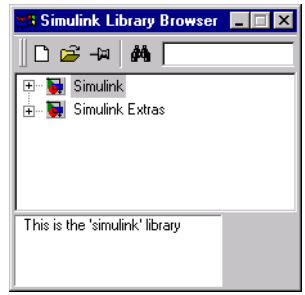
Starting Simulink

To start Simulink, you must first start MATLAB. Consult your MATLAB documentation for more information. You can then start Simulink in two ways:

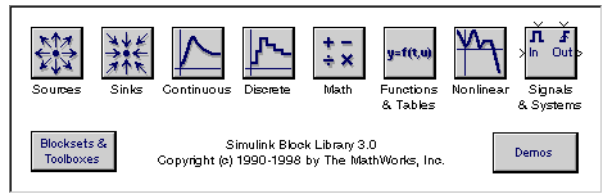
- Click on the Simulink icon  on the MATLAB toolbar.
- Enter the `simulink` command at the MATLAB prompt.

On Microsoft Windows platforms, starting Simulink displays the Simulink Library Browser.



The Library Browser displays a tree-structured view of the Simulink block libraries installed on your system. You can build models by copying blocks from the Library Browser into a model window (this procedure is described later in this chapter).

On UNIX platforms, starting Simulink displays the Simulink block library window.



The Simulink library window displays icons representing the block libraries that come with Simulink. You can create models by copying blocks from the library into a model window.

Note On Windows, you can display the Simulink library window by right-clicking the Simulink node in the Library Browser window.

Creating a New Model

To create a new model, click the **New** button on the Library Browser's toolbar (Windows only) or choose **New** from the library window's **File** menu and select **Model**. You can move the window as you do other windows. Chapter 2 describes how to build a simple model. "Modeling Equations" on page 3–58 describes how to build systems that model equations.

Editing an Existing Model

To edit an existing model diagram, either:

- Choose the **Open** button on the Library Browser's toolbar (Windows only) or the **Open** command from the Simulink library window's **File** menu and then choose or enter the model filename for the model you want to edit.
- Enter the name of the model (without the `.mdl` extension) in the MATLAB command window. The model must be in the current directory or on the path.

Entering Simulink Commands

You run Simulink and work with your model by entering commands. You can enter commands by:

- Selecting items from the Simulink menu bar
- Selecting items from a context-sensitive Simulink menu (Windows only)
- Clicking buttons on the Simulink toolbar (Windows only)
- Entering commands in the MATLAB command window

Using the Simulink Menu Bar to Enter Commands

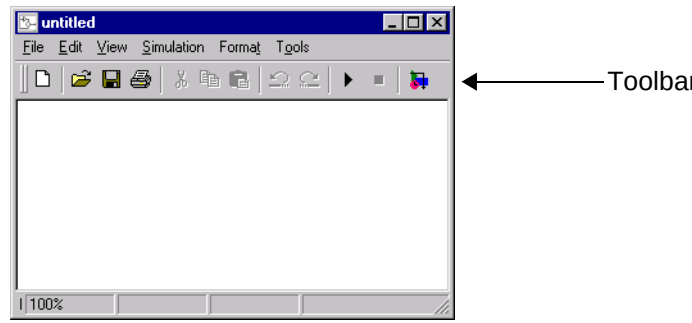
The Simulink menu bar appears near the top of each model window. The menu commands apply to the contents of that window.

Using Context-Sensitive Menus to Enter Commands

The Windows version of Simulink displays a context-sensitive menu when you click the right mouse button over a model or block library window. The contents of the menu depend on whether a block is selected. If a block is selected, the menu displays commands that apply only to the selected block. If no block is selected, the menu displays commands that apply to a model or library as a whole.

Using the Simulink Toolbar to Enter Commands

Model windows in the Windows version of Simulink optionally display a toolbar beneath the Simulink menu bar. To display the toolbar, check the **Toolbar** option on the Simulink **View** menu.



