SIMULINK®

Dynamic System Simulation for MATLAB®

Modeling

Simulation

Implementation

Simulink 2.1 New Features

How to Contact The MathWorks:

508-647-7000	Phone
508-647-7001	Fax
The MathWorks, Inc. 24 Prime Park Way Natick, MA 01760-1500	Mail
`http://www.mathworks.com`	Web
`ftp.mathworks.com`	Anonymous FTP server
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`subscribe@mathworks.com`	Subscribing user registration
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

Simulink 2.1 New Features

Printing History: May 1997 First printing

1 Simulink 2.1 New Features

2 Simulink 2.0 New Features

Introduction

This document provides a brief description of the significant new features included in Simulink® 2.0 and Simulink 2.1. The manual contains two chapters:

- Chapter 1 describes features new to Simulink 2.1. If you are upgrading from Simulink 2.0 to Simulink 2.1, read this chapter.
- Chapter 2 describes features new to Simulink 2.0. If you are upgrading from Simulink 1.3 to Simulink 2.1, read both chapters.

The organization of sections in each chapter and the chapter titles reflect the order and titles of the chapters in *Using Simulink*.

Simulink 2.1 New Features

Creating a Model

Undo Command

You can cancel the effects of up to 101 separate operations by choosing **Undo** from the **Edit** menu. You can undo these operations:

- Adding or deleting a block.
- Adding or deleting a line.
- Adding or deleting a model annotation.
- Editing a block name.

You can reverse the effects of the **Undo** command by choosing **Redo** from the **Edit** menu.

Libraries

Simulink 2.1 provides support for user libraries. This feature enables users to copy blocks into their models from external libraries and automatically update the copied blocks when the source library changes.

In previous releases, Simulink supported this feature only for built-in blocks (built-in blocks are blocks provided in Simulink block libraries that are not masked blocks). When you copied a Simulink built-in block into your model and upgraded Simulink to a new version, your models would be updated automatically to include the new built-in blocks.

This feature allows users who develop their own block libraries, or who use those provided by others (such as blocksets), to ensure that their models automatically include the most recent versions of these blocks.

Terminology

It is important to understand the terminology used with this feature.

Library – A collection of library blocks. A library must be explicitly created using **New Library** from the **File** menu.

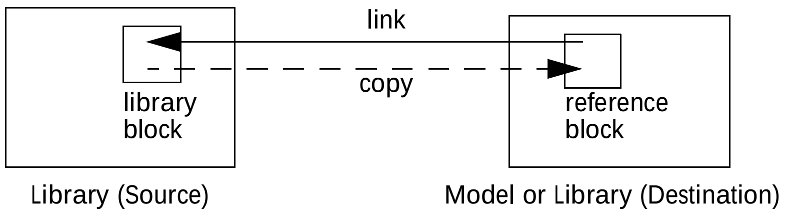
Library block – A block in a library.

Reference block – A copy of a library block. While the reference block remains linked to the library block, the reference block is updated whenever the library block changes.

Link – The connection between the reference block and its library block that allows Simulink to update the reference block when the library block changes.

Copy – The operation that creates a reference block from either a library block or another reference block.

This figure illustrates the terminology used in this feature:



Creating a Library

To create a library, select **Library** from the **New** submenu of the **File** menu. Simulink displays a new window, labeled **Library: untitled**. If an untitled window already appears, a sequence number is appended.

If, in previous versions of Simulink, you provided customized blocks using a model, you can copy the contents of the model into the library.

The library must be named (saved) before you can copy blocks from it.

You can create a library from the command line using this command:

```
new_system('newlib', 'Library')
```

This command creates a new library named `'newlib'`. To display the library, use the `open_system` command. These commands are described in Chapter 11 of *Using Simulink*.

Modifying a Library

When you open a library, it is automatically locked and you cannot modify its contents. To unlock the library, select **Unlock Library** from the **Edit** menu. Closing the library window locks the library.

Copying a Library Block into a Model

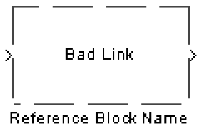
When you copy a library block into a model or another library, Simulink creates a link to the library block in the library. The reference block is a copy of the library block. You can modify block parameters but you cannot mask the block or, if it is masked, edit the mask. Also, you cannot set callback parameters for a reference block. If you look under the mask of a reference block, Simulink displays the underlying system for the library block.

The library and reference blocks are linked *by name*; that is, the reference block is linked to the specific block and library whose names are in effect at the time the copy is made.

If Simulink is unable to find either the library block or the source library on your MATLAB® path when it attempts to update the reference block, the link becomes *unresolved*. Simulink issues an error message and displays these blocks using red dashed lines. The error message is:

```
Failed to find block "source-block-name"
in library "source-library-name"
referenced by block
"reference-block-path".
```

The unresolved reference block is displayed like this (colored red):



To fix a bad link, you must either:

- Delete the unlinked reference block and copy the library block back into your model, or
- Add the directory that contains the required library to the MATLAB path and select **Update Diagram** from the **Edit** menu, or
- Double-click on the reference block and, on the dialog box that appears, correct the pathname and click on **Apply** or **Close**.

All blocks have a new parameter called `LinkStatus` that indicates whether the block is a reference block. Possible values for the parameter are:

- `'none'` indicates that the block is not a reference block.
- `'resolved'` indicates that the block is a reference block and that the link is resolved.
- `'unresolved'` indicates that the block is a reference block but that the link is unresolved.

Updating a Linked Block

Simulink updates reference blocks in a model or library at these times:

- When the model or library is loaded.
- When you select **Update Diagram** from the **Edit** menu or run the simulation in these circumstances:
  - When the library containing the library blocks is unlocked
  - When the model contains a reference block with an unresolved link
- When you query the `LinkStatus` parameter of a block using the `set_param` command (see the next section)
- When you use the `find_system` command

Breaking a Link to a Library Block

You can break the link between a reference block and its library block to cause the reference block to become an unlinked copy of the library block. When you break the link, changes to the library block no longer affect the block. Breaking links to library blocks enables you to transport a model as a stand-alone model, without the libraries.

To break the link between a reference block and its library block, select the block, then choose **Break Library Link** from the **Edit** menu. You can also break the link between a reference block and its library block from the command line by changing the value of the `LinkStatus` parameter to `'none'` using this command:

```
set_param('refblock', 'LinkStatus', 'none')
```

You can save a system and break all links between reference blocks and library blocks using this command:

```
save_system('sys', 'newname', 'BreakLinks')
```

Finding the Library Block for a Reference Block

To find the source library and block linked to a reference block, select the reference block, then choose **Go To Library Link** from the **Edit** menu. If the library is open, Simulink selects the library block (displaying selection handles on the block) and makes the source library the active window. If the library is not open, Simulink opens it and selects the library block.

Getting Information About Library Blocks in a System

Use the `libinfo` command to get information about reference blocks in a system. The syntax of the command is:

```
libdata = libinfo(sys)
```

where `sys` is the name of the system. The command returns a structure of size n-by-1, where n is the number of library blocks in `sys`. Each element of the structure has four fields:

- `Block`, the block path
- `Library`, the library name
- `ReferenceBlock`, the reference block path
- `LinkStatus`, the link status, either `'resolved'` or `'unresolved'`

New Block Callback Parameters

Simulink 2.1 provides these new block callback parameters:

- `InitFcn` executes before the block diagram is compiled and before the block parameters are evaluated.
- `StartFcn` executes after the block diagram is compiled and before the simulation starts.
- `StopFcn` executes at any termination of a simulation.
- `UndoDelete` executes when a block delete is undone.

Displaying Line Widths

You can display the widths of all lines in a model by turning on **Line Widths** from the **Format** menu. Simulink indicates the width of each signal at the block that originates the signal and the block that receives it.

When you start a simulation or update the diagram and Simulink detects a mismatch of input and output ports, it displays an error message and shows line widths in the model.

