus Torvalds' Linux® platforms is now built
with a compiler that utilizes glibc version 2.3.6. To work with MATLAB
V7.6 (R2008a), MEX-file S-functions compiled on a Linux® platform must be
rebuilt.

Version 7.0 (R2007b) Simulink Software

This table summarizes what's new in V7.0 (R2007b):

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

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

- "Simulation Performance" on page 139
- "Component-Based Modeling" on page 141
- "Embedded MATLAB Function Blocks" on page 142
- "Data Management" on page 143
- "Configuration Management" on page 146
- "Embedded Software Design" on page 147
- "Block Enhancements" on page 148
- "Usability Enhancements" on page 150
- "S-Functions" on page 150

Simulation Performance

Simulink Accelerator

Simulink® Accelerator™ has been incorporated into Simulink software, and a new Rapid Accelerator mode has been added for faster simulation through code generation technology. See "Accelerating Models" in *Simulink User's Guide*.

> **Note** When using From File blocks in Rapid Accelerator mode, the
> corresponding MAT file must be in the current directory.

Compatibility Considerations. A license is no longer required to use the
Accelerator or Rapid Accelerator modes.

Simulink Profiler

Simulink Profiler has been incorporated into Simulink software for the
identification of simulation performance bottlenecks. See "Capturing
Performance Data" in *Simulink User's Guide*.

Compiler Optimization Level

Simulink Accelerator mode, Rapid Accelerator mode, and model reference
simulation targets can now specify the compiler optimization level used
(choose between minimizing compilation time or simulation time). See
"Customizing the Build Process" in *Simulink User's Guide*.

Compatibility Considerations. The new model configuration parameter
Compiler optimization level defaults to `Optimizations off (faster
builds)`. As a result, you might notice shorter build times, but longer
execution times, compared to previous releases. However, any previously
defined custom compiler optimization options using `OPT_OPTS` will be honored,
and model behavior should be unchanged.

Variable-Step Discrete Solver

Simulink software has been enhanced to no longer take unnecessary time
steps at multiples of the maximum step size when using a variable-step
discrete solver.

Referenced Models Can Execute in Normal or Accelerator Mode

In previous releases, Simulink software executed all referenced models by
generating code for them and executing the generated code. In this release,
Simulink software can execute appropriately configured referenced models
interpretively. Such execution is called Normal mode execution, and execution
via generated code is now called Accelerator mode execution. The technique of

executing a referenced model via generated code has not changed, but it did not previously need a separate name because it was the only alternative.

Many restrictions that previously applied to all referenced model execution now apply only to Accelerator mode execution, and are relaxed in Normal mode. For example, some Simulink tools that did not work with referenced models because they are incompatible with generated code can now be used by executing the referenced model in Normal mode.

Normal mode also has some restrictions that do not apply to Accelerator mode. For example, at most, one instance of a given model in a referenced model hierarchy can execute in Normal mode. See "Referencing a Model" in *Simulink User's Guide* for information about using referenced models in Normal and Accelerator mode.

Accelerator and Model Reference Targets Now Use Standard Internal Functions

For more consistent simulation results, Simulink Accelerator mode, Rapid Accelerator mode, and the model reference simulation target now perform mathematical operations with the same internal functions that MATLAB and Simulink products use.

Component-Based Modeling

New Instance View Option for the Model Dependency Viewer

The Model Dependency viewer has a new option to display each reference to a model and indicate whether the reference is simulated in Accelerator or Normal mode. See "Referencing a Model" and "Using the Model Dependency Viewer" in *Simulink User's Guide*.

Mask Editor Now Requires Java

The Mask Editor now requires that the MATLAB product start with Java enabled. See "Simulink Mask Editor" in *Simulink User's Guide*.

Compatibility Considerations. You can no longer use the Mask Editor if you start MATLAB with the `-nojvm` option.

Embedded MATLAB Function Blocks

Complex and Fixed-Point Parameters

Embedded MATLAB Function blocks now support complex and fixed-point parameters.

Support for Algorithms That Span Multiple M-Files

You can now generate embeddable code for external M-functions from Embedded MATLAB function blocks. This feature allows you to call external functions from multiple locations in an M-file or model and include these functions in the generated code.

Compatibility Considerations. In previous releases, Embedded MATLAB function blocks did not compile external M-functions, but instead dispatched them to the MATLAB product for execution (after warning). Now, the default behavior is to compile and generate code for external M-functions called from Embedded MATLAB function blocks. If you do not want Embedded MATLAB function blocks to compile external M-functions, you must explicitly declare them to be extrinsic, as described in "Calling MATLAB Functions" in the Embedded MATLAB documentation.

Loading R2007b Embedded MATLAB Function Blocks in Earlier Versions of Simulink Software

If you save Embedded MATLAB Function blocks in R2007b, you will not be able to load the corresponding model in earlier versions of Simulink software. To work around this issue, save your model in the earlier version before loading it, as follows:

1 In the Simulink Editor, select **File > Save As**.

2 In the **Save as type** field, select the version in which you want to load the model.

   For example, if you want to load the model in Simulink R2007a, select `Simulink 6.6/R2007a Models (*.mdl)`.

Data Management

New Diagnostic for Continuous Sample Time on Non-Floating-Point Signals

A new diagnostic detects continuous sample time on non-floating-point signals.

New Standardized User Interface for Specifying Data Types

This release introduces a new standardized user interface, the **Data Type Assistant**, for specifying data types associated with Simulink blocks and data objects, as well as Stateflow data. See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

The **Data Type Assistant** appears on the dialogs of the following Simulink blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference
- Discrete Derivative
- Discrete-Time Integrator
- Dot Product
- Embedded MATLAB Function (Ports and Data Manager dialog)
- Gain
- Inport
- Interpolation Using Prelookup
- Logical Operator
- Lookup Table
- Lookup Table (2-D)

- Lookup Table Dynamic

- Math Function

- MinMax

- Multiport Switch

- Outport

- Prelookup

- Product, Divide, Product of Elements

- Relational Operator

- Relay

- Repeating Sequence Interpolated

- Repeating Sequence Stair

- Saturation

- Saturation Dynamic

- Signal Specification

- Sum, Add, Subtract, Sum of Elements

- Switch

- Weighted Moving Average (obsolete — replaced by the Discrete FIR Filter block)

The **Data Type Assistant** appears on the dialogs of the following Simulink data objects:

- Simulink.BusElement

- Simulink.Parameter

- Simulink.Signal

- Simulink.StructElement

New Block Parameters for Specifying Minimum and Maximum Values

The following new block parameters are available for specifying the minimum and maximum values of signals and other block parameters.

- **Output minimum**, **Minimum**

- **Output maximum**, **Maximum**

- **Parameter minimum**

- **Parameter maximum**

These new parameters selectively appear on the dialogs of the following Simulink blocks:

- Abs

- Constant

- Data Store Memory

- Data Type Conversion

- Difference

- Discrete Derivative

- Discrete-Time Integrator

- Gain

- Inport

- Interpolation Using Prelookup

- Lookup Table

- Lookup Table (2-D)

- Math Function

- MinMax

- Multiport Switch

- Outport

- Product, Divide, Product of Elements

- Relay

- Repeating Sequence Interpolated

- Repeating Sequence Stair

- Saturation

- Saturation Dynamic

- Signal Specification

- Sum, Add, Subtract, Sum of Elements

- Switch

New Range Checking of Block Parameters

In this release, Simulink software performs range checking of parameters associated with blocks that specify minimum and maximum values (see "New Block Parameters for Specifying Minimum and Maximum Values" on page 145). Simulink software alerts you when values of block parameters lie outside a range that corresponds to its specified minimum and maximum values and data type. See "Checking Parameter Values" in *Simulink User's Guide* for more information.

New Diagnostic for Checking Signal Ranges During Simulation

In the Configuration Parameters dialog, the **Diagnostics > Data Validity** pane contains a new diagnostic, **Simulation range checking**, which alerts you during simulation when blocks output signals that exceed specified minimum or maximum values (see "New Block Parameters for Specifying Minimum and Maximum Values" on page 145). For more information about using this diagnostic, see "Checking Signal Ranges" in *Simulink User's Guide*.

Configuration Management

Disabled Library Link Management

The following new features help manage disabled library links and protect against accidental loss of work:

- "Disabled Link" appears in the title bar of a Model Editor window that displays a subsystem connected to a library by a disabled link.

- ToolTips for library-linked blocks include the link status as well as the destination block for the link.

- New diagnostics warn when saving a model that contains disabled or parameterized library links.

- New Model Advisor checks let you search for disabled or parameterized library links in a model.

See "Disabling Links to Library Blocks" in *Simulink User's Guide* for more information.

Model Dependencies Tools

The model dependencies manifest tools have these new capabilities:

- Enhanced analysis to detect file dependencies from Stateflow transitions, Embedded MATLAB functions, and requirements documents. See "Scope of Dependency Analysis" in *Simulink User's Guide*.

- Model dependencies tools now save user manifest edits for reuse the next time a manifest is generated. See "Editing Manifests" in *Simulink User's Guide*.

Embedded Software Design

Legacy Code Tool Enhancement

The Legacy Code Tool has been enhanced to allow the use of `void*` and `void**` to declare variables that represent memory allocated for specific instances of items such as file descriptors, device drivers, and memory managed externally.

For more information, see:

- "Integrating Existing C Functions into Simulink Models with the Legacy Code Tool" in the *Developing S-Functions*

- `legacy_code` function documentation in the *Simulink Reference*

Block Enhancements

Product Block Reorders Inputs Internally

In previous releases, a Product block whose

- **Number of inputs** parameter begins with a divide character (/)
- **Multiplication** parameter specifies `Element-wise(.*)`

computes the reciprocal of its first input before multiplying or dividing by subsequent inputs. For example, if a Product block specifies division for its first input, *u1*, and multiplication for its second input, *u2*, previous versions of Simulink software compute

```
(1 / u1) * u2
```

In this release, the Product block internally reorders its first two inputs if particular conditions apply, such that Simulink software now computes

```
u2 / u1
```

See the Product block documentation in the *Simulink Reference* for more information.

Block Data Tips Now Work on All Platforms

In previous releases, block data tips worked only on Microsoft Windows platforms. In this release, the data tips work on all platforms. Also, the data tip for a library link, even if disabled, now includes the name of the library block it references.

Enhanced Data Type Support for Blocks

The following blocks now allow you to specify the data type of their outputs:

- Abs
- Multiport Switch
- Saturation
- Saturation Dynamic

• Switch

The following blocks now support single-precision floating-point inputs, outputs, and parameter values:

• Discrete Filter

• Discrete State-Space

• Discrete Transfer Fcn

New Simulink Data Class Block Object Properties

The following properties have been added to the `Simulink.BlockData` class:

• **AliasedThroughDataType**

• **AliasedThroughDataTypeID**

New Break Link Options for save_system Command

The `save_system` command's `BreakLink` option has been replaced by two options: `BreakAllLinks` and `BreakUserLinks`. The first option duplicates the behavior of the obsolete `BreakLink` option, i.e., it replaces all library links, including links to Simulink block libraries with copies of the referenced library blocks. The `BreakUserLinks` option replaces only links to user-defined libraries.

Compatibility Considerations. The `save_system` command continues to honor the `BreakLink` option but displays a warning message at the MATLAB command line that the option is deprecated.