The toolbar contains buttons corresponding to frequently used Simulink commands, such as those for opening, running, and closing models. You can run such commands by clicking on the corresponding button. For example, to open a Simulink model, click on the button containing an open folder icon. You can determine which command a button executes by moving the mouse pointer over the button. A small window appears containing text that describes the button. The window is called a tooltip. Each button on the toolbar displays a tooltip when the mouse pointer hovers over it. You can hide the toolbar by unchecking the **Toolbar** option on the Simulink **View** menu.

Using the MATLAB Window to Enter Commands

When you run a simulation and analyze its results, you can enter MATLAB commands in the MATLAB command window. Running a simulation is discussed in Chapter 4, and analyzing simulation results is discussed in Chapter 5.

Undoing a Command

You can cancel the effects of up to 101 consecutive 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 an **Undo** command by choosing **Redo** from the **Edit** menu.

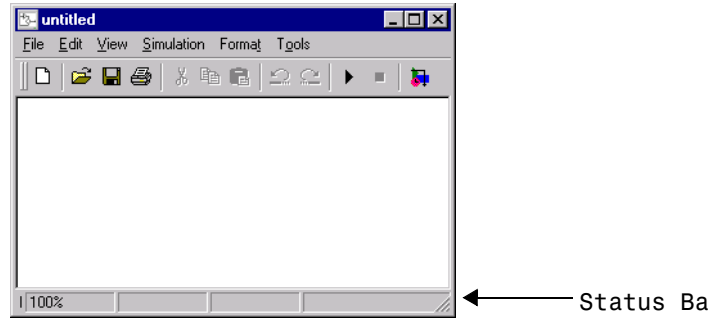
Simulink Windows

Simulink uses separate windows to display a block library browser, a block library, a model, and graphical (scope) simulation output. These windows are not MATLAB figure windows and cannot be manipulated using Handle Graphics® commands.

Simulink windows are sized to accommodate the most common screen resolutions available. If you have a monitor with exceptionally high or low resolution, you may find the window sizes too small or too large. If this is the case, resize the window and save the model to preserve the new window dimensions.

Status Bar

The Windows version of Simulink displays a status bar at the bottom of each model and library window.



When a simulation is running, the status bar displays the status of the simulation, including the current simulation time and the name of the current solver. You can display or hide the status bar by checking or unchecking the **Status Bar** item on the Simulink **View** menu.

Zooming Block Diagrams

Simulink allows you to enlarge or shrink the view of the block diagram in the current Simulink window. To zoom a view:

- Select **Zoom In** from the **View** menu (or type `r`) to enlarge the view.
- Select **Zoom Out** from the **View** menu (or type `v`) to shrink the view.
- Select **Fit System to View** from the **View** menu (or press the space bar) to fit the diagram to the view.
- Select **Normal** from the **View** menu to view the diagram at actual size.

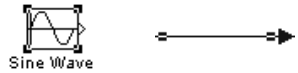
By default, Simulink fits a block diagram to view when you open the diagram either in the model browser's content pane or in a separate window. If you change a diagram's zoom setting, Simulink saves the setting when you close the diagram and restores the setting the next time you open the diagram. If you want to restore the default behavior, choose **Fit System to View** from the **View** menu the next time you open the diagram.

Selecting Objects

Many model building actions, such as copying a block or deleting a line, require that you first select one or more blocks and lines (objects).

Selecting One Object

To select an object, click on it. Small black square "handles" appear at the corners of a selected block and near the end points of a selected line. For example, the figure below shows a selected Sine Wave block and a selected line:



When you select an object by clicking on it, any other selected objects become deselected.

Selecting More than One Object

You can select more than one object either by selecting objects one at a time, by selecting objects located near each other using a bounding box, or by selecting the entire model.