Changing the Font of a Signal Label

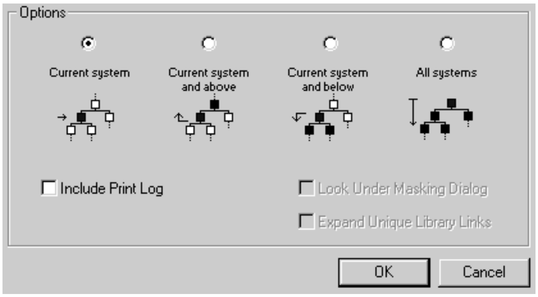
You can change the font of a signal label by selecting the signal, choosing **Font** from the **Format** menu, then selecting a font from the **Set Font** dialog box.

Print Dialog Box

This release provides a **Print** dialog box that enables you to selectively print systems within your model. Using the dialog box, you can:

- Print the current system only
- Print the current system and all systems above it in the model hierarchy
- Print the current system and all systems below it in the model hierarchy, with the option of looking into the contents of masked and library blocks
- Print all systems in the model, with the option of looking into the contents of masked and library blocks

The portion of the **Print** dialog box that supports selective printing is similar on supported platforms. This figure shows how it looks on a Microsoft Windows system. In this figure, only the current system is to be printed:



Options

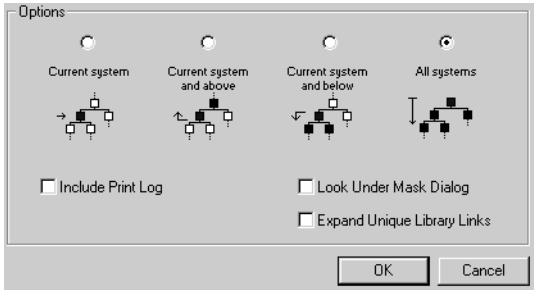
[✓ Current system] Current system and above Current system and below All systems

☐ Include Print Log

☐ Look Under Masking Dialog

☐ Expand Unique Library Links

When you select either the **Current system and below** or **All systems** option, two check boxes become enabled. In this figure, **All systems** is selected.



Options

Current system Current system and above Current system and below [✓ All systems]

☐ Include Print Log

☐ Look Under Mask Dialog

☐ Expand Unique Library Links

The **Look Under Mask Dialog** check box prints the contents of masked subsystems when encountered at or below the level of the current block. (When printing all systems, the top-level system is considered the current block so Simulink looks under any masked blocks encountered.)

The **Expand Unique Library Links** check box prints the contents of library blocks when those blocks are systems. Only one copy is printed regardless of how many copies of the block are contained in the model. For more information about libraries, see “Libraries” on page 1-2.

You can choose to print the print log, which lists the blocks and systems printed. To print the print log, select the **Include Print Log** check box.

Running a Simulation

Changing Parameters During a Simulation

A useful feature of Simulink is the ability to change block parameter values during a simulation. Simulink 2.1 enables you to change block parameters in the workspace and update the block diagram with those values. To do this, after making the changes in the command window, make the model window the active window, then choose **Update Diagram** from the **Edit** menu.

Running a Simulation Using Commands

You can use the `set_param` command to start, stop, pause, or continue a simulation, or update a block diagram. Similarly, you can use the `get_param` command to check the status of a simulation.

The format of the `set_param` command for this use is:

```
set_param('sys', 'SimulationCommand', 'cmd')
```

where `'sys'` is the name of the system and `'cmd'` is `'start'`, `'stop'`, `'pause'`, `'continue'`, or `'update'`.

The format of the `get_param` command for this use is:

```
get_param('sys', 'SimulationStatus')
```

Simulink returns `'stopped'`, `'initializing'`, `'running'`, `'paused'`, `'terminating'`, and `'external'` (used with Real-Time Workshop®).

The simget Command

If you use the `simget` command to get the value of a model parameter whose value is set by a variable, the variable must exist in the workspace. If the variable is undefined, Simulink returns an error message.

For example, these commands set the value of the Refine model parameter for the vdp system to var, then use simget to determine the value of the parameter. If var is not defined, Simulink returns an error message.

```
vdp;
set_param('vdp','Refine','var');
simget('vdp','Refine')
??? Error using ==> simget
All variables used in the Parameters dialog must be defined in the
workspace.
The following parameters have variables which are undefined:
    Refine
```

Masking

Simulink 2.1 provides these enhancements to the masking feature:

Masking Subsystems

The **Create Mask** menu item has been changed to **Mask Subsystem**. You can now mask only Subsystem blocks. Masked blocks that are not subsystems will continue to work properly but in the future, Simulink will support masked subsystems only.

Accessing User-Written Help Documentation

You can include user-written documentation for a masked block's help. You can specify any of the following for the masked block help text:

- URL specification (a string starting with `http:`, `www`, `file:`, `ftp:`, or `mailto:`)
- `web` command (launches a browser)
- `eval` command (evaluates a MATLAB string)
- Static text displayed in the web browser (the option previously supported)

Simulink examines the first line of the masked block help text. If it detects a URL specification, `web` command, or `eval` command, it accesses the block help as directed; otherwise, the full contents of the masked block help text are displayed in the browser.

These examples illustrate several acceptable commands:

```
web([docroot '/My Blockset Doc/' get_param(gcb,'MaskType') '.html'])
eval('!Word My_Spec.doc')
http://www.mathworks.com
file:///c:/mydir/helpdoc.html
www.mathworks.com
```

Using NaN and inf in Plot Commands

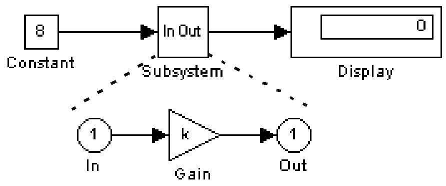
Plot commands used to generate the block icon can include `NaN` and `inf`. When `NaN`s or `inf`s are encountered, Simulink stops drawing, then begins redrawing at the next numbers that are not `NaN` or `inf`.

Rotating a Masked Block

Rotating a masked block causes Simulink to execute the block's initialization commands. Also, if you turn on block rotation for a masked block having a graphical icon, the icon rotation is consistent with the block port rotation.

Accessing Masked Block Parameters

You can now access parameters for masked blocks the same way you access parameters for blocks that are not masked. For example, this system, called mysys, has a simple masked subsystem. The masked subsystem's dialog box includes a prompt for the Gain block's **Gain** parameter.



On the mask dialog box for the subsystem, the variable k is associated with the **Gain** parameter. This command sets the **Gain** parameter:

```
set_param('mysys/Subsystem', 'k', '10')
```

When this model is simulated, the Display block shows the value 80.

Looking Under Mask from Command Line

You can look under a block mask or OpenFcn callback from the command line using the open_system command. Specify 'force' as the second argument. For example, this command displays the underlying blocks that make up the Ramp block in the mymodel system:

```
open_system('mymodel/Ramp', 'force')
```

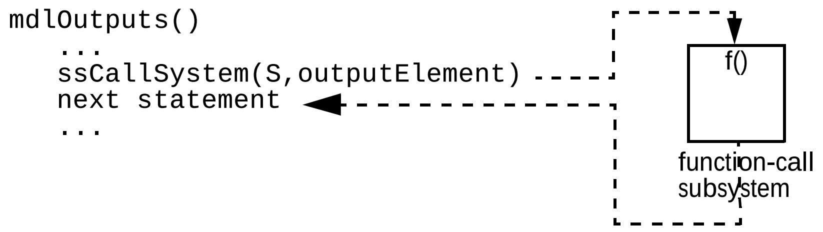
Conditionally Executed Subsystems

Simulink 2.1 enables you to create a triggered subsystem whose execution is determined by logic internal to an S-function, instead of by the value of a signal. The subsystems are called *function-call subsystems*. To use them, you must use a new macro in the S-function and change the trigger type of the Trigger block.