Simulink Software Checks Data Type of the Initial Condition Signal of the Integrator Block

When the output port of the Constant or IC block is connected to the Initial Condition port of the Integrator block, Simulink software now compares the data type of the Initial Condition input signal of the Integrator block with the **Constant value** parameter or **Initial value** parameter of the Constant block or IC block, respectively.

Compatibility Considerations. If the data type for the output port of the Constant or IC blocks does not match the data type of the Initial Condition input signal for the Integrator block, Simulink software returns an error at compile time.

Usability Enhancements

Model Advisor

Model Advisor has been enhanced to navigate checks, display status, and report results. Also, this release contains a new "Model Advisor Checks" reference.

Alignment Commands

This release contains new block alignment, distribution, and resize commands to align groups of blocks along their edges, equalize interblock spacing, and resize blocks to be all the same size. See "Aligning, Distributing, and Resizing Groups of Blocks Automatically" in *Simulink User's Guide* for more information.

S-Functions

New S-Function APIs to Support Singleton Dimension Handling

The following functions have been added:

- ssPruneNDMatrixSingletonDims
- ssGetInputPortDimensionSize
- ssGetOutputPortDimensionSize

See "SimStruct Functions — Alphabetical List" in *Developing S-Functions* for more information.

New Level-2 M-File S-Function Example

This release includes a new Level-2 M-file S-function example in
`sfundemos.mdl`. The Simulink model `msfcndemo_varpulse.mdl` uses the
S-function `msfcn_varpulse.m` to create a variable-width pulse generator.

Version 6.6.1 (R2007a+) Simulink Software

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	No

Version 6.6 (R2007a) Simulink Software

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	No

New features and changes introduced in this version are

- "Multidimensional Signals" on page 154

- "New Block Parameters" on page 158

- "GNU Compiler Upgrade" on page 158

- "Changes to Concatenate Block" on page 158

- "Changes to Assignment Block" on page 159

- "Changes to Selector Block" on page 160

- "Improved Model Advisor Navigation and Display" on page 161

- "Change to Simulink.ModelAdvisor.getModelAdvisor Method" on page 161

- "New Simulink Blocks" on page 162

- "Change to Level-2 M-File S-Function Block" on page 162

- "Model Dependency Analysis" on page 162

- "Model File Monitoring" on page 163

- "Legacy Code Tool Enhancements" on page 163

- "Continuous State Names" on page 164

- "Changes to Embedded MATLAB Function Block" on page 165

- "Referenced Models Support Non-Zero Start Time" on page 169

- "New Functions Copy a Model to a Subsystem or Subsystem to Model" on page 169

Multidimensional Signals

This release includes support for multidimensional signals, including:

- Sourcing of multidimensional signals
- Logging or displaying of multidimensional signals
- Large-scale modeling applications, such as those from model referencing
- Buses and nonvirtual buses
- Code generation with Real-Time Workshop software
- S-functions, including Level-2 M-File S-functions
- Stateflow charts

For further details, see:

- "Multidimensional Signals in Simulink Blocks" on page 155
- "Multidimensional Signals in S-Functions" on page 157

Simulink software supports signals with up to 32 dimensions. Do not use signals with more than 32 dimensions.

Multidimensional Signals in Simulink Blocks

The following blocks were updated to support multidimensional signals. See "Signal Dimensions" in the Simulink documentation for further details.

- Abs
- Assignment
- Bitwise Operator
- Bus Assignment
- Bus Creator
- Bus Selector
- Compare to Constant
- Compare to Zero
- Complex to Magnitude-Angle
- Complex to Real-Imag
- Concatenate
- Constant
- Data Store Memory
- Data Store Read
- Data Store Write
- Data Type Conversion
- 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 M-File S-Functions

To use multidimensional signals in Level-2 M-file S-functions, you must set the new `Simulink.MSFcnRunTimeBlock` property, `AllowSignalsWithMoreThan2D`.

New Block Parameters

This release introduces the following common block parameters.

- `PreCopyFcn`: Allows you to assign a function to call before the block is copied. See "Block Callback Parameters" in the Simulink documentation for details.

- `PreDeleteFcn`: Allows you to assign a function to call before the block is deleted. See "Block Callback Parameters" in the Simulink documentation for details.

- `StaticLinkStatus`: Allows you to obtain the link status of a block without updating out-of-date reference blocks See "Determining Link Status" in the Simulink documentation for details.

GNU Compiler Upgrade

This release upgrades the GNU® compiler to GCC 4.0.3 on Mac® platforms and GCC 4.1.1 on Linux platforms. The Fortran runtime libraries for the previous GCC 3.x versions are no longer included with MATLAB.

Compatibility Considerations

C, C++, or Fortran MEX-files built with the previous 3.x version of the GCC compiler are not guaranteed to load in this release. Rebuild the source code for these S-functions using the new version of the GCC compiler.

Changes to Concatenate Block

This release includes the following changes to the Concatenate block:

- Its **Mode** parameter provides two settings, namely, `Vector` and `Multidimensional array`.

- Its parameter dialog box contains a new option, **Concatenate dimension**, specifying the output dimension along which to concatenate the input arrays.

- The block displays a new icon when its **Mode** parameter is set to `Multidimensional array`.

This release updates Concatenate blocks when loading models created in previous releases.

Changes to Assignment Block

This release includes the following changes to the Assignment block:

- Enter the number of dimensions in the **Number of output dimensions** parameter, then configure the input and output with the **Index Option**, **Index**, and **Output Size** parameters.

- The parameter dialog box has the following new parameters:
  - **Number of output dimensions**
  - **Index Option**
  - **Index**
  - **Output Size**

- The **Initialize output (Y)** parameter replaces **Output (Y)** and has renamed options.

- The **Action if any output element is not assigned** parameter replaces **Diagnostic if not all required dimensions populated**.

- The block displays a new icon depending on the value of **Number of input dimensions** and the **Index Option** settings.

The following parameters are obsolete:

- **Input type**
- **Use index as start value**
- **Source of element indices**
- **Elements**
- **Source of row indices**
- **Rows**
- **Source of column indices**
- **Columns**

• **Output dimensions**

This release updates Assignment blocks when loading models created in previous releases.

Changes to Selector Block

This release includes the following changes to the Selector block:

• Enter the number of dimensions in the **Number of input dimensions** parameter, then configure the input and output with the **Index Option**, **Index**, and **Output Size** parameters.

• The parameter dialog box has the following new parameters:

  - **Number of input dimensions**
  - **Index Option**
  - **Index**
  - **Output Size**

• The behavior of the **Sample time** parameter has changed. See the Selector block **Sample time** parameter for details.

• The block displays a new icon depending on the value of **Number of input dimensions** and the **Index Option** settings.

The following parameters are obsolete:

• **Input type**
• **Use index as starting value**
• **Source of row indices**
• **Rows**
• **Source of column indices**
• **Columns**
• **Output port dimensions**

This release updates Selector blocks when loading models created in previous releases.

Improved Model Advisor Navigation and Display

This release improves the Model Advisor graphical user interface (GUI) for navigating lists of checks and viewing the status of completed checks. While Model Advisor functionality and content are largely unchanged from R2006b, the Model Advisor checks display and are navigated differently than in previous versions, and the generated Model Advisor report, if requested, displays in a MATLAB web browser window that is separate from the Model Advisor GUI.

To exercise the new features, open Model Advisor for a model (for example, enter `modeladvisor('vdp')` at the MATLAB command line) and then follow the instructions in the Model Advisor window. For more information about Model Advisor navigation and display, see "Consulting the Model Advisor" in the Simulink documentation.

Change to Simulink.ModelAdvisor.getModelAdvisor Method

In this release, when using the `getModelAdvisor` method defined by the `Simulink.ModelAdvisor` class to change Model Advisor working scope to a different model, you must either close the previous model or invoke the `getModelAdvisor` method with `'new'` as the second argument. For example, if you previously set scope to modelX with

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
```

and you want to change scope to modelY, you must either close modelX or use

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY', 'new');
```

If you try to change scope between models without the `'new'` argument, an error message is displayed.

Compatibility Considerations

In previous releases, you could change Model Advisor working scope without closing the current session. This is no longer allowed.

If your code contains a code pattern such as the following,

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
...
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY');
```

you must add the `'new'` argument to the second and subsequent invocations of `getModeladvisor`. For example:

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
...
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY', 'new');
```

Alternatively, you can close ModelX before issuing `Simulink.ModelAdvisor.getModelAdvisor('modelY')`.

New Simulink Blocks

This release introduces the following blocks:

- The Permute Dimensions block enables you to rearrange the dimensions of a multidimensional signal.

- The Squeeze block enables you to remove singleton dimensions from a multidimensional signal.

Change to Level-2 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 software. (see Model File Change Notification in "Managing Model Versions").

- Warning to notify the user if multiple models or libraries with the same name exist, as Simulink software may not be using the one the user expects. (see "Shadowed Files").

Legacy Code Tool Enhancements

- New fields in the Legacy Code Tool data structure: `InitializeConditionsFcnSpec` and `SampleTime`. `InitializeConditionsFcnSpec` defines a function specification for a reentrant function that the S-function calls to initialize and reset states. `SampleTime` allows you to specify whether sample time is inherited from the source block, represented as a tunable parameter, or fixed.

- Support for state (persistent memory) arguments in registered function specifications.

- Support for complex numbers specified for input, output, and parameter arguments in function specifications. This support is limited to use with Simulink built-in data types.

- Support for multidimensional arrays specified for input and output arguments in function specifications. Previously, multidimensional array support applied to parameters only.

- Ability to apply the `size` function to any dimension of function input data—input, output, parameter, or state. The data type of the `size` function's return value can be any type except complex, bus, or fixed-point.

- A new Legacy Code Tool option, `'backward_compatibility'`, which you can specify with the `legacy_code` function. This option enables Legacy Code Tool syntax, as made available from MATLAB Central in releases prior to R2006b, for a given MATLAB session.

- The following new demos:

```
sldemo_lct_sampletime
sldemo_lct_work
sldemo_lct_cplxgain
```

        sldemo_lct_ndarray

For more information, see

- "Integrating Existing C Functions into Simulink Models with the Legacy Code Tool" in the Writing S-Functions documentation
- "Automating Generation of Files for Fully Inlined S-Functions Using Legacy Code Tool" in the Real-Time Workshop documentation
- legacy_code function reference page

Compatibility Considerations

If you are using a version of the Legacy Code Tool that was accessible from MATLAB Central before R2006b, the syntax for using the tool differs from the syntax currently supported by Simulink software. To continue using the old style syntax, for example, legacy_code_initialize.m, issue the following call to legacy_code for a given MATLAB session:

```
legacy_code('backward_compatibility');
```

Continuous State Names

State names can now be assigned in those blocks that employ continuous states. The names are assigned with the ContinuousStateAttributes "Block-Specific Parameters" parameter, or in the Blocks Parameter dialog box.

The following blocks support continuous state names:

- Integrator
- State-Space
- Transfer Fcn
- Variable Transport Delay
- Zero-Pole

Logging of continuous states is illustrated in the sldemo_hydrod demo.

Changes to Embedded MATLAB Function Block

This release introduces the following changes to the Embedded MATLAB Function block:

- "New Function Checks M-Code for Compliance with Embedded MATLAB Subset" on page 165

- "Support for Multidimensional Arrays" on page 165

- "Support for Function Handles" on page 166

- "Enhanced Support for Frames" on page 166

- "New Embedded MATLAB Runtime Library Functions" on page 166

- "Using & and | Operators in Embedded MATLAB Function Blocks" on page 168

- "Calling get Function from Embedded MATLAB Function Blocks" on page 169

- "Documentation on Embedded MATLAB Subset has Moved" on page 169

New Function Checks M-Code for Compliance with Embedded MATLAB Subset

Embedded MATLAB function blocks introduce a new function, Embedded MATLAB MEX (emlmex), that checks M-code for compliance with the syntax and semantics of the Embedded MATLAB subset. You can add Embedded MATLAB-compliant code to Embedded MATLAB Function blocks and Truth Table blocks in Simulink models. For more information, see "Working with Embedded MATLAB MEX" in the Embedded MATLAB documentation.

Support for Multidimensional Arrays

Embedded MATLAB Function blocks now support multidimensional signals and parameter data, where the number of dimensions can be greater than 2. This feature is fully integrated with support for multidimensional signals in Simulink software. Supported functions in the "Embedded MATLAB 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 the Embedded MATLAB Subset" in the Embedded MATLAB documentation.

Enhanced Support for Frames

Embedded MATLAB Function blocks can now input and output frame-based signals directly in Simulink models. You no longer need to attach Frame Conversion blocks to inputs and outputs to achieve this functionality. See "Working with Frame-Based Signals" in the Simulink documentation.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide 31 new runtime library functions in the following categories:

- "Casting Functions" on page 167

- "Derivative and Integral Functions" on page 167

- "Discrete Math Functions" on page 167

- "Exponential Functions" on page 167

- "Filtering and Convolution Functions" on page 167

- "Logical Operator Functions" on page 167

- "Matrix and Array Functions" on page 167

- "Polynomial Functions" on page 168

- "Set Functions" on page 168

- "Specialized Math" on page 168

- "Statistical Functions" on page 168

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 Function blocks no longer support & and | operators in `if` and `while` conditional statements.

Compatibility Considerations. In prior releases, these operators compiled without error, but their short-circuiting behavior was not implemented correctly. Substitute && and || operators instead.

Calling get Function from Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now support the 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 a Model" and "Specifying a Simulation Start and Stop Time" for information about these capabilities. See "Referencing Configuration Sets" on page 172 for information about a convenient way to give all models in a hierarchy the same configuration parameters, including simulation start time.

New Functions Copy a Model to a Subsystem or Subsystem to Model

Two new functions exist that you can use to copy contents between a block diagram and a subsystem.

Simulink.BlockDiagram.copyContentsToSubSystem
    Copies the contents of a block diagram to an empty subsystem.

`Simulink.SubSystem.copyContentsToBlockDiagram`
    Copies the contents of a subsystem to an empty block diagram.

For details, see the reference documentation for each function.

New Functions Empty a Model or Subsystem

Two new functions exist that you can use to delete the contents of a block diagram or subsystem.

`Simulink.BlockDiagram.deleteContents`
    Deletes the contents of a block diagram.

`Simulink.SubSystem.deleteContents`
    Deletes the contents of a subsystem.

For details, see the reference documentation for each function.

Default for Signal Resolution Parameter Has Changed

In the Configuration Parameters dialog, **Diagnostics > Data Validity** pane, the default setting for **Signal resolution** is now **Explicit only**. Previously, the default was **Explicit and warn implicit**. Equivalently, the default value of the `SignalResolutionControl` parameter is now `UseLocalSettings` (previously `TryResolveAllWithWarnings`). See "Diagnostics Pane: Data Validity" for more information.

Compatibility Considerations

Due to this change, labeling a signal is no longer enough to cause it to resolve by default to a signal object. You must also do one of the following:

- In the signal's Signal Properties dialog, select **Signal name must resolve to Simulink data object** and specify a `Simulink.Signal` object in the **Signal name** field. Simulink software then resolves that signal to that signal object.

- In the Configuration Parameters dialog, set **Diagnostics > Data Validity > Signal resolution** to `Explicit and warn implicit` (to post warnings) or `Explicit and implicit` (to suppress warnings). Simulink software then resolves all labeled signals to signal objects by matching their names, posting a warning of each resolution if so directed.

Models built in R2007a will default to **Explicit only**. Models created in previous versions will retain the **Signal resolution** value with which they were saved, and will run as they did before. New models may therefore behave differently from existing models that retain the previous default behavior. To specify the previous default behavior in a new model, change **Signal resolution** to **Explicit and warn implicit**.

Conversion Function. 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 software provides the `disableimplicitsignalresolution` function, which you can use to change settings throughout a model so that it does not use implicit signal resolution. See the function's reference documentation, or type:

```
help disableimplicitsignalresolution
```

171

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 "Setting Up 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 software 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 software provides diagnostics that report cases where buses are mixed with vectors, and includes capabilities that you can use to upgrade a model to eliminate such mixtures, as described in the following sections of the Simulink documentation:

- "Using Composite Signals" — A new chapter in R2007a that describes the specification and use of composite signals.

- "Avoiding Mux/Bus Mixtures" — Ambiguities that arise when composite signal types are intermixed, and the tools available for eliminating such mixtures.

- "Using Diagnostics for Mux/Bus Mixtures" — Two diagnostic options for detecting mixed composite signals: "Mux blocks used to create bus signals" and "Bus signal treated as vector".

- "Using the Model Advisor for Mux/Bus Mixtures" — Model Advisor checks that detect mixed composite signals and recommend alternatives.

- Bus to Vector — A block that you can insert into a bus used implicitly as a vector to explicitly convert the bus to a vector.

- Simulink.BlockDiagram.addBusToVector — A function that creates a report of every bus used implicitly as a vector, and optionally inserts a Bus to Vector block into every such bus, replacing the implicit use with an explicit conversion.

Enhanced Support for Tunable Parameters in Expressions

Expressions that index into tunable parameters, such as `P(1)+P(2)/P(i)`, retain their tunability in generated code, including simulation code that is generated for a referenced model. Both the indexed parameter and the index itself can be tuned.

Parameter expressions of the form P(i) retain their tunability if all of the following are true:

- The index i is a constant or variable of `double` datatype

- `P` is a 1D array, or a 2D array with one row or one column, of `double` datatype

- `P` does not resolve to a mask parameter, but to a variable in the model or the base workspace

New Loss of Tunability Diagnostic

Previously, any loss of tunability generated a warning. In R2007a, you can use the **Loss of Tunability** diagnostic to control whether loss of tunability is ignored or generates a warning or error. See "Detect loss of tunability" for details.

Port Parameter Evaluation Has Changed

Previously, resolution of port parameters of a masked subsystem began within the subsystem, which could violate the integrity of the mask. For example, if a subsystem mask defines parameter A, and a port of the subsystem uses A to set some port attribute, resolving A by starting within the masked block makes A externally visible, though it should be visible only within the mask.

To fix this problem, in R2007a masked subsystem port parameter resolution starts in the containing system rather than within the masked subsystem, then proceeds hierarchically upward as it did before. This change preserves the integrity of the masked subsystem, but can change model behavior if any subsystem port previously depended for resolution on a variable defined within the mask.

Compatibility Considerations

A model whose ports did not reference variables defined within a mask are unaffected. A model that resolved any port parameter by accessing a variable within a masked block may behave differently or become vulnerable to future changes in behavior, as follows:

- If the port parameter's value cannot be evaluated, because the evaluation would require access to a variable defined only within the mask, an error occurs.

- If an appropriate variable exists outside the mask but has a different value than the corresponding variable within the mask, no error occurs, but model behavior may change.

- If an appropriate variable exists and has the same value inside and outside the mask, no behavioral change occurs, but later changes to the variable outside the mask may have unexpected effects.

To ensure correct results, change the model as needed so that any port parameter that previously depended on any variables defined within a mask give the intended results using the new resolution search path.

Data Type Objects Can Be Passed Via Mask Parameters

Previously, if a masked subsystem contained a block that needed to specify a data type using a data type object, the block could access the object only in the base workspace. The data type object could *not* be passed into the subsystem through a mask parameter. Parameterizing data types used by blocks under a mask was therefore not possible.

To support parameterized data types inside masked subsystems, you can now use a mask parameter to pass a data type object into a subsystem. Blocks in the subsystem can then use the object to specify data types under the mask.

Expanded Options for Displaying Subsystem Port Labels

This release provides an expanded set of options for displaying port labels on a subsystem block. The options include displaying:

- The label on the corresponding port block

- The name of the corresponding port block

- The name of the signal connected to the corresponding block

See the documentation for the **Show Port Labels** option on the Subsystem block's parameter dialog box for more information.

Model Explorer Customization Option Displays Properties of Selected Object

This release introduces a `Selection Properties` node to the Model Explorer's **Customize Contents** pane. The node allows you to customize the Model Explorer's **Contents** pane to display only the properties of the currently selected object. See "The Model Explorer: Overview" for more information.

Change to PaperPositionMode Parameter

In this release, when exporting a diagram as a graphic with the PaperPositionMode model parameter set to `auto`, Simulink software sizes the exported graphic to be the same size as the diagram's image on the screen when viewed at normal size. When PaperPositionMode is set to `manual`, Simulink software sizes the exported image to have the height and width specified by the model's PaperPosition parameter.

Compatibility Considerations

In previous releases, a model's PaperPosition parameter determined the size of the exported graphic regardless of the setting of the model's PaperPositionMode parameter. To reproduce the behavior of previous releases, set the PaperPositionMode parameter to `manual`.

New Simulink.Bus.objectToCell Function

A new function, `Simulink.Bus.objectToCell`, is available for converting bus objects to a cell array that contains bus information. For details, see the description of `Simulink.Bus.objectToCell`.

Simulink.Bus.save Function Enhanced To Allow Suppression of Bus Object Creation

The `Simulink.Bus.save` function has been enhanced such that when using the `'cell'` format you have the option of suppressing the creation of bus objects when the saved M-file executes. To suppress bus object creation, specify the optional argument `'false'` when you execute the saved M-file.

For more detail, see the description of `Simulink.Bus.save`.

Change in Version 6.5 (R2006b) Introduced Incompatibility

A change introduced in Version 6.5 (R2006b) introduces an incompatibility between this release and releases preceding Version 6.5 (R2006b). See "Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error" on page 184 for more information.

Nonverbose Output During Code Generation

Simulink Accelerator now defaults to nonverbose output when generating code. A new parameter, `AccelVerboseBuild`, has been added to control how much information is displayed. See "Customizing the Build Process" for more information.

Version 6.5 (R2006b) Simulink Software

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" on page 179

- "Enhanced Lookup Table Blocks" on page 179

- "Legacy Code Tool" on page 179

- "Simulink Software Now Uses Internal MATLAB Functions for Math Operations" on page 180

- "Enhanced Integer Support in Math Function Block" on page 180

- "Configuration Set Updates" on page 181

- "Command to Initiate Data Logging During Simulation" on page 181

- "Commands for Obtaining Model and Subsystem Checksums" on page 182

- "Sample Hit Time Adjusting Diagnostic" on page 182

- "Function-Call Models Can Now Run Without Being Referenced" on page 182

- "Signal Builder Supports Printing of Signal Groups" on page 182

- "Method for Comparing Simulink Data Objects" on page 183

- "Unified Font Preferences Dialog Box" on page 183

- "Limitation on Number of Referenced Models Eliminated for Single References" on page 183

- "Parameter Objects Can Now Be Used to Specify Model Configuration Parameters" on page 183

- "Parameter Pooling Is Now Always Enabled" on page 184

- "Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error" on page 184

- "Changes to Integrator Block's Level Reset Options" on page 185

- "Embedded MATLAB Function Block Features and Changes" on page 185

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 *Developing S-Functions* for more information.