To implement a function-call subsystem:

- In the Trigger block, select the **function-call** value for the **Trigger type** parameter.
- In the S-function, use the `ssCallSystem` macro to call the triggered subsystem. For more details, see "Using Function-Call Subsystems" on page 1-14.
- In the model, connect the S-Function block output directly to the trigger port.

Function-call subsystems are not executed directly by Simulink. The S-function determines when to execute the subsystem. When the subsystem completes execution, control returns to the S-function. This figure illustrates the interaction between a function-call subsystem and an S-function:



```
mdlOutputs()
    ...
    ssCallSystem(S,outputElement)
    next statement
    ...
```

Function-call subsystems are generally used by Stateflow™ blocks. For more information on their use, see the Stateflow documentation.

S-Functions

Simulink 2.1 provides numerous enhancements to S-functions. All items below apply to C MEX S-functions unless otherwise noted.

Revised sfuntmpl Template

The S-function template has been revised for this release. The template, `sfuntmpl.c`, is located in `simulink/src`. For an amply commented version of the template, see `sfuntmpl.doc` in the same directory.

Using Function-Call Subsystems

Subsystem trigger ports now have a fourth option function-call selection. A subsystem that contains a trigger port configured as a function-call is referred to as a *function-call subsystem*. These subsystems can be connected only to S-functions configured to execute function-call subsystems. To configure an S-function to call a function-call subsystem:

1. Specify which elements are to execute the function-call system in `mdlInitializeSampleTimes`. For example:

   ```
   ssSetCallSystemOutput(S,0);  /* call on 1st element */
   ssSetCallSystemOutput(S,2);  /* call on 3rd element */
   ```

2. Execute the subsystem in the appropriate `mdlOutputs`, `mdlUpdates`, or `mdlDerivatives` routine. For example:

   ```
   static void mdlOutputs(...)
   {
       if (((int)u[0]) % 2 == 1) {
           ssCallSystem(S,0);
       } else {
           ssCallSystem(S,2);
       }
   ...
   }
   ```

`sfun_fcncall.c` illustrates an S-function configured to execute function-call subsystems.

Instantaneous Update of S-Function Inputs

You can now get instantaneous updates of S-function inputs by accessing the inputs through pointers. This is required for execution of function-call subsystems whose inputs feed back into your S-function. To configure your S-function to use input pointers, set, in `mdlInitializeSizes`:

```
ssSetOptions(S,SS_OPTION_USING_ssGetUPtrs)
```

Then, in all the routines where you access the input, include

```
UPtrsType uPtrs = ssGetUPtrs(S);
```

The `i`th input is then `*uPtrs[i]`.

Output and Work Vector Widths

For S-functions with dynamically sized inputs and outputs, you can now configure your output width to be a function of your input width and vice versa. To do this, define these routines:

```
int mdlGetInputPortWidth(int outputWidth) /* return input width */
int mdlGetOutputPortWidth(int inputWidth) /* return output width */
```

Also, you can now configure your work vector widths based on the input width, output width, or sample times. To do this, define a `mdlSetWorkWidths` routine.

`sfun_dynsize.c` demonstrates how to use the `mdlSetInputPortWidth`, `mdlSetOutputPortWidth`, and `mdlSetWorkWidths` optional methods to configure the input port width, output port width, and real work vector length based on the size of the signal driving the S-function.

Unconnected Dynamically Sized S-Functions

Dynamically sized S-functions can now determine when they aren't connected using the `ssGetInputConnected(S)` and `ssGetOutputConnected(S)` macros.

Nonsampled Zero Crossings for Continuous S-Functions

Continuous S-functions can now register nonsampled zero crossings. Nonsampled zero crossings are used to “hone-in” on state events (i.e., discontinuities in the first derivative) of some signal, usually a function of an input to your S-function. To register nonsampled zero crossings, set the number of nonsampled zero crossings in `mdlInitializeSizes` using:

```
ssSetNumNonsampledZCs(S, num)
```

Then, define the `mdlZeroCrossings` routine to return the nonsampled zero crossings. `sfun_zc.c` shows how to use nonsampled zero crossings.

New Mode Work Vector

There is support for a new work vector, referred to as the *mode vector*. Elements are integer values that are typically used with nonsampled zero crossings. Specify the number of mode vector elements in `mdlInitializeSizes` using `ssSetNumModes(S,num)`. You can then access the mode vector using `ssGetModeVector`.

Parameter Changes

S-functions can now be notified of parameter changes. To be notified of parameter changes, your S-function must register a `mdlCheckParameters` routine. This routine will be called any time after `mdlInitializeSizes` has been called.

Improved Error Handling

Error handling has been improved to work consistently across Real-Time Workshop and Simulink. To report an error, S-functions should include these statements:

```
ssSetErrorStatus(S, "Error string");
return;
```

It is important that `Error string` be persistent memory, not a stack variable.

`sfun_errhdl.c` shows how to perform error checking. The S-function checks to see that required parameters are of the correct format.

Exception Handling

Each time an S-function is invoked, Simulink performs overhead tasks associated with exception handling. If the S-function does not contain any routines that can generate exceptions, you can improve the performance of the simulation.

If you are not using routines that can throw an exception, set this option in `mdlInitializeSizes`:

```
ssSetOption(S, SS_OPTION_EXCEPTION_FREE_CODE);
```

Do not specify the `SS_OPTION_EXCEPTION_FREE_CODE` option if the S-function contains any routines that can throw an exception:

- All routines that start with `mex` can throw an exception.
- Routines that start with `mx` can throw an exception, except routines that get a pointer or determine the size of parameters, such as `mxGetPr`, `mxGetData`, `mxGetNumberOfDimensions`, `mxGetM`, `mxGetN`, and `mxGetNumberOfElements`.

If the S-function must allocate memory, avoid using `mxCalloc`. Instead, use the `stdlib.h` `calloc` routine directly and perform your own error handling, then free the memory in `mdlTerminate`.

Normal or Real-Time Workshop Simulation

S-functions can now determine if they are running in a normal simulation or as part of Real-Time Workshop code generation or as part of external mode. This is done using `ssGetSimMode(S)`.

Removing Ports When No Inputs and/or Outputs

Any C MEX, Fortran MEX, or M-file S-function that indicates it has no input and/or outputs will have the corresponding port(s) removed from the S-Function block icon. The port is removed when the simulation starts or when you choose **Update Diagram** from the **Edit** menu.

Sample Times and Input/Output Port Widths

Sample times can now be a function of the input/output port widths. In `mdlInitializeSizes`, you can specify that sample times are a function of `ssGetNumInputs` and `ssGetNumOutputs`.

Additional Macros in mdlInitializeSizes

The `ssGetPath`, `ssGetModelName`, and `ssSetStatus` macros work in `mdlInitializeSizes`.

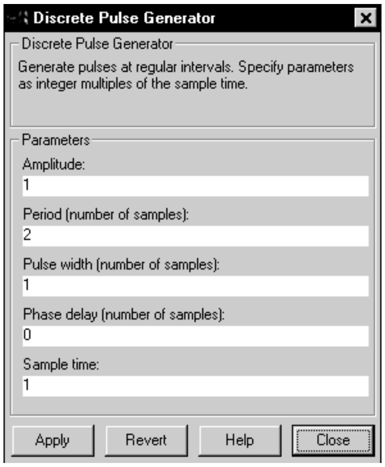
Blocks

This release makes numerous changes to the block libraries. The new and revised blocks are described in more detail below.