Simulink Software Now Uses Internal MATLAB Functions for Math Operations

In previous releases, Simulink software used the host compiler's C++ Math Library functions to perform most mathematical operations on floating-point data. Some of those functions produced results that were slightly inconsistent with MATLAB results. In this release, Simulink software calls the same internal routines that MATLAB calls for most trigonometric, exponential, and rounding and remainder operations involving floating-point data. This ensures that when Simulink and MATLAB products operate on the same platform, they produce the same numerical results.

In particular, Simulink software now performs mathematical operations with the same internal functions that MATLAB uses to implement the following M-functions:

- `sin`, `cos`, `tan`
- `asin`, `acos`, `atan`, `atan2`
- `sinh`, `cosh`, `tanh`
- `asinh`, `acosh`, `atanh`
- `log`, `log2`, `log10`
- `mod`, `rem`
- `power`

Note By default, in this release Real-Time Workshop software continues to use C Math Library functions in the code that it generates from a Simulink model.

Enhanced Integer Support in Math Function Block

The `sqrt` operation in the Math Function block now supports built-in integer data types.

Configuration Set Updates

This release includes the following changes to model configuration parameters and configuration sets.

- This release includes a new command, openDialog, that displays the **Configuration Parameters** dialog box for a specified configuration set. This command allows display of configuration sets that are not attached to any model.

- The attachConfigSet command now includes an allowRename option that determines how the command handles naming conflicts when attaching a configuration set to a model.

- This release includes a new attachConfigSetCopy command that attaches a copy of a specified configuration set to a model.

- The new **Sample hit time adjusting** diagnostic controls whether Simulink software notifies you when the solver has to adjust a sample time specified by your model to solve the model. The associated model parameter is TimeAdjustmentMsg.

- The default value of the **Multitask data store** diagnostic has changed from Warning to Error for new models. This change does not affect existing models.

- The name of the **Block reduction optimization** parameter has changed to **Block reduction.**

Command to Initiate Data Logging During Simulation

The command

```
set_param(bdroot, 'SimulationCommand', 'WriteDataLogs')
```

writes all logging variables during simulation. See "Logging Signals" for more information.

Commands for Obtaining Model and Subsystem Checksums

This release includes commands for obtaining model and subsystem checksums.

- `Simulink.BlockDiagram.getChecksum`

  Get checksum for a model. Simulink Accelerator software uses this checksum to control regeneration of simulation targets. You can use this command to diagnose target rebuild problems.

- `Simulink.SubSystem.getChecksum`

  Get checksum for a subsystem. Real-Time Workshop software uses this checksum to control reuse of code generated from a subsystem that occurs more than once in a model. You can use the checksum to diagnose code reuse problems. See "Determining Why Subsystem Code Is Not Reused".

Sample Hit Time Adjusting Diagnostic

The **Sample hit time adjusting** diagnostic controls whether Simulink software notifies you when the solver has to adjust a sample time specified by your model to solve the model. The associated model parameter is `TimeAdjustmentMsg`.

Function-Call Models Can Now Run Without Being Referenced

This release allows you to simulate a function-call model, i.e., a model that contains a root-level function-call trigger block, without having to reference the model. In previous releases, the function-call model had to be referenced by another model in order to be simulated.

Signal Builder Supports Printing of Signal Groups

This release adds printing options to the Signal Builder block's editor. It allows you to print waveforms displayed in the editor to a printer, file, the clipboard, or a figure window. For details, see "Printing, Exporting, and Copying Waveforms".

Method for Comparing Simulink Data Objects

This release introduces an `isContentEqual` method for Simulink data objects that allows you to determine whether a Simulink data object has the same property values as another Simulink data object. For more information, see "Comparing Data Objects".

Unified Font Preferences Dialog Box

In this release, the **Simulink Preferences** dialog box displays font settings for blocks, lines, and annotations on a single pane instead of on separate tabbed panes as in previous releases. This simplifies selection of font preferences.

Limitation on Number of Referenced Models Eliminated for Single References

In previous releases, all distinct models referenced in a model hierarchy counted against the limitation imposed by Microsoft Windows on the number of distinct referenced models that can occur in a hierarchy. In this release, models configured to be instantiable only once do not account against this limit. This means that a model hierarchy can reference any number of distinct models on Windows platforms as long as they are referenced only once and are configured to be instantiable only once (see "Simulink Model Referencing Limitations" for more information).

Parameter Objects Can Now Be Used to Specify Model Configuration Parameters

This release allows you to use `Simulink.Parameter` objects to specify model configuration as well as block parameters. For example, you can specify a model's fixed step size as `Ts` and its stop time as `20*Ts` where `Ts` is a workspace variable that references a parameter object. When compiling a model, Simulink software replaces a reference to a parameter object in a model configuration parameter expression with the object's value.

Compatibility Considerations

In previous releases, you could use expressions of the form `p.Value()`, where `p` references a parameter object, in model configuration parameter

expressions. Such expressions cause expression evaluation errors in this release when you compile a model. You should replace such expressions with a simple reference to the parameter object itself, i.e., replace `p.Value()` with `p`.

Parameter Pooling Is Now Always Enabled

In previous releases, the **Parameter Pooling** optimization was optional and was enabled by default. Due to internal improvements, disabling **Parameter Pooling** would no longer be useful in any context. The optimization is therefore part of standard R2006b operation, and has been removed from the user interface.

Compatibility Considerations

Upgrading a model to R2006b inherently provides the effect that enabling **Parameter Pooling** did in previous releases. No compatibility considerations result from this change. If the optimization was disabled in an existing model, a warning is generated when the model is first upgraded to R2006b. This warning requires no action and can be ignored.

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

In this release, attempting to reference an symbol in an uninitialized mask workspace generates an error. This can happen, for example, if a masked subsystem's initialization code attempts to set a parameter of a block that resides in a masked subsystem in the subsystem being initialized and one or more of the block's parameters reference variables defined by the mask of the subsystem in which it resides (see "Initialization Command Limitations" for more information).

Compatibility Considerations

In this release, updating or simulating models created in previous releases may generate unresolvable symbol error messages. This can happen if the model contains masked subsystems whose initialization code sets parameters on blocks residing in lower-level masked subsystems residing in the top-level masked subsystem. To eliminate these errors, change the initialization code to avoid the use of `set_param` commands to set parameters in lower-level masked subsystems. Instead, use mask variables in upper-level

masked subsystems to specify the values of parameters of blocks residing in lower-level masked subsystems. See "Defining Mask Parameters"for information on using mask variables to specify block parameter values.

Changes to Integrator Block's Level Reset Options

This release changes the behavior of the `level` reset option of the Integrator block. In releases before Simulink 6.3, the `level` reset option resets the integrator's state if the reset signal is nonzero or changes from nonzero in the previous time step to zero in the current time step. In Simulink 6.3, 6.4, and 6.4.1, the option resets the integrator only if the reset signal is nonzero. This release restores the `level` reset behavior of releases that preceded Simulink 6.3. It also adds a `level hold` option that behaves like the `level` reset option of Simulink 6.3, 6.4, and 6.4.1.

Compatibility Considerations

A model that uses the `level` reset option could produce results that differ in this release from those produced in Simulink 6.3, 6.4, and 6.4.1. To reproduce the results of previous releases, change the model to use the new `level hold` option instead.

Embedded MATLAB Function Block Features and Changes

Support for Structures

You can now define structures as inputs, outputs, local, and persistent variables in Embedded MATLAB Function blocks. With support for structures, Embedded MATLAB Function blocks give you the ability to read and write Simulink bus signals at inputs and outputs of Embedded MATLAB Function blocks. See "Working with 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 The Code Analyzer with Embedded MATLAB Code" 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.

- cov
- ifftshift
- std
- var

Discrete Math.

- gcd
- lcm

Exponential.

- `expm1`
- `log10`
- `log1p`
- `log2`
- `nextpow2`
- `nthroot`
- `reallog`
- `realpow`
- `realsqrt`

Interpolation and Computational Geometry.

- `cart2pol`
- `cart2sph`
- `pol2cart`
- `sph2cart`

Linear Algebra.

- `cond`
- `det`
- `ipermute`
- `kron`
- `permute`
- `planerot`
- `rand`
- `randn`
- `rank`

- shiftdim

- squeeze

- subspace

- trace

Logical.

- isstruct

Specialized Plotting.

- histc

Transforms.

- bitrevorder

Trigonometric.

- hypot

New Requirement for Calling MATLAB Functions from Embedded MATLAB Function Blocks

To call external MATLAB functions from Embedded MATLAB Function blocks, you must first declare the functions to be extrinsic. (External MATLAB functions are functions that have not been implemented in the Embedded MATLAB runtime library.) MATLAB Function blocks do not compile or generate code for extrinsic functions; instead, they send the function to MATLAB for execution during simulation. There are two ways to call MATLAB functions as extrinsic functions in Embedded MATLAB Function blocks:

- Use the new construct `eml.extrinsic` to declare the function extrinsic

- Call the function using `feval`

For details, see "Calling MATLAB Functions" in the Embedded MATLAB documentation.

Compatibility Considerations. Currently, Embedded MATLAB Function blocks use implicit rules to handle calls to external functions:

- For simulation, Embedded MATLAB Function blocks send the function to MATLAB for execution

- For code generation, Embedded MATLAB Function blocks check whether the function affects the output of the Embedded MATLAB function in which it is called. If there is no effect on output, Embedded MATLAB Function blocks proceed with code generation, but exclude the function call from the generated code. Otherwise, Embedded MATLAB Function blocks generate a compiler error.

In future releases, Embedded MATLAB Function blocks will apply these rules only to external functions that you call as extrinsic functions. Otherwise, they will compile external functions by default, potentially causing unpredictable behavior or generating errors. For reliable simulation and code generation, 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 Function blocks attempted to silently convert values returned by MATLAB functions to predeclared data type and sizes if a mismatch occurred. Now, such mismatches always generate an error, as in this example:

```
x = int8(zeros(3,3)); % Predeclaration
x = eval('5'); % Calls MATLAB function eval
```

This code now generates an error because the Embedded MATLAB function predeclares x as a 3–by-3 matrix, but MATLAB function returns x as a scalar double. To avoid errors, reconcile predeclared data types and sizes with the actual types and sizes returned by MATLAB function calls in your Embedded MATLAB Function blocks.

Embedded MATLAB Function Blocks Cannot Output Character Data

Embedded MATLAB Function blocks now generate an error if any of its outputs is character data.

Compatibility Considerations. In the previous release, Embedded MATLAB Function blocks silently cast character array outputs to `int8` scalar arrays. This behavior does not match MATLAB, which represents characters in 16–bit unicode.

Version 6.4.1 (R2006a+) Simulink Software

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 Software

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

- "Signal Object Initialization" on page 193
- "Icon Shape Property for Logical Operator Block" on page 193
- "Data Type Property of Parameter Objects Now Settable" on page 193
- "Range-Checking for Parameter and Signal Object Values" on page 193
- "Expanded Menu Customization" on page 194
- "Bringing the MATLAB Desktop Forward" on page 194
- "Converting Atomic Subsystems to Model References" on page 194
- "Concatenate Block" on page 194
- "Model Advisor Changes" on page 195
- "Built-in Block's Initial Appearance Reflects Parameter Settings" on page 195
- "Double-Click Model Block to Open Referenced Model" on page 196
- "Signal Logs Reflect Bus Hierarchy" on page 196
- "Tiled Printing" on page 196
- "Solver Diagnostic Controls" on page 196
- "Diagnostic Added for Multitasking Conditionally Executed Subsystems" on page 197

• "Embedded MATLAB Function Block Features and Changes" on page 197

Signal Object Initialization

This release introduces the use of signal objects to specify initial values for signals and states. This allows you to initialize signals or states in the model, not just those generated by blocks that have initial condition or value parameters. For details, see "Using Signal Objects to Initialize Signals and Discrete States" in the online Simulink documentation.

Icon Shape Property for Logical Operator Block

The Logical Operator block's parameter dialog box contains a new property, **Icon shape**, settings for which can be either `rectangular` or `distinctive`. If you select `rectangular` (the default), the block appears as it does in previous releases. If you select `distinctive`, the block appears as the IEEE® standard graphic symbol for the selected logic operator.

Data Type Property of Parameter Objects Now Settable

This release allows you to set the data type of a `Simulink.Parameter` object via either its `Value` property or via its `Data type` property. In previous releases, you could specify the data type of a parameter object only by setting the object's `Value` property to a typed value expression.

Range-Checking for Parameter and Signal Object Values

This release introduces range checking for `Simulink.Parameter` and `Simulink.Signal` objects. Simulink software checks whether a parameter's **Value** or a signal's **Initial value** falls within the values you specify for the object's **Minimum** and **Maximum** properties. If not, Simulink software generates a warning or error.

Compatibility Considerations

Previous releases ignored such violations since the `Minimum` and `Maximum` properties were intended for use in documenting parameter and signal objects. In this release, Simulink software displays a warning if you load

a parameter object or a signal object does not specify a valid range or its value falls outside the specified range. If you get such a warning, change the parameter or signal object's `Value` or `Minimum` or `Maximum` values so that the `Value` falls within a valid range.

Expanded Menu Customization

The previous release of Simulink software allows you to customize the Simulink editor's **Tools** menu. This release goes a step further and allows you to customize any Simulink (or Stateflow) editor menu (see "Customizing the Simulink User Interface" in the online Simulink documentation).

Bringing the MATLAB Desktop Forward

The Model Editor's **View** menu includes a new command, **MATLAB Desktop**, that brings the MATLAB desktop to the front of the windows displayed on your screen.

Converting Atomic Subsystems to Model References

This release adds a command, **Convert to Model Block**, to the context (right-click) menu of an atomic subsystem. Selecting this command converts an atomic subsystem to a model reference . See Atomic Subsystem and "Converting a Subsystem to a Referenced Model" for more information.

The function `sl_convert_to_model_reference`, which provided some of the same capabilities as **Convert to Model Block**, is obsolete and has been removed from the documentation. The function continues to work, so no incompatibility arises, but it posts a warning when called. The function will be removed in a future release.

Concatenate Block

The new Concatenate block concatenates its input signals to create a single output signal whose elements occupy contiguous locations in memory. The block typically uses less memory than the Matrix Concatenation block that it replaces, thereby reducing model memory requirements.

Compatibility Considerations

This release replaces obsolete Matrix Concatenation blocks with Concatenate blocks when loading models created in previous releases.

Model Advisor Changes

Model Advisor Tasks Introduced

This release introduces Model Advisor tasks for referencing models and upgrading a model to the current version of Simulink software. See "Consulting the Model Advisor" in the online Simulink documentation for more information.

Model Advisor API

This release introduces an application program interface (API) that enables you to run the Model Advisor from the MATLAB command line or from M-file programs. For example, you can use the API to create M-file programs that determine whether a model passes selected Model Advisor checks whenever you open, simulate, or generate code from the model. See "Running the Model Advisor Programmatically" in the online Simulink documentation for more information.

Built-in Block's Initial Appearance Reflects Parameter Settings

In this release, when you load a model containing nonmasked, built-in blocks whose appearance depends on their parameter settings, such as the Selector block, the appearance of the blocks reflect their parameter settings. You no longer have to update the model to update the appearance of such blocks.

Compatibility Considerations

In previous releases, model or block callback functions that use `set_param` to set a built-in, nonmasked block's parameters could silently put the block in an unusable state. In this release, such callbacks will trigger error messages if they put blocks in an unusable state.

Double-Click Model Block to Open Referenced Model

In this release, double-clicking a Model block that specifies a valid referenced model opens the referenced model, rather than the Block Parameters dialog box as in previous releases. To open the Block Parameters dialog box, choose **Model Reference Parameters** from the **Context** or **Edit** menu. See "Navigating a Model Block" for details.

Signal Logs Reflect Bus Hierarchy

In this release, signal logs containing buses reflect the structure of the buses themselves instead of flattening bus data as in previous releases (see `Simulink.TsArray`).

Tiled Printing

This release introduces a tiled printing option that allows you to distribute a block diagram over multiple pages. You can control the number of pages over which Simulink software distributes the block diagram, and hence, the total size of the printed image. See "Tiled Printing" in the online Simulink documentation for more information.

Solver Diagnostic Controls

In this release, the **Configuration Parameters** dialog box includes the following enhancements:

- The **Diagnostics** pane contains a new diagnostic, **Consecutive zero crossings violation**, that alerts you if Simulink software detects the maximum number of consecutive zero crossings allowed. You can specify the criteria that Simulink software uses to trigger this diagnostic using two new **Solver diagnostic controls** on the **Solver** pane:

  - **Consecutive zero crossings relative tolerance**

  - **Number of consecutive zero crossings allowed**

  For more information, see "Preventing Excessive Zero Crossings" in the online Simulink documentation.

- The **Solver** pane contains a new solver diagnostic control, **Number of consecutive min step size violations allowed**, that Simulink software

uses to trigger the **Min step size violation** diagnostic (see "Number of consecutive min steps" in the online Simulink documentation).

Diagnostic Added for Multitasking Conditionally Executed Subsystems

This release adds a sample-time diagnostic that detects an enabled subsystem in multitasking solver mode that operates at multiple rates or a conditionally executed subsystem that contain an asynchronous subsystem. Such subsystems can cause corrupted data or non-deterministic behavior in a real-time system using code generated from the model. See the documentation for the **Multitask Conditionally Executed Subsystem** diagnostic for more information.

Embedded MATLAB Function Block Features and Changes

Option to Disable Saturation on Integer Overflow

The properties dialog for Embedded MATLAB Function blocks provides a new **Saturate on Integer Overflow** check box that lets you disable saturation on integer overflow to generate more efficient code. When you enable saturation on integer overflow, Embedded MATLAB Function blocks add additional checks in the generated code to detect integer overflow or underflow. Therefore, it is more efficient to disable this option if your algorithm does not rely on overflow behavior. For more information, see "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 software, including [Slope Bias] scaling (see "Specifying Simulink Fixed Point Data Properties" in the online Simulink documentation). For information about the features added to the Fixed-Point Toolbox software, see "Fixed-Point Toolbox Release Notes".

Support for Integer Division

Embedded MATLAB Function blocks support the new MATLAB function `idivide`, which performs integer division with a variety of rounding options. It is recommended that the rounding option used for integer division in Embedded MATLAB Function blocks match the rounding option in the parent Simulink model.

The default rounding option for `idivide` is `'fix'`, which rounds toward zero. This option corresponds to the choice **Zero** in the submenu for **Signed integer division rounds to:**, a parameter that you can set in the Hardware Implementation Pane of the Configuration Parameters dialog in Simulink software (see "Hardware Implementation Pane" in the online Simulink documentation). If this parameter is set to **Floor** in the Simulink model that contains the Embedded MATLAB Function block, it is recommended that you pass the rounding option `'floor'` to `idivide` in the block.

For a complete list of Embedded MATLAB runtime library functions provided in this release, see "New Embedded MATLAB Runtime Library Functions" on page 198.

New Embedded MATLAB Runtime Library Functions

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

- "Integer Arithmetic" on page 199
- "Linear Algebra" on page 199
- "Logical" on page 200
- "Polynomial" on page 200

- "Trigonometric" on page 200

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 MATLAB Function Blocks

You can no longer set the FIMATH property CastBeforeSum to false for fixed-point data in Embedded MATLAB Function blocks.

Compatibility Considerations. The reason for the restriction is that Embedded MATLAB Function blocks do not produce the same numerical results as MATLAB when CastBeforeSum is false. In the previous release, Embedded MATLAB Function blocks set CastBeforeSum to false by default for the default FIMATH object. If you have existing models that contain Embedded MATLAB Function blocks in which CastBeforeSum is false, you will get an error when you compile or update your model. To correct the issue, you must set CastBeforeSum to true. To automate this process, you can run the utility slupdate either from the Model Advisor or by typing the following command at the MATLAB command line:

```
slupdate ('modelname')
```

where '*modelname*' is the name of the model containing the Embedded MATLAB Function block that generates the error. slupdate prompts you to update this property by selecting one of these options:

Option	Action
Yes	Updates the first occurrence of `CastBeforeSum=false` in Embedded MATLAB Function blocks in the offending model and then prompts you for each subsequent one found in the model.
No	Does not update any occurrences of `CastBeforeSum=false` in the offending model.
All	Updates all occurrences of `CastBeforeSum=false` in the offending model.

Note `slupdate` detects `CastBeforeSum=false` only in *default* `FIMATH` objects defined for Simulink software signals in Embedded MATLAB Function blocks. If you modified the `FIMATH` object in an Embedded MATLAB Function block, update `CastBeforeSum` manually in your model and fix the errors as they are reported.

Type Mismatch of Scalar Output Data in Embedded MATLAB Function Blocks Generates Error

Embedded MATLAB Function blocks now generate an error if the output type inferred by the block does not match the type you explicitly set for a scalar output.

Compatibility Considerations. In previous releases, a silent cast was inserted from the computed type to the set type when mismatches occurred. In most cases, you should not need to set the output type for Embedded MATLAB Function blocks. When you do, insert an explicit cast in your Embedded MATLAB script. For example, suppose you declare a scalar output *y* to be of type int8, but its actual type is double. Replace *y* with a temporary variable *t* in your script and then add the following code:

```
y = int8(t);
```

Implicit Parameter Type Conversions No Longer Supported in Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now generate an error if the type of a parameter inferred by the block does not match the type you explicitly set for the parameter.

Compatibility Considerations. In the previous release, if the type you set for a parameter did not match the actual parameter value, Embedded MATLAB Function blocks implicitly cast the parameter to the specified type. Now you receive a compile-time error when type mismatches occur for parameters defined in Embedded MATLAB Function blocks.

There are two workarounds:

- Change the scope of the data from **Parameter** to **Input**. Then, connect to the input port a **Constant** block that brings in the parameter and casts it to the desired type.

- Cast the parameter inside your Embedded MATLAB function to the desired type.