- The new Discrete Pulse Generator block (in the Sources library) is a discrete version of the Pulse Generator block.
- The Signal Generator block no longer permits the use of the Random Noise signal and produces a warning message when the Random Noise signal is selected. The Signal Generator block has a continuous sample time, so using this block to generate random noise in a purely discrete system is inefficient. Consider using either the Random Number block or the new Uniform Random Number block, described below, and set the sample time.
- The Sine Wave block now defaults to a continuous sample time, although you can set the sample time. The signal displayed on the block icon indicates whether the sample time is continuous or discrete.
- The Random Number block (in the Sources library) now provides greater control of its signal. You can specify the mean and standard deviation for the random numbers. Also, the block now has a continuous sample time.
- The new Uniform Random Number block (in the Sources library) enables you to add uniformly distributed random noise to your system. You can specify the interval of the random numbers. By default, the block has a continuous sample time.
- The Scope block now displays up to 30 lines. The Scope cycles through the six colors in the order described in *Using Simulink*. Also, floating Scopes can auto-scale their input only *during* a simulation.
- The Elementary Math block is replaced by three blocks: the Math Function block, the Rounding Function block, and the Trigonometric Function block. Each block provides related functions; this release also provides additional functions.
- The new Manual Switch block (in the Nonlinear library) enables you to switch input by double-clicking on the block icon.
- The Product block (in the Nonlinear library) now multiplies or divides its inputs or input elements.
- The Rate Limiter block limits the rate at which the block output reaches the block input. The block works the same as it did in Simulink 1.3.

Discrete Pulse Generator Block

The new Discrete Pulse Generator block generates pulses at regular intervals. The **Period**, **Pulse width**, and **Phase delay** parameters are all expressed in terms of the number of samples.



Discrete Pulse Generator

Generate pulses at regular intervals. Specify parameters as integer multiples of the sample time.

Parameters

Amplitude:
1

Period (number of samples):
2

Pulse width (number of samples):
1

Phase delay (number of samples):
0

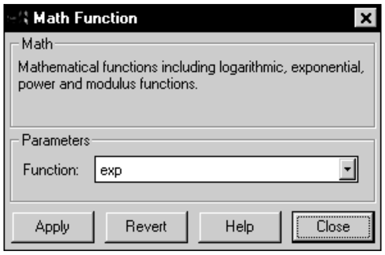
Sample time:
1

Manual Switch Block

The Manual Switch block enables you to switch between two inputs by double-clicking on the block during a simulation. The block has no dialog box.

Math Function Block

The Math Function block provides these general mathematical functions: `exp`, `log`, `10`^{u}, `log10`, `square`, `sqrt`, `pow`, `reciprocal`, `hypot`, `rem`, and `mod`.



Math

Mathematical functions including logarithmic, exponential, power and modulus functions.

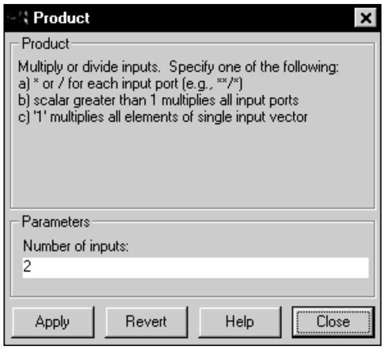
Parameters

Function: exp

You select the function from the **Function** list. The block icon displays the name or a representation of the selected function. Simulink automatically draws the appropriate number of input ports (one input port for all functions except pow, hypot, rem, and mod, which have two ports). For more information about these functions, see the MATLAB documentation or the online MATLAB Function Reference.

Product Block

The Product block now enables you to multiply or divide block inputs or elements of a single vector input.



Product

Multiply or divide inputs. Specify one of the following:
a) * or / for each input port (e.g., **/*)
b) scalar greater than 1 multiplies all input ports
c) '1' multiplies all elements of single input vector

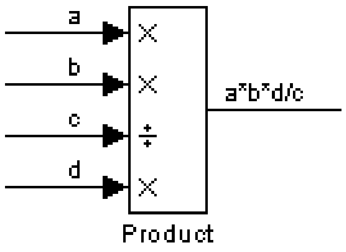
Parameters

Number of inputs:
2

You can specify the **Number of inputs** parameter in three ways:

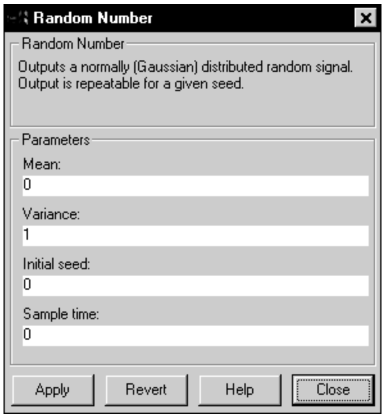
- An integer value causes the block to generate output that is the product of the inputs.
- A value of 1 or * causes the block to generate output that is the product of all elements of the input vector. A value of / causes the block to output the reciprocal of the product of all elements of the input vector.
- A combination of * and / symbols causes the block to generate output that is the product of the inputs, where each input associated with the backslash is converted to its reciprocal first. See the example below for an illustration.

This example shows a simple model that illustrates the use of * and / symbols. The value of the **Number of inputs** parameter is **/*:



Random Number Block

The Random Number block, which generates normally distributed random numbers, now enables you to specify the mean and standard deviation of the random numbers it generates. You can also specify a sample time.



Random Number

Outputs a normally (Gaussian) distributed random signal.
Output is repeatable for a given seed.

Parameters

Mean:
0

Variance:
1

Initial seed:
0

Sample time:
0

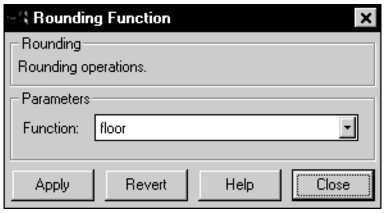
Rate Limiter Block

The Rate Limiter block has been modified to limit the rate of change of the block output. At each time step, Simulink computes the difference between the current block input and the previous block output. The block output is then limited by block parameters. The rate is expressed by this equation:

$$rate = \frac{u(i) - y(i-1)}{t(i) - t(i-1)}$$

Rounding Function Block

The Rounding Function block provides these rounding functions: `floor`, `ceil`, `round`, and `fix`.



Rounding
Rounding operations.
Parameters
Function: floor

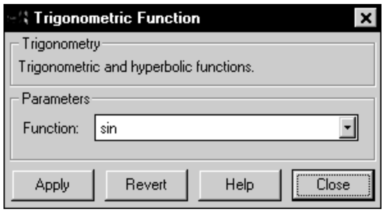
You select the function from the **Function** list. The block icon displays the name of the selected function.

Trigger Block

The Trigger block now provides a fourth **Trigger type** parameter choice, **function-call**. Choosing this trigger type causes the triggered subsystem to act as a function that executes immediately after being called by an S-function. This feature is used primarily with Stateflow and Real-Time Workshop.

Trigonometric Function Block

The Trigonometric Function block provides these trigonometric functions: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `sinh`, `cosh`, and `tanh`.

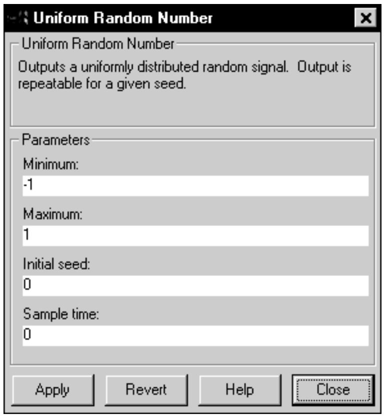


Trigonometry
Trigonometric and hyperbolic functions.
Parameters
Function: sin

You select the function from the **Function** list. The block icon displays the name of the selected function.

Uniform Random Number Block

The Uniform Random Number block enables you to add uniformly distributed random noise to your system. You can specify the range of the random numbers.


Uniform Random Number

Uniform Random Number

Outputs a uniformly distributed random signal. Output is repeatable for a given seed.

Parameters

Minimum:
-1

Maximum:
1

Initial seed:
0

Sample time:
0

Change to Extras Library

The Flip-Flop sublibrary has been moved to the Blocksets & Toolboxes library in the Simulink Extras sublibrary.

The Set-point PID Controller block has been removed from the Additional Linear sublibrary.

Additional Topics

Zero Crossings – A Clarification

The description in Chapter 10 of *Using Simulink* does not describe a situation where a zero crossing is not detected.

If the zero crossing occurs within a time step, such that the values at the beginning and end of the step do not indicate a zero crossing, the solver will step over the crossing without detecting it.

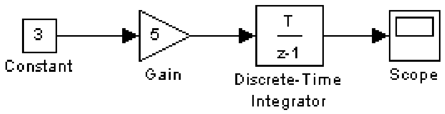
If you suspect this is happening, tighten the error tolerances to force the solver to take smaller steps. For more information about error tolerances, see Chapter 4 of *Using Simulink* or the online help for the **Simulation Parameters** dialog box.

Invariant Constants

Blocks either have explicitly defined sample times or inherit their sample times from blocks that feed them or are fed by them.

Simulink assigns Constant blocks a sample time of infinity, also referred to as a constant sample time. Other blocks have constant sample time if they receive their input from a Constant block and do not inherit the sample time of another block. This means that the output of these blocks does not change during the simulation unless the parameters are explicitly modified by the model user.

For example, in this model, both the Constant and Gain blocks have constant sample time:



Because Simulink supports the ability to change block parameters during a simulation, all blocks, even blocks having constant sample time, must generate their output at the model’s effective sample time.

Because of this feature, *all* blocks compute their output at each sample time hit, or, in the case of purely continuous systems, at every simulation step. For blocks having constant sample time whose parameters do not change during a simulation, evaluating these blocks during the simulation is inefficient and slows down the simulation.

Simulink 2.1 provides the new Invariant Constants feature that removes all blocks having constant sample times from the simulation “loop.” The effect of this feature is twofold: first, parameters for these blocks cannot be changed during a simulation; and second, simulation speed is improved. The speed improvement depends on model complexity, the number of blocks with constant sample time, and the effective sampling rate of the simulation.

You can apply the invariant constant feature to your model by entering this command in the MATLAB command window:

```
set_param('model_name', 'InvariantConstants', 'on')
```

You can turn off the feature by issuing the command again, assigning the parameter the value of `'off'`.

You can determine which blocks have constant sample time by turning on **Sample Time Colors** from the **Format** menu. Blocks having constant sample time are colored magenta.

Model Construction Commands

Vectorization of get_param Command

The get_param command now accepts a cell array of full path specifiers, enabling you to get the values of a parameter common to all objects specified in the cell array.

The format of the command is:

```
get_param( { objects }, 'ParameterName')
```

For example, these commands display the block types of all blocks in the vdp system:

```
>> blks = find_system(gcs, 'Type', 'block');
>> get_param(blks, 'BlockType')

ans =

    'Fcn'
    'Integrator'
    'Integrator'
    'SubSystem'
    'SubSystem'
    'Gain'
    'Mux'
    'Product'
    'Scope'
    'Sum'
    'Outport'
    'Outport'
```

Changes to find_system Command

The find_system command now enables you to search within a masked system (unless the mask consists only of an icon) or follow a link into a library. The syntax for each constraint is:

```
find_system('LookUnderMasks', 'on', ...)
find_system('FollowLinks', 'on', ...)
```

The default value for both constraints is `'off'`. If the default values for these constraints are in effect, the search does not extend into the masked systems or linked blocks.

New gcbh Command

The new `gcbh` command gets the handle of the current block. You can use the command to identify or address blocks that have no parent system. The command should be most useful to blockset authors.

Changes to slupdate Command

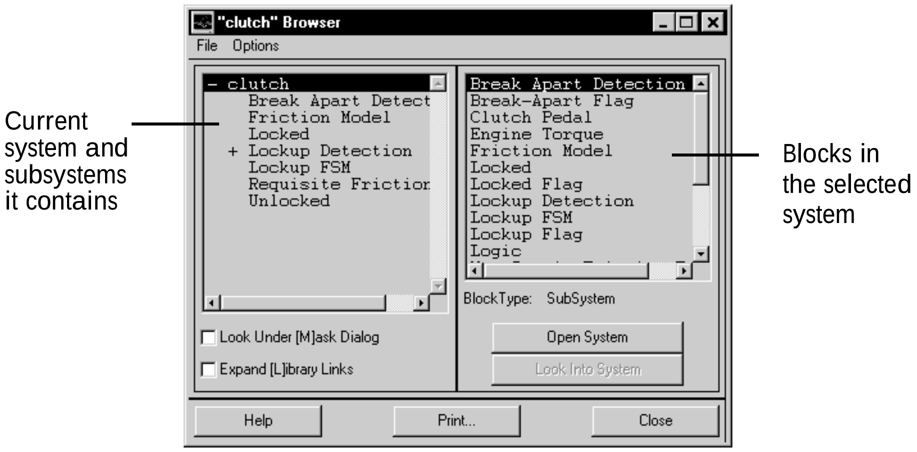
In this release, `slupdate` replaces the Elementary Math block with either the Math Function block, the Rounding Function block, or the Trigonometric Function block, depending on the selected function. It also fixes a bug in the mask for the Look-Up Table (2D) block.

The Model Browser

Simulink 2.1 provides a model browser that enables you to:

- Navigate a model hierarchically.
- Open systems in a model directly.
- Determine the blocks contained in a model.

To open the Browser, select **Show Browser** from the **File** menu. The Browser window appears, displaying information about the current model. This figure shows the Browser window displaying the contents of the `clutch` system:



```
- clutch
    Break Apart Detect
    Friction Model
    Locked
  + Lockup Detection
    Lockup FSM
    Requisite Friction
    Unlocked
```

```
Break Apart Detection
Break-Apart Flag
Clutch Pedal
Engine Torque
Friction Model
Locked
Locked Flag
Lockup Detection
Lockup FSM
Lockup Flag
Logic
```

BlockType: SubSystem

Contents of the Browser Window

The Browser window consists of:

- The systems list. The list on the left contains the current system and the subsystems it contains, with the current system selected.
- The blocks list. The list on the right contains the names of blocks in the selected system. Initially, this window displays blocks in the top-level system.
- The **File** menu, which contains the **Print**, **Close Model**, and **Close Browser** menu items.

- The **Options** menu, which contains these menu items: **Open System**, **Look Into System**, **Display Alphabetical/Hierarchical List**, **Expand All**, **Look Under Mask Dialog**, and **Expand Library Links**.
- **Options** check boxes and buttons: **Look Under [M]ask Dialog** and **Expand [L]ibrary Links** check boxes, and **Open System** and **Look Into System** buttons. By default, Simulink does not display contents of masked blocks and blocks that are library links. These check boxes enable you to override the default.
- The block type of the selected block.
- Dialog box buttons: **Help**, **Print**, and **Close**.

Interpreting List Contents

Simulink identifies masked blocks, reference blocks, blocks with defined `OpenFcn` parameters, and systems that contain subsystems using these symbols before a block or system name:

- A plus sign (+) before a system name in the systems list indicates that the system is expandable, which means that it has systems beneath it. Double-click on the system name to expand the list and display its contents in the blocks list. When a system is expanded, a minus sign (–) appears before its name.
- [M] indicates that the block is masked, having either a mask dialog box or a mask workspace. See Chapter 6 of *Using Simulink* for more information about masking.
- [L] indicates that the block is a reference block. For more information, see “Libraries” on page 1-2.
- [O] indicates that an open function (`OpenFcn`) callback is defined for the block. For more information about block callbacks, see Chapter 3 of *Using Simulink*.
- [S] indicates that the system is a Stateflow block.

Opening a System

You can open any block or system whose name appears in the Blocks list. To open a system:

1. In the systems list, select by single-clicking on the name of the parent system that contains the system you want to open. The parent system's contents appear in the blocks list.
2. Depending on whether the system is masked, linked to a library block, or has an open function callback, you open it as follows:

   If the system has no symbol to its left, double-click on its name or select its name and click on the **Open System** button.