Fixed-Point Parameters Not Supported

Embedded MATLAB Function blocks generate a compile-time error if you try to bring a `fi` object defined in the base workspace into Embedded MATLAB Function blocks as a parameter.

There are two workarounds:

- Change the scope of the data from **Parameter** to **Input**. Then, connect to the input port a **Constant** block that brings in the parameter and casts it to fixed-point type.

- Cast the parameter inside your Embedded MATLAB function to fixed-point type.

Embedded MATLAB Function Blocks Require C Compiler for Windows 64

No C compiler ships with MATLAB and Simulink products on Windows 64. Because Embedded MATLAB Function blocks perform simulation through

code generation, you must supply your own MEX-supported C compiler to use these blocks. The C compilers available at the time of this writing for Windows 64 include Microsoft Visual Studio® 2005 and the Microsoft Platform SDK.

Version 6.3 (R14SP3) Simulink Software

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" on page 205

- "Block Enhancements" on page 207

- "Modeling Enhancements" on page 209

- "Simulation Enhancements" on page 211

- "User Interface Enhancements" on page 212

- "MEX-Files on Windows Systems" on page 213

- "Fixed-Point Functions No Longer Supported for Use in Signal Objects" on page 213

- "Parameter Object Expressions No Longer Supported in Dialog Boxes" on page 213

- "MEX-File Extension Changed" on page 214

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 "Defining Function-Call Models" in the Simulinkdocumentation for more information.

Using Noninlined S-Functions in Referenced Models. This release adds limited support for use of noninlined S-functions in models referenced by other models. For example, you can simulate a model that references models containing noninlined S-functions. However, you cannot use Real-Time Workshop software to generate a standalone executable (Real-Time Workshop target) for the model. See "Simulink Model Referencing Limitations" in the Simulink documentation for information on other limitations.

Referenced Models Without Root I/O Can Inherit Sample Times. Previous releases of Simulink software do not allow referenced models without root-level input or output ports to inherit their sample time. This release removes this restriction.

Referenced Models Can Use Variable Step Solvers. Previous releases of Simulink software do not allow models to reference models that require variable-step solvers. This release removes this restriction.

Model Dependency Graphs Accessible from the Tools Menu. This release adds a **Model Reference Dependency Graph** item to the Model Editor's **Tools** menu. The item displays a graph of the models referenced by the model displayed in the Model Editor. You can open any model in the dependency graph by clicking its node. See "Viewing a Model Reference Hierarchy" in the Simulinkdocumentation for more information.

Command That Converts Atomic Subsystems to Model References. This release introduces a MATLAB command that converts an atomic subsystem to a model reference. See `Simulink.SubSystem.convertToModelReference` in the Simulink Reference documentation for more information.

Model Reference Demos. This release has the following model reference demo changes:

- Model reference demo names are now prepended with `sldemo_`. For example, the demo `mdlref_basic.mdl` is now `sldemo_mdlref_basic.mdl`.

- You can no longer use the `mdlrefdemos` command from the MATLAB command prompt to access model reference demos. Instead, you can navigate to the Simulink demos tab either though the Help browser, or by typing `demos` at the command prompt, then navigating to the Simulink demos category and browsing the demos.

Block Enhancements

Variable Transport Delay, Variable Time Delay Blocks
This release replaces the Variable Transport Delay block of previous releases with two new blocks. The Variable Transport Delay block of previous releases implemented a variable time delay behavior, which is now implemented by the Variable Time Delay block introduced in this release. This release changes the behavior of the Variable Transport Delay block to model variable transport delay behavior, e.g., the behavior of a fluid flowing through a pipe.

Additional Reset Trigger for Discrete-Time Integrator Block
This release adds a `sampled level` trigger option for causing the Discrete-Time Integrator to reset. The new reset trigger is more efficient than the `level` reset option, but may introduce a discontinuity when integration resumes.

Note In Simulink 6.2 and 6.2.1, the level reset option behaves like the sampled level option in this release. This release restores the level reset option to its original behavior.

Input Port Latching Enhancements
This release includes the following enhancements to the signal latching capabilities of the Inport block.

Label Clarified for Triggered Subsystem Latch Option. The dialog box for an Inport block contains a check box to latch the signal connected to the system via the port. This check box applies only to triggered subsystems and hence is enabled only when the Inport block resides in a triggered subsystem. In this release, the label for the check box that selects this option has changed from **Latch (buffer) input** to **Latch input by delaying outside signal**. This change is intended to make it clear what the option does, i.e., cause the subsystem to see the input signal's value at the previous time step when the subsystem executes at the current time step (equivalent to inserting a Memory block at the input outside the subsystem). The Inport block's icon displays <Lo> to indicate that this option is selected.

Latch Option Added for Function-Call Subsystems. This release adds a check box labeled **Latch input by copying inside signal** to the Inport block's dialog box. This option applies only to function-call subsystems and hence is enabled only if the Inport block resides in a function-call subsystem. Selecting this option causes Simulink software to copy the signal output by the block into a buffer before executing the contents of the subsystem and to use this copy as the block's output during execution of the subsystem. This ensures that the subsystem's inputs, including those generated by the subsystem's context, will not change during execution of the subsystem. The Inport block's icon displays <Li> to indicate that this option is selected.

Improved Function-Call Inputs Warning Label

In previous releases, the dialog box for a function-call subsystem contains a check box labeled **Warn if function-call inputs arise inside called context**. This release changes the label to **Warn if function-call inputs are context-specific**. This change is intended to indicate more clearly the warning's purpose, i.e., to alert you that some or all of the function-call inputs come from the function-call subsystem's context and hence could change while the function-call subsystem is executing.

Note In this release, you can avoid this function-call inputs problem by selecting the **Latch input by copying inside signal** option on the subsystem's Inport blocks (see "Latch Option Added for Function-Call Subsystems" on page 208).

Modeling Enhancements

Annotations
This release introduces the following enhancements to model annotations:

- Annotation properties dialog box (see "Annotations Properties Dialog Box" in the Simulinkdocumentation)

- Annotation callback functions (see "Annotation Callback Functions" in the Simulinkdocumentation)

- Annotation application programming interface (see "Annotations API" in the Simulinkdocumentation)

Custom Signal Viewers and Generators
This release allows you to add custom signal viewers and generators so that you can manage them in the Signal & Scope Manager. See "Adding Custom Viewers and Generators" in the Simulinkdocumentation for further details.

Model Explorer Search Option
This release adds an `Evaluate Property Values During Search` option to the Model Explorer. This option applies only for searches by property value. If enabled, the option causes the Model Explorer to evaluate the value of each property as a MATLAB expression and compare the result to the search value. If disabled (the default), the Model Explorer compares the unevaluated property value to the search value.

Using Signal Objects to Assign Signal Properties
Previous releases allow you to use signal objects to check signal property values assigned by signal sources. This release allows you, in addition, to use signal objects to assign values to properties not set by signal sources. See `Simulink.Signal` in the Simulink Reference documentation for more information.

Bus Utility Functions

This release introduces the following bus utility functions:

- `Simulink.Bus.save`

- `Simulink.Bus.createObject`

- `Simulink.Bus.cellToObject`

Fixed-Point Support in Embedded MATLAB Function Blocks

In this release, the Embedded MATLAB Function block supports 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 Function Blocks

Embedded MATLAB Function blocks now supports input triggers
and function-call outputs. See "Ports and Data Manager" in the
Simulinkdocumentation for more information.

Find Options Added to the Data Object Wizard

This release adds find options to the **Data Object Wizard**. The options
enable you to restrict the search for model data to specific kinds of objects. See
"Data Object Wizard" in the Simulinkdocumentation for more information.

Simulation Enhancements

Viewing Logged Signal Data

This release can display logged signal data in the MATLAB **Times Series Tools** viewer on demand or whenever a simulation ends or you pause a simulation. See "Viewing Logged Signal Data" in the Simulink documentation for more information.

Importing Time-Series Data

In this release, root-level Inport blocks can import data from time-series (see `Simulink.Timeseries` in the Simulink Reference documentation) and time-series array (see `Simulink.TSArray` in the Simulink Reference documentation) objects residing in the MATLAB workspace. See "Importing Data from a Workspace" in the Simulinkdocumentation for more information. From Workspace blocks can also import time-series objects. The ability to import time-series objects allows you to use data logged from one simulation as input to another simulation.

Using a Variable-Step Solver with Rate Transition Blocks

Previous releases of Simulink software generate an error if you try to use a variable-step solver to solve a model that contains Rate Transition blocks. This release allows you to use variable-step as well as fixed-step solvers to simulate a model. Note that you cannot generate code from a model that uses a variable-step solver. However, you may find it advantageous, in some cases, to use a variable-step solver to test aspects of the model not directly related to code generation. This enhancement allows you to switch back and forth between the two types of solver without having to remove and reinsert Rate Transition blocks.

Additional Diagnostics

This release adds the following simulation diagnostics:

- "Enforce sample times specified by Signal Specification blocks" in the online Simulink documentation

- "Extraneous discrete derivative signals" in the online Simulink documentation

211

- "Detect read before write" in the online Simulink documentation

- "Detect write after read" in the online Simulink documentation

- "Detect write after write" in the online Simulink documentation

Data Integrity Diagnostics Pane Renamed, Reorganized

This release changes the name of the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box to the **Data Validity** pane. It also reorganizes the pane into groups of related diagnostics. See "Diagnostics Pane: Data Validity" in the online Simulink documentation for more information.

Improved Sample-Time Independence Error Messages

When you enable the `Ensure sample time independent` solver constraint (see "Periodic sample time constraint" for more information), Simulink software generates several error messages if the model is not sample-time independent. In previous releases, these messages were not specific enough for you to determine why a model failed to be sample-time independent. In this release, the messages point to the specific block, signal object, or model parameter that causes the model not to be sample-time independent.

User Interface Enhancements

Model Viewing

This release adds the following model viewing enhancements:

- A command history for pan and zoom commands (see "Viewing Command History" in the Simulinkdocumentation)

- Keyboard shortcuts for panning model views (see "Model Viewing Shortcuts" in the Simulinkdocumentation)

Customizing the Simulink User Interface

This release allows you to use M-code to perform the following customizations of the standard Simulink user interface:

- Add custom commands to the Model Editor's **Tools** menu (see "Disabling and Hiding Dialog Box Controls" in the Simulinkdocumentation)

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

MEX-Files on Windows Systems

In this release, the extension for files created by the MATLAB `mex` command on Windows systems has changed from `dll` to `mexw32` or `mexw64`.

Compatibility Considerations

If you have implemented any S-functions in C, Ada, or Fortran or have models that reference other models, you should

- Recreate any `mexopts.bat` files (other than the one in your MATLAB preferences directory) that you use to build S-functions and model reference simulation targets

- Rebuild your S-functions

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.

MEX-File Extension Changed

In this release, the extension for files created by the MATLAB `mex` command has changed from `dll` to `mexw32` (and `mexw64`).

Compatibility Considerations

If you use a `mexopts.bat` file other than the one created by the `mex` command in your MATLAB preferences directory to build Accelerator targets, you should recreate the file from the `mexopts.bat` template that comes with this release.

Version 6.2 (R14SP2) Simulink Software

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

- "Multiple Signals on Single Set of Axes" on page 215
- "Logging Signals to the MATLAB Workspace" on page 215
- "Legends that Identify Signal Traces" on page 216
- "Displaying Tic Labels" on page 216
- "Opening Parameters Dialog Box" on page 216
- "Rootlevel Input Ports" on page 216

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 base workspace. See "Logging Signals" for more information.

Legends that Identify Signal Traces

Viewers can now display a legend that identifies signal traces.

Displaying Tic Labels

Viewers can now display tic labels both inside and outside scope axes.

Opening Parameters Dialog Box

You can open a viewer's parameters dialog box by right-clicking on the viewer scope.

Rootlevel Input Ports

Compatibility Considerations

If you save a model with rootlevel input ports in this release and load it in a previous release, you will get the following warning:

```
Warning: model, line xxx block_diagram does not have a parameter
named 'SignalName'.
```

You can safely ignore this warning.

Version 6.1 (R14SP1) Simulink Software

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:

- "Changed Source Dialog Box Behavior" on page 217

- "Changed Model Explorer Source Behavior" on page 218

- "Affected Blocks" on page 218

- "Model Load Warnings" on page 219

In this release, Simulink software 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 software 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 software 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 product

  - Communications Blockset product

  - Embedded Target for TI's C6000™ DSP

  - Signal Processing Blockset product (formerly DSP Blockset)

  - Simulink Fixed Point product (formerly Fixed-Point Blockset)

  - System Identification Toolbox™ product

  - xPC Target™ product

  - xPC TargetBox® product

  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 Software

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" on page 221
- "Configuration Sets" on page 221
- "Model Referencing" on page 221
- "Model Workspaces" on page 222
- "Implicit Fixed-Step Solver" on page 223
- "The Signal and Scope Manager" on page 223
- "Data Object Type Enhancements" on page 223
- "Block Enhancements" on page 224
- "Signal Enhancements" on page 227
- "Rate Transition Enhancements" on page 228
- "Execution Context Enhancements" on page 229
- "Algebraic Loop Minimization" on page 229
- "Level-2 M-File S-Functions" on page 229
- "Panning Model Diagrams" on page 230
- "MATLAB Data Type Conversions" on page 230

- "Signal Object Resolution Changes" on page 230

- "Loading Models Containing Non-ASCII Characters" on page 231

- "Change in Sample Time Behavior of Unary Minus Block" on page 232

- "Initial Output of Conditionally Executed Subsystems" on page 232

- "Execution Context Default Changes" on page 232

- "Simulink Accelerator Switch Blocks Can Abort Code Generation" on page 232

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: Overview" 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 "Setting Up 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 "Configuration Parameters Dialog Box" 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 Real-Time Workshop products 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 a Model" 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 "Simulink Model Referencing Limitations".

Model Workspaces

In this release, Simulink software 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 "Using 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 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 "Introducing the Signal and 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 "Simulink 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 software 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 Real-Time Workshop software.

- The Model block allows you to include other models in a model (see "Model Referencing" on page 221).

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 software 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 built in 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 software propagated buses only through virtual blocks, such as subsystems. In this release, Simulink software 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 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 software 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 the Simulink 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 "Automatically handle rate transition for data transfer" 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 software 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 software 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 software 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 software displays a warning. You can also specify that Simulink software 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 products to support character sets from different encoding systems.

Compatibility Considerations

This change causes Simulink software 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 software checks the characters in the model against the current encoding setting of MATLAB. If they are incompatible, Simulink software 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 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.

Simulink Accelerator Switch Blocks Can Abort Code Generation

In Release 13, accelerator code generation aborted for the case of a Switch block configured with the **Criteria for passing first input** set to u2 ~=0,

with vector inputs of width greater than the `RollThreshold` (5). Code generation aborted with the following message:

```
%exit directive: Real-Time Workshop Fatal in block: "/B_1_28",
block type "Switch": No parameters to roll.
```

This release fixes the problem.

Version 5.1 (R13SP1) Simulink Software

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" on page 234

- "Enhanced Debugger" on page 235

- "Context-Sensitive Data Typing of Tunable Parameters" on page 237

- "Conditional Execution Behavior" on page 239

- "Function-Call Subsystem Enhancements" on page 242

- "External Increment Option Added To For Iterator Block" on page 242

- "Performance Improvements" on page 243

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 software 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 software 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 software 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 software 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

237

this release, by contrast, a typecast never occurs because the block itself determines the appropriate type for the parameter. Hence, in this release, Simulink software 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.
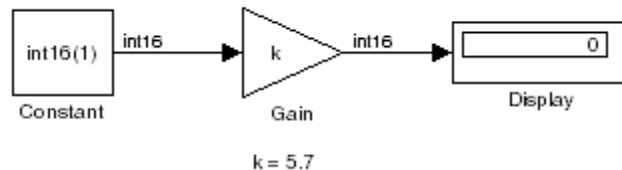
Assume that the model's **Inline parameters** setting is selected (thereby making parameters nontunable by default) and the model declares k as a tunable parameter on the **Model Configuration Parameters** dialog box. Also assume that the user has specified the value of k on the MATLAB command line as 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 software to assign different data types to the same parameter, which is impossible. Therefore, in this release, Simulink software signals an error to alert the user to the conflict. By contrast, in Release 13, Simulink software 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.
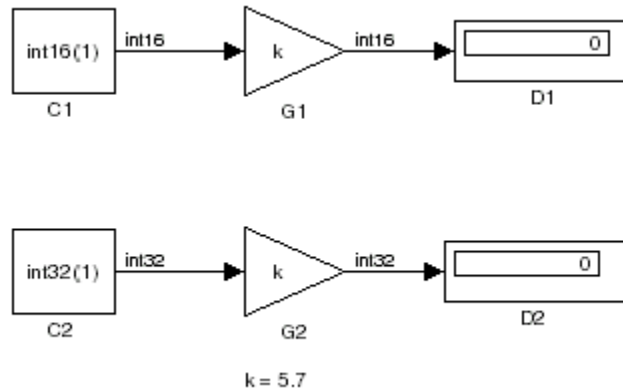
The two Gain blocks in this model both use k, a tunable parameter of unspecified type, as their gain parameter. Computing the outputs of the blocks requires that the gain parameter be of types `int16` and `int32`, respectively. In Release 13, Simulink software regards the data type of k to be `double` and the Gain blocks use typecasts to convert k to the required type in each case. Simulink software 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.
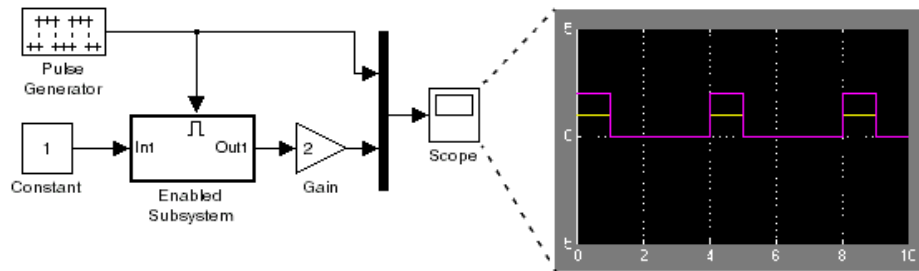
Simulink software computes the outputs of the Constant block and Gain Block only when the Enabled Subsystem executes (i.e., at time steps 0, 4, 8, and so on). This is because the output of the Constant block is required and the input of the Gain block changes only when the Enabled Subsystem executes. When CE behavior is off, Simulink software 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 software 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 software defines an *execution context* as a set of blocks to be executed as a unit. At model compilation time, Simulink software 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 software 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 software 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 software 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 software 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.
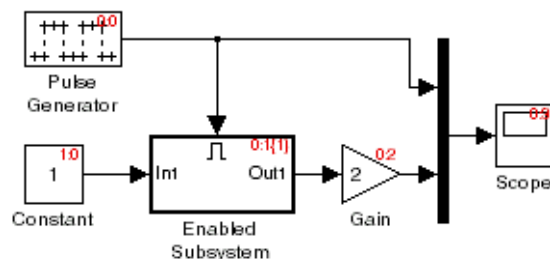
In this example, the sorted order index of the enabled subsystem is `0:1{1}`. The `0` indicates that the enable subsystem resides in the model's root system. The first `1` indicates that the enabled subsystem is the second block on the root system's sorted list (zero-based indexing). The `1` in curly 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 software 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 software or Simulink Accelerator for models with large parameter sets. This improvement involves writing out parameter references instead of the entire parameter data into the Real-Time Workshop file for parameters whose size exceeds 10 elements. The parameter values for such references are retrieved directly from Simulink software during the code generation process.

- Decreased memory usage during various phases of code generation process in Real-Time Workshop software or 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 Software

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 software.

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 software. 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 the MATLAB product 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.

247

Version 5.0 (R13) Simulink Software

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:

- "Block Enhancements" on page 248
- "Simulation Enhancements" on page 253
- "Modeling Enhancements" on page 254
- "Platform Limitations for HP and IBM" on page 257

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" on page 249
- "Lookup Table Editor" on page 250
- "Model Verification Block Library" on page 250
- "Signal Builder Block" on page 250

- "DocBlock" on page 251

- "Rate Transition Block" on page 251

- "Block Library Reorganization" on page 251

- "Model Linearization Blocks" on page 251

- "Data Store Read/Write Block Navigation" on page 251

- "Enhanced S-Function Builder" on page 251

- "Miscellaneous Block Enhancements" on page 252

Fixed-Point Block Library

Simulink software 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 software 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 software 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 software.

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 software 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 software 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 Timed-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" on page 253
- "Algebraic Loop Highlighting" on page 253
- "Conditional Execution Behavior" on page 253
- "Reorganized Simulation Diagnostics" on page 254
- "Enhanced Diagnostic Viewer" on page 254

Invalid Loop Highlighting

Simulink software 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 software 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 software executed all blocks required to compute all inputs to each switch at each time step. In this release, Simulink software, by default, executes only the blocks required to compute the control input and

253

the data input selected by the control input at each time step. Similarly, code generated from the model by Real-Time Workshop software 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 systems

- 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" on page 255
- "Production Hardware Characteristics" on page 256
- "Including Symbols and Greek Letters in Block Diagrams" on page 256
- "True Color Support" on page 256
- "Print Details" on page 256
- "Boolean Logic Signals" on page 256
- "Model Discretizer" on page 256

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 "Working with 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 software 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 "Implement logic signals as Boolean data (vs. double)" 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 software, 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 software 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 Software

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" on page 258
- "Modeling Enhancements" on page 260
- "Simulink Debugger" on page 263
- "Block Library" on page 264
- "Triggered Subsystems" on page 266
- "Running Simulink 4.1 Models in Simulink 4.0 Software" on page 267
- "Direct Feedthrough Compensation Deprecated" on page 268
- "Improved Invalid Model Configuration Diagnostics" on page 268
- "Bug Fixes" on page 269

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 the Simulink documentation).

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 the Simulink documentation).

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 software 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 software connects as many ports on the two blocks as possible (see "Connecting Blocks").

Autorouting Signal Lines

Simulink software 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 260).

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 the Simulink documentation 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 the Simulink documentation 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 software 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 software 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 software 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 software 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 "Implement logic signals as Boolean data (vs. double)" in the Simulink documentation 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 software.

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 software 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 software did not issue a warning in some cases where this may occur.