   If the system has an [M] or [O] before its name, select the system name and click on the **Look Into System** button.

Looking into a Masked System or a Linked Block

By default, the Browser considers masked systems (identified by [M]) and linked blocks (identified by [L]) as blocks and not subsystems. If you click on **Open System** while a masked system or linked block is selected, the Browser displays the system or block's dialog box (**Open System** works the same way as double-clicking on the block in a block diagram). Similarly, if the block's `OpenFcn` callback parameter is defined, clicking on **Open System** while that block is selected executes the callback function.

You can direct the Browser to look beyond the dialog box or callback function by selecting the block in the blocks list, then clicking on **Look Into System**. The Browser displays the underlying system or block.

Displaying List Contents Alphabetically

By default, the systems list indicates the hierarchy of the model. Systems that contain systems are preceded with a plus sign (+). When those systems are expanded, the Browser displays a minus sign (–) before their names. To display systems alphabetically, select the **Display Alphabetical List** menu item on the **Options** menu.

Documentation Errata

The *Using Simulink* manual contains several errors. Note that these errors have been corrected in the most recent online copy of the manual.

- Dead Zone block: The paragraph that describes the output signal when the lower and upper limits are the same is incorrect. Instead, the behavior of the output obeys the rules described in the three bulleted sentences that appear above the paragraph on the block reference page.
- From File and From Workspace blocks: The rules that determine how the blocks handle two or more columns at the same time value are incorrect. The rule stated in the first bullet is correct and applies to all situations in which time values are repeated. Please disregard the second bullet.
- Random Number block: The sample time for this block is continuous and the default **Sample time** parameter value is 0. (The block dialog box has been revised and is described elsewhere in this chapter.)
- Relational Operator block: In the table that lists operators and the output they produce, the != operator should be ~=.
- Scope block: The sample time for this block is inherited from the driving block but can be set from the **Settings** page of the **Properties** dialog box.
- Sine Wave block: The sample time for this block is continuous and the default **Sample time** parameter value is 0.

Simulink 2.0 New Features

Introduction

This chapter provides a brief description of many of the significant new features included in Simulink 2.

Loading Simulink 1.3 Models

Simulink 2 loads Simulink 1.3 models without any problems, although it issues a warning message that indicates you are loading a model created by a previous version. Simulink automatically converts older models to the new model file format and marks a loaded model as modified; when you close a model, Simulink asks whether you want to save the modified file.

Updating Simulink 1.3 Models

The new `slupdate` command updates Simulink 1.3 models containing specific blocks to Simulink 2 format. If your model includes any of these blocks, run `slupdate` to convert them:

- The From Workspace block's default value for the buffer parameter has been changed to `inf`.
- The Graph scope's function has been replaced with the new Scope block.
- The Hit Crossing block is now a built-in block.
- The Memory block is now a built-in block.
- The Pulse Generator block has been rewritten.
- The Quantizer block is now a built-in block.
- The 2-D Table Look-Up block is now a built-in block and has been renamed Table Look-Up (2-D).

Also, `slupdate` calls `addterms` to terminate any unconnected input and output ports by attaching Ground and Terminator blocks, respectively. These blocks are described in *Using Simulink*.

To update a Simulink 1.3 model, open the model, then enter

```
slupdate('sys')
```

where sys is the model name.

For each out-of-date block, Simulink asks whether you want to update it.

Loading Models Containing Reset Integrator Blocks

Although the function performed by the Reset Integrator block has been built into the Integrator block, `slupdate` does not replace Reset Integrator blocks with Integrator blocks. It is difficult to automatically configure the new Integrator block to satisfy all the different ways the Reset Integrator block could be used in Simulink 1.3 models. The Reset Integrator block is not available in the Linear library; we encourage you to apply the Integrator block instead.

Changes to the Simulink User Interface

Simulink 2 has a substantially new look, with a new menu structure, redesigned block dialog boxes, and revised block library layout.

All Simulink commands are organized under four top-level menus: **File**, **Edit**, **Simulation**, and **Format**. The menu items are described in the sections of the manual that discuss the functions they perform.

The **Simulation Parameters** dialog box is new for Simulink 2. Changes are discussed in “Running a Simulation” on page 2-7.

The **Mask Editor** dialog box has been redesigned, making it easier to mask blocks. Changes to masking are described in “Using Masks to Customize Blocks” on page 2-9.

Improved Documentation

Using Simulink has been completely rewritten and has been improved in many ways:

- The new manual combines the previous edition of the *Simulink User's Guide* and the *Simulink Version 1.3 Release Notes* into a single book.
- An expanded chapter on building models provides clearer and more complete instructions. A useful table of keyboard shortcuts offers a quick guide for more experienced users.
- Additional chapters and appendices provide information not previously available.

Creating a Model

For more information about the features discussed in this section, see Chapter 3 of *Using Simulink*.

Starting Simulink

You can start Simulink 2 in these ways:

- Enter a model file name to display the block diagram for that model (the same way you did using Simulink 1.3).
- Enter `simulink` in the MATLAB command window to display the Simulink block library.
- Microsoft Windows and Macintosh users can click on the Simulink toolbar button to display the Simulink block library and get a new model window. The toolbar button looks like this:

Macintosh users can launch MATLAB by double-clicking on a model file icon.

Model and Block Parameters

Many model and block parameter names have changed in Simulink 2. If you use the `set_param` command to set model or block parameters, you need to make sure the parameter names are correct. Chapter 11 of *Using Simulink* discusses the `set_param` command and Appendix A lists model and block parameters.

Working with Blocks

Simulink 2 provides many enhancements for blocks and block libraries.

Changes to the Block Libraries

The Simulink block libraries have been modified. Each library has a new icon that depicts the kind of blocks it contains. Also, the layout of block icons within each library has been modified to make it easier to find the block you want. Many block icons have been changed and icon sizes are more consistent.

The Extras Library

The Extras block library, which contains blocks that supplement built-in blocks, has been moved to the new Blocksets and Toolboxes library. When you open that icon, Simulink searches through your installed software and displays icons for any blocksets and toolboxes it finds. The Extras library always appears in the list. To examine blocks in the Extras library, open that icon.

Blocks in the Extras library are not documented.

The Demos Library

A separate and expanded Demos library provides more demo models that illustrate Simulink 2 and MATLAB 5 features.

Changes to Block Dialog Boxes

All block dialog boxes have been redesigned. Block descriptions (these appear below the block type on the dialog box) have been rewritten to provide more useful information. The **Help** button now accesses the online reference page for the block, using the new Simulink Block Browser, described on page 2-12.

Block Callback Routines

Simulink 2 provides additional callback parameters that enable you to define callback routines that execute when a particular action is performed on a block or model. For example, you can define a callback routine associated with a block's `ModelCloseFcn` parameter that executes when the model is closed.

See Chapter 3 of *Using Simulink* for more information.

Moving a Block Name

You can move a block name to the opposite side of the block by dragging the name. This is equivalent to using the **Flip Name** menu item from the **Format** menu.

Additional Colors

Simulink 2 provides more colors for the screen background and the block background and foreground. The new colors are light blue, dark green, orange, and gray.

Working with Lines

Simulink 2 provides many enhancements for lines.

Drawing Lines Between Blocks

Simulink 2 draws connecting lines either as straight lines or as perpendicular horizontal and vertical line segments. If you hold down the **Shift** key while drawing a connecting line, Simulink draws a straight diagonal line that snaps to the target input port. When you draw a line to connect blocks, if the cursor is within the target block, the line is connected to the closest input port.

Labeling Signals

In Simulink 2, you can label signals to annotate your model. Signal labels can be located at either end or at the center of one or more line segments. Labels remain attached to lines as they are moved. Labels themselves can be copied or moved using drag-and-drop techniques.

Signal Label Propagation

Signal label propagation is the automatic labeling of a line carrying a signal that is labeled someplace else in a model. Simulink 2 supports the propagation of signal labels through connecting blocks, stored in the Connections block library.

Labeling Ports

Simulink 2 labels ports on Subsystem blocks using the Inport and Outport block names in the underlying subsystem.

Copying a Model to the PC Clipboard

Microsoft Windows users can copy the block diagram into the clipboard for use with another application. Choose **Copy Model** from the **Edit** menu. The default copy format is Windows Metafile. You can change the format by choosing **Preferences** from the MATLAB command window **File** menu. Then, select the **Copying Options** tab.

Running a Simulation

Simulink 2 provides many improvements in this area. All features discussed in this section are described in detail in Chapter 4 of *Using Simulink*.

Expanded Simulation Parameters Dialog Box

A new **Simulation Parameters** dialog box provides more information about solvers and gives you more error control. Also, it is now easier to manage workspace I/O and levels of diagnostic messages and intervention.

The **Simulation Parameters** dialog box consists of three “pages.” The **Solver** page enables you to select a solver and specify its parameters. The **Workspace I/O** page enables you to manage input from and output to the workspace. The **Diagnostics** page enables you to control the level of intervention for certain events.

In Simulink 1.3, to provide output at specified times, you would use the `HitTimes` parameter. In Simulink 2, you select the **Produce additional output** choice on the **Output options** list, on the **Solver** page.

State-of-the-Art Integrators

Simulink 2 incorporates the set of integration algorithms developed for the MATLAB ODE suite. These solvers provide faster, more accurate simulation results. The ODE suite includes variable-order and fixed-step nonstiff and stiff solvers. Selecting the appropriate solver is easier with the improved **Simulation Parameters** dialog box.

An important additional benefit of the new solvers is that it is no longer necessary, or even advisable, to adjust step size to get better granularity in the simulation results. The variable-step solvers automatically set step sizes to provide accurate results. Also, because Simulink 2 provides fixed-step solvers, it is no longer necessary to set minimum and maximum step sizes to the same value to force the use of a fixed-step solver.

This table indicates, for each integration method supported in Simulink 1.3, the corresponding solver provided in Simulink 2.

Table 2-1: Simulink 1.3 Integrators and Simulink 2 Solvers

If you used this integrator in Simulink 1.3	Consider using this solver in Simulink 2
`linsim`	`ode45` (nonstiff) or `ode15s` (stiff)
`rk23`	`ode23`
`rk45`	`ode45`
`adams`	`ode113`
`gear`	`ode15s`
`euler`	`ode1`

Running a Simulation from the Command Line

In Simulink 2 you can use the `sim` and `simset` commands to run a simulation from the command line. These commands give you more control of simulation parameters and provide command line access to all parameters that can be set on the **Simulation Parameters** dialog box. The `simget` command enables you to obtain values of simulation parameters and solver properties for a model.

Using Masks to Customize Blocks

Masking enables users to create block dialog boxes or customize block icons.

Simulink 2 provides these enhancements to masking:

- A new user interface
- An easier way to define dialog box prompts, including a more direct way to associate a variable to a block parameter
- The ability to add pop-up menus and check boxes to the mask dialog box
- An improved way to define initialization commands, including the ability to assign a user-entered value to a variable without evaluating it
- A separate workspace for each masked block (similar to an M-file function)
- A simpler way to examine a block's mask
- The ability to examine the blocks in a masked subsystem without destroying the mask
- New plotting options
- New options for controlling the appearance of icons

In Simulink 2, mask parameter names have changed. Simulink preserves the integrity of masked blocks created using all prior Simulink versions. Masking is described in Chapter 6 of *Using Simulink*, and mask parameters are listed in Appendix A.

Executing a Callback Routine when Opening a Masked Block

In Simulink 1.3, to execute a MATLAB command when the user double-clicked on a masked block, you entered an `eval` command in the mask's **Dialog String** parameter. To do this in Simulink 2, you associate a callback routine with the block's `OpenFcn` parameter. Any Simulink 1.x blocks that use `eval` commands in way are automatically converted to use the `OpenFcn` parameter instead. See Chapter 3 of *Using Simulink* for more information.

You can assign the callback routine to the block's `OpenFcn` parameter using the `set_param` command. For more information, see Chapter 11 of *Using Simulink*.

Conditionally Executed Subsystems

Conditionally executed subsystems are subsystems whose execution depends on their input. Conditionally executed subsystems are described in Chapter 7 of *Using Simulink*.

Simulink 2 provides support for three types of conditionally executed subsystems:

Enabled subsystem	Executes while the control signal is positive, starting execution at the simulation step where the control signal crosses zero (from the negative to the positive direction) and continuing execution while the control signal remains positive.
Triggered subsystem	Executes at the simulation step when a trigger event occurs. A trigger event can occur on the rising edge, falling edge, or either of a trigger signal.
Triggered and enabled subsystem	Executes once on the simulation step when a trigger event occurs if the enable control signal has a positive value at that step.

Conditionally executed subsystems are useful in a variety of applications. For example, in the automotive industry, triggered subsystems can be used to model the dynamics of an internal combustion engine. In the aerospace industry, enabled subsystems can be used to model complex flight control laws, where different controllers are enabled during different flight regimes.

S-Functions

If you are writing C MEX-file S-functions and place an S-function in an enabled subsystem configured to reset its states, the `mdlInitializeConditions` function is called upon reset. To figure out if `mdlInitializeConditions` is called from a reset or at simulation start, use the `ssIsFirstInitCond(S)` macro.

It is now possible to write variable step S-functions. An optional function, `mdlGetTimeOfNextVarHit`, provides the time of the next hit for the S-Function block.

Otherwise, S-functions work as they did in Simulink 1.3.

For more information about S-functions, see Chapter 8 of *Using Simulink*.

Simulink Blocks

This section discusses new, revised, and obsoleted blocks. All blocks are described in Chapter 9 of *Using Simulink*.

Online Block Reference Browser

The current reference pages for Simulink blocks are available as online help. You can access these pages in two ways:

- By clicking on the **Help** button on any block dialog box. The reference page for the block is displayed.
- By accessing the MATLAB Help Desk, then selecting **Simulink Blocks** from the Simulink Topics area.
  - You can access the MATLAB Help Desk on any supported platform by entering the `helpdesk` command in the MATLAB command window.
  - If you're using Microsoft Windows or a Macintosh, you can also access the MATLAB Help Desk by clicking on the **Help** toolbar button (a question mark appears on the icon) or selecting the **Help Desk** menu item from the MATLAB **Help** menu.

Revised Block Dialog Boxes

All block dialog boxes have been redesigned for Simulink 2. In addition to a more visually appealing layout, each dialog box includes an **Apply** button to accept current settings and keep the dialog box open, a **Revert** button to restore the original settings when the block was most recently opened, a **Close** button that applies the changes and closes the dialog box, and a **Help** button that accesses the Online Block Reference Browser, described above.

The block descriptions have been rewritten.

Vectorization of Blocks

A *vectorized* block can accept a vector input signal or generate a vector output signal, or both. In Simulink 2, almost all blocks are vectorized. To find out whether a block is vectorized, consult the online Block Browser or check the reference page for the block in the manual. At the end of each block reference page a table provides information about block characteristics, including whether the block is vectorized.

New, Enhanced, and Renamed Blocks

Simulink 2 provides several new blocks, enhanced versions of existing blocks, and renamed blocks. All blocks are described in detail in Chapter 9 of *Using Simulink* and in the online Block Browser.

Algebraic Constraint

The Algebraic Constraint block constrains the input signal f(*z*) to zero and outputs an algebraic state *z*. The block outputs the value necessary to produce a zero at the input. The output must affect the input through some feedback path. This enables you to specify algebraic equations for index 1 differential/algebraic systems (DAE's).

Backlash

The Backlash block no longer has an **Initial input** parameter. The initial center of the deadband width is defined by the **Initial output** parameter.