Flagging Function-Call Subsystem Cycles

In previous versions, Simulink software allowed you to build models containing function-call-cycles, i.e., function-call subsystems that directly or indirectly call themselves.
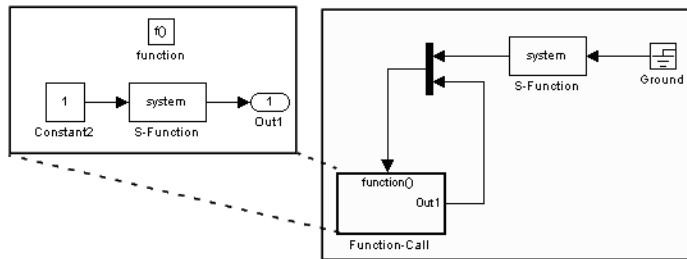
Such models cannot be correctly simulated. Accordingly, Simulink software now displays an error message when you attempt to run or update a diagram containing function-call cycles.

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 software 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 software 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 software. Consequently, models that contain invalid S-functions may produce different answers in this version of Simulink software.

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 Software

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 software 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 software 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 software 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 268).

- S-functions are now sorted like built-in blocks (see "S-Functions Sorted Like Built-In Blocks" on page 266).

- Simulink software no longer inserts implicit latches in triggered subsystems that directly or indirectly trigger themselves (see "Self-Triggering Subsystems Are No Longer Allowed" on page 267, 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 software 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 software (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 software.

Variable sample time S-functions

Simulink software 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® systems 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 Software

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" on page 271

- "Modeling Enhancements" on page 274

- "Simulink Debugger" on page 276

- "Block Library" on page 276

- "SB2SL" on page 279

- "Port Name Property" on page 280

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 "Simulink Preferences Window" in the Simulink documentation).

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 Reference Blocks" in the Simulink documentation).

Note Simulink software 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 the Simulink documentation, 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 software now provides two modes for opening subsystems. In multiwindow mode, Simulink software opens each subsystem in a new window. In single-window mode, Simulink software 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 Simulinkdocumentation.

Modeling Enhancements

Hierarchical Variable Scoping

This release extends the ability of Simulink software to resolve references to variables in masked subsystems. Previously Simulink software could resolve references only to variables in a block's local workspace. With this release, Simulink software 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 software.

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 software also supports scalar (dimensionless) signals and vector signals (one-dimensional arrays of signals). Simulink software 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 software 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 the Simulink documentation 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 the Simulink documentation).

Block Execution Order

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

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 Simulink documentation.

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 software now supports S-functions coded in ADA. See *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 software 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 software 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 software does not allow you to specify them as atomic or virtual.

SB2SL

SB2SL Extends Code Generation Support

SB2SL, which is included as part of the Simulink product, 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 Real-Time Workshop software. If you use Real-Time Workshop software version 4.0 to generate code for models you have converted from SystemBuild to Simulink software (using SB2SL), then code is generated for most translated blocks in the model.

The blocks that do *not* support code generation through Real-Time Workshop software version 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 Software

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

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version V7.5 (R2010a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Trigger Port Enhancements" on page 8 • "Custom Floating-Point Types No Longer Supported" on page 11 • "Models with No States Now Return Empty Variables" on page 12 • "To File Block Enhancements" on page 13 • "Enhanced Support for Proper Use of Bus Signals" on page 14 • "New Square Root Block" on page 17 • "Multiport Switch Block Allows Explicit Specification of Data Port Indices " on page 20 • "Data Type Duplicate Block Enhancement" on page 24 • "Lookup Table and Lookup Table (2-D) Blocks To Be Deprecated in a Future Release" on page 25 • "Elementary Math Block Now Obsolete" on page 29 • "DocBlock Block RTF File Compression" on page 29

Version (Release)	New Features and Changes with Version Compatibility Impact
	• "Simulink Extras PID Controller Blocks Deprecated" on page 29
	• "Model Explorer Column Views" on page 30
	• "Model Explorer Display of Masked Subsystems and Linked Library Subsystems" on page 31
	• "Legacy Code Tool Enhanced to Support Enumerated Data Types and Structured Tunable Parameters" on page 35
V7.4 (R2009b)	See the **Compatibility Considerations** subheading for each of these new features or changes:
	• "Improved Accuracy of Variable-Step Discrete Solver" on page 38
	• "New Compilation Report for Embedded MATLAB Function Blocks" on page 43
	• "Integer Arithmetic Applied to Sample Hit Computations" on page 38
	• "Simulation Restart in R2009b" on page 46
	• "Data Class Infrastructure Partially Deprecated" on page 45
	• "Discrete Transfer Fcn Block Has Performance, Data Type, Dimension, and Complexity Enhancements" on page 49
	• "Direct Lookup Table (n-D) Block Enhancements" on page 54
	• "Trigger Port Enhancements" on page 8

Version (Release)	New Features and Changes with Version Compatibility Impact
V7.3 (R2009a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Lookup Table (n-D) and Interpolation Using Prelookup Blocks Perform Efficient Fixed-Point Interpolations" on page 66 • "New Rounding Modes Added to Multiple Blocks" on page 67 • "Discrete Filter Block Performance, Data Type, Dimension, and Complexity Enhancements" on page 70 • "Dot Product Block Converted from S-Function to Core Block" on page 72 • "Removal of Lookup Table Designer from the Lookup Table Editor" on page 77 • "Signal Can Resolve to at Most One Signal Object" on page 63
V7.2 (R2008b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Conditionally Executed Subsystem Initial Conditions" on page 80 • "Data Type Override Now Works Consistently on Outputs" on page 85 • "Improperly-Scaled Fixed-Point Relational Operators Now Match MATLAB Results" on page 86 • "One Parameter Controls Accelerator Mode Build Verbosity" on page 83

Version (Release)	New Features and Changes with Version Compatibility Impact
	• "Signal Logging and Test Points Are Controlled Independently" on page 91 • "Signal Logging Consistently Retains Duplicate Signal Regions" on page 92 • "Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors" on page 88 • "Mapping of Target Object Properties to Parameters in the Configuration Parameters Dialog Box" on page 111 • "Legacy Code Tool Enhancement" on page 121
V7.1 (R2008a)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Specify Scaling Explicitly for Fixed-Point Data" on page 125 • "Array Format Cannot Be Used to Export Multiple Matrix Signals" on page 126 • "Changing Nontunable Values Does Not Affect the Current Simulation" on page 127 • "Detection of Illegal Rate Transitions" on page 127 • "Explicit Scaling Required for Fixed-Point Data" on page 127 • "Rate Transition Blocks Needed on Virtual Buses" on page 131 • "Sample Times for Virtual Blocks" on page 132 • "New Discrete FIR Filter Block Replaces Weighted Moving Average Block" on page 133

Version (Release)	New Features and Changes with Version Compatibility Impact
	• "Solver Controls" on page 136
	• "S-Functions" on page 138
V7.0 (R2007b)	See the **Compatibility Considerations** subheading for each of these new features or changes:
	• "Simulink® Accelerator" on page 139
	• "Mask Editor Now Requires Java" on page 141
	• "Support for Algorithms That Span Multiple M-Files" on page 142
	• "New Break Link Options for save_system Command" on page 149
	• "Simulink Software Checks Data Type of the Initial Condition Signal of the Integrator Block" on page 149
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 158
	• "Change to Simulink.ModelAdvisor.getModelAdvisor Method" on page 161
	• "Legacy Code Tool Enhancements" on page 163
	• "Using & and	Operators in Embedded MATLAB Function Blocks" on page 168
	• "Calling get Function from Embedded MATLAB Function Blocks" on page 169

Version (Release)	New Features and Changes with Version Compatibility Impact
	• "Default for Signal Resolution Parameter Has Changed" on page 171
	• "Port Parameter Evaluation Has Changed" on page 174
	• "Referencing Configuration Sets" on page 172
	• "Change to PaperPositionMode Parameter" on page 176
	• "Change in Version 6.5 (R2006b) Introduced Incompatibility" on page 177
V6.5 (R2006b)	See the **Compatibility Considerations** subheading for each of these new features or changes: • "Enhanced Lookup Table Blocks" on page 179 • "Parameter Objects Can Now Be Used to Specify Model Configuration Parameters" on page 183 • "New Requirement for Calling MATLAB Functions from Embedded MATLAB Function Blocks" on page 188 • "Type and Size Mismatch of Values Returned from MATLAB Functions Generates Error" on page 189 • "Changes to Integrator Block's Level Reset Options" on page 185 • "Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error" on page 184 • "Embedded MATLAB Function Blocks Cannot Output Character Data" on page 190
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 193 • "Concatenate Block" on page 194 • "Built-in Block's Initial Appearance Reflects Parameter Settings" on page 195 • "Setting FIMATH Cast Before Sum to False No Longer Supported in Embedded MATLAB MATLAB Function Blocks" on page 201 • "Type Mismatch of Scalar Output Data in Embedded MATLAB Function Blocks Generates Error" on page 202 • "Implicit Parameter Type Conversions No Longer Supported in Embedded MATLAB Function Blocks" on page 203
V6.3 (R14SP3)	See the **Compatibility Considerations** and/or **Limitations** subheading for each of these changes: • "Model Referencing" on page 205 • "MEX-Files on Windows Systems" on page 213 • "Fixed-Point Functions No Longer Supported for Use in Signal Objects" on page 213 • "Parameter Object Expressions No Longer Supported in Dialog Boxes" on page 213 • "MEX-File Extension Changed" on page 214

Version (Release)	New Features and Changes with Version Compatibility Impact
V6.2 (R14SP2)	See the **Compatibility Considerations** and/or **Limitations** subheadings for each of these changes: • "Rootlevel Input Ports" on page 216
V6.1 (R14SP1)	See the **Compatibility Considerations** for this change: • "Model Load Warnings" on page 219
V6.0 (R14)	See the **Compatibility Considerations** and **Limitations** subheadings for each of these changes: • "Model Referencing" on page 221 • "MATLAB Data Type Conversions" on page 230 • "Signal Object Resolution Changes" on page 230 • "Loading Models Containing Non-ASCII Characters" on page 231 • "Change in Sample Time Behavior of Unary Minus Block" on page 232 • "Initial Output of Conditionally Executed Subsystems" on page 232 • "Execution Context Default Changes" on page 232 • "Internal Signal Structures Revamped" on page 228
V5.0.1 (R13.0.1)	See the **Compatibility Considerations** subheading for this change: • "Tunable Parameters for Unified Fixed-Point Blocks" on page 245

Version (Release)	New Features and Changes with Version Compatibility Impact
V5.0 (R13)	See the **Compatibility Considerations** subheadings for each of these changes: • "Enhanced Diagnostic Viewer" on page 254 • "Enhanced Mask Editor" on page 255 • "Model Discretizer" on page 256
V4.1 (R12+)	See the **Compatibility Considerations** subheadings for each of these changes: • "Simulink Block Library Reorganization" on page 265 • "S-Functions Sorted Like Built-In Blocks" on page 266 • "Added Latched Triggered Subsystems" on page 266 • "Self-Triggering Subsystems Are No Longer Allowed" on page 267 • "Running Simulink 4.1 Models in Simulink 4.0 Software" on page 267 • "Direct Feedthrough Compensation Deprecated" on page 268 • "Improved Invalid Model Configuration Diagnostics" on page 268
V4.0 (R12)	See the **Compatibility Considerations** for this change: • "Port Name Property" on page 280