Data Store Memory, Data Store Read, and Data Store Write

The Data Store Memory, Data Store Read, and Data Store Write blocks enable the model to write and read data to and from a memory region during a simulation.

Discrete-Time Integrator

The Discrete-Time Integrator block enables you to define limits on the integration, which provides the capabilities of the (obsoleted) Discrete-Time Limited Integrator. The block supports these integration methods: Forward Euler, Backward Euler, and Trapezoidal.

Display

The Display block shows the value of its input signal. You can control the display format and the frequency of the display. You can use the block as a floating Display to probe different signals during a simulation.

Enable

The Enable block is used with conditionally executed subsystems. Adding an Enable block to a subsystem creates an *enabled* subsystem. Adding both an Enable and a Trigger block creates a *triggered and enabled* subsystem. For more information about conditionally executed subsystems, see page 2-10 of this book or Chapter 7 of *Using Simulink*.

Fcn

The rules of precedence for operations for the Simulink 1.3 Fcn block did not conform to the C language standard. The Simulink 2 Fcn block conforms to these standards.

Filter

The Filter block has been renamed Discrete Filter.

From

The From block, when used with a Goto block, provides a convenient way to pass a signal from one block to another without physically connecting the blocks.

Goto

The Goto block, when used with a From block, provides a convenient way to pass a signal from one block to another without physically connecting the blocks.

Goto Tag Visibility

The Goto Tag Visibility block defines the scope of a Goto block tag.

Ground

Connecting a Ground block to a block's input port prevents Simulink from issuing a warning message about the block's unconnected port. The block outputs a zero-valued signal.

Hit Crossing

A rewritten Hit Crossing block accurately enables you to detect when the input signal crosses a particular value.

IC

The IC block enables you to define an initial value for a signal. You can also use the IC block to provide an initial guess to the algebraic loop solver, described on page 2-19.

Inner Product

The Inner Product block has been renamed Dot Product.

Integrator

The revised Integrator block now combines features included in the now obsoleted Limited Integrator and Reset Integrator blocks. In addition, you can add a port on the block to output the state. Also, you can specify the absolute tolerance for the block’s state.

Memory

You can choose whether or not the Memory block’s sample time is inherited from its driving block. The block dialog box contains a check box labeled **Inherit sample time**. If the check box is selected, the block inherits its sample time from the driving block. If the box is not selected, the block’s sample time is continuous.

All Memory blocks in existing models have a sample time of continuous, although the default sample time for Memory blocks copied from the Nonlinear library is inherited. For Memory blocks to work as they did in Simulink 1.3 (with continuous sample time), make sure the check box described above is not selected.

MinMax

The MinMax block detects either the minimum or maximum of its input signal(s).

Multiport Switch

The Multiport Switch block chooses a block input from among multiple inputs. An integer-valued control input determines which input to pass through to the output port.

Note

The Note block has been removed. You can provide model annotations by creating an annotation, described in Chapter 3 of *Using Simulink*. Simulink automatically converts Note blocks to annotations in existing models.

Outport

The Outport block adds an option that, when used in a conditionally executed subsystem, allows the block to control whether its output is reset to an initial value or held at its most recent value while the subsystem is disabled.

Pulse Generator

The Pulse Generator block parameters have changed: The **Pulse period** parameter is now **Period**, the **Pulse width** parameter is now **Duty cycle**, the **Pulse height** parameter is now **Amplitude**, and the **Pulse start time** parameter is now **Start time**. You can use `slupdate` to replace old Pulse Generator blocks. `slupdate` converts the **Pulse width** parameter to the **Duty cycle** parameter.

Ramp

The Ramp block provides a signal that starts at a specified time and value and changes by a specified rate.

Rate Limiter

The **Rising slew rate** and **Falling slew rate** parameters now accept values of `inf` and `–inf`, respectively. These values pass the input through the block without applying limits.

Relay

The Relay block **Input for on** and **Input for off** parameters have been renamed to **Switch on point** and **Switch off point**.

Scope

An enhanced oscilloscope-like Scope block provides vastly improved graphical display of signals. The Scope block allows you to zoom in on the block input in the *x* (time) direction, *y* direction, or both directions; display all the input to the block; limit the data displayed; and save the signal data to the workspace at the end of simulation.

Selector

The Selector block acts like a patch panel for cross wiring of input vector elements. You enter a vector parameter that indicates the input vector elements that make up the block output.

Signal Generator

The Signal Generator's dialog box has been rearranged to simplify choosing a wave form and defining signal parameters. The default frequency is now Hertz.

Step Fcn

The Step Fcn block has been renamed Step.

Terminator

Connecting a block's output port to a Terminator block prevents Simulink from issuing warning messages about unconnected ports. The block does not process the signal.

To File

The To File block provides **Decimation** and **Sample time** parameters to limit the amount of data written to the file.

To Workspace

The To Workspace block provides **Decimation** and **Sample time** parameters to limit the amount of data written to the workspace variable.

Trigger

The Trigger block is used with conditionally executed subsystems. Adding a Trigger block to a subsystem creates a *triggered* subsystem. Adding both an Enable and a Trigger block creates a *triggered and enabled* subsystem. These special subsystems are described in Chapter 7 of *Using Simulink*.

2-D Look-Up Table

The 2-D Look-Up Table has been renamed Look-Up Table (2-D).

Variable Transport Delay

In Simulink 1.3, the **Initial input** parameter was not being set. This has been corrected in the current release.

Width

The Width block generates as output the width of the input vector.

Obsoleted Blocks

Several blocks are no longer available. The functions they perform are included in other blocks:

- The Discrete-Time Limited Integrator block has been replaced by the Discrete-Time Integrator block.
- The Limited Integrator block has been replaced by the Integrator block.
- The Note block has been replaced by the model annotation feature.
- The Reset Integrator block has been replaced by the Integrator block.

In addition, the renamed blocks: Filter, Inner Product, Step Fcn, and 2-D Look-Up Table, are no longer available.

Additional Topics

Algebraic Loops

The algebraic loop solver has been improved for Simulink 2 and is able to solve a larger class of algebraic loops. It can now attempt to solve algebraic loops that have multirate components, as well as loops containing blocks with nonsmooth outputs (such as the Abs, Saturation, or Quantizer blocks).

Algebraic loops and the algebraic loop solver are described in Chapter 10 of *Using Simulink*.

Zero Crossing Detection

A *zero crossing* occurs when a signal makes a transition to zero, crosses zero, or makes a transition from zero. A zero crossing also occurs when a signal reaches some defined threshold (not necessarily at zero), such as an upper limit in a Saturation block.

Many Simulink blocks have built-in (intrinsic) detection of zero crossings. Simulink can detect the point where a signal crosses zero to within computer tolerance. This feature, which results in more accurate simulations, is discussed in Chapter 10 of *Using Simulink*.

Model Construction Commands

Several model construction commands have been added:

- `find_system`, which finds a Simulink system or block
- `gcb`, which gets the path of the current block
- `gcs`, which gets the path of the current system
- `save_system`, which saves a Simulink system

These commands are described in Chapter 11 of *Using Simulink*.

Also, the new `Lines` property enables you to obtain a structure array of all the lines in a block diagram:

```
get_param('sys','Lines')
```

returns a structure array of all the lines in the model named `sys`. The fields of each structure in the array are:

- `Handle` is the handle to the line
- `Name` is the line's name
- `Parent` is the handle to the subsystem or block diagram owning the line
- `SrcBlock` is the handle to the source block driving the line
- `SrcPort` is the port number of the source block driving the line
- `DstBlock` is the handle to the block being driven by the line
- `DstPort` is the port number of the destination block
- `Points` is the array of points describing the line
- `Branch` is the structure array of any branch lines on this line

Model File Format

Simulink 2 saves models in a structured file format, which results in faster loading and saving of models, and produces more readable model files. Also, model files are now easier to post-process or transfer to other applications.

The model file format is described in Appendix B of *Using Simulink*.