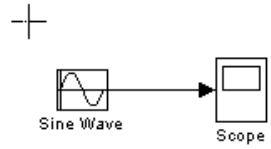
Selecting Multiple Objects One at a Time

To select more than one object by selecting each object individually, hold down the **Shift** key and click on each object to be selected. To deselect a selected object, click on the object again while holding down the **Shift** key.

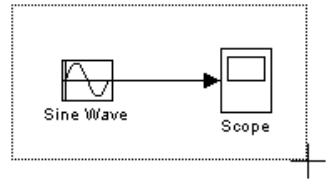
Selecting Multiple Objects Using a Bounding Box

An easy way to select more than one object in the same area of the window is to draw a bounding box around the objects.

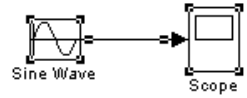
1 Define the starting corner of a bounding box by positioning the pointer at one corner of the box, then pressing and holding down the mouse button. Notice the shape of the cursor.



2 Drag the pointer to the opposite corner of the box. A dotted rectangle encloses the selected blocks and lines.



3 Release the mouse button. All blocks and lines at least partially enclosed by the bounding box are selected.



Selecting the Entire Model

To select all objects in the active window, choose **Select All** from the **Edit** menu. You cannot create a subsystem by selecting blocks and lines in this way; for more information, see “Creating Subsystems” on page 3–51.

Blocks

Blocks are the elements from which Simulink models are built. You can model virtually any dynamic system by creating and interconnecting blocks in appropriate ways. This section discusses how to use blocks to build models of dynamic systems.

Block Data Tips

On Microsoft Windows, Simulink displays information about a block in a pop-up window when you allow the pointer to hover over the block in the diagram view. To disable this feature or control what information a data tip includes, select **Block Data Tips** from the Simulink **View** menu.

Virtual Blocks

When creating models, you need to be aware that Simulink blocks fall into two basic categories: nonvirtual and virtual blocks. Nonvirtual blocks play an active role in the simulation of a system. If you add or remove a nonvirtual block, you change the model's behavior. Virtual blocks, by contrast, play no active role in the simulation. They simply help to organize a model graphically. Some Simulink blocks can be virtual in some circumstances and nonvirtual in others. Such blocks are called conditionally virtual blocks. The following table lists Simulink's virtual and conditionally virtual blocks.

Table 3-1: Virtual Blocks

Block Name	Condition Under Which Block Will Be Virtual
Bus Selector	Always virtual.
Data Store Memory	Always virtual.
Demux	Always virtual.
Enable Port	Always virtual.
From	Always virtual.
Goto	Always virtual.
Goto Tag Visibility	Always virtual.

Table 3-1: Virtual Blocks (Continued)

Block Name	Condition Under Which Block Will Be Virtual
Ground	Always virtual.
Inport	Always virtual *unless* the block resides in a conditionally executed subsystem *and* has a direct connection to an outport block.
Mux	Always virtual.
Outport	Virtual if the block resides within any subsystem block (conditional or not), and does *not* reside in the root (top-level) Simulink window.
Selector	Always virtual.
Subsystem	Virtual if the block is not conditionally executed.
Terminator	Always virtual.
Test Point	Always virtual.
Trigger Port	Virtual if the outport port is not present.

Copying and Moving Blocks from One Window to Another

As you build your model, you often copy blocks from Simulink block libraries or other libraries or models into your model window. To do this, follow these steps:

1. Open the appropriate block library or model window.
2. Drag the block you want to copy into the target model window. To drag a block, position the cursor over the block icon, then press and hold down the mouse button. Move the cursor into the target window, then release the mouse button.

You can also drag blocks from the Simulink Library Browser into a model window. See “Browsing Block Libraries” on page 3-25 for more information.

Note Simulink hides the names of Sum, Mux, Demux, and Bus Selector blocks when you copy them from the Simulink block library to a model.This is done to avoid unnecessarily cluttering the model diagram. (The shapes of these blocks clearly indicates their respective functions.)

You can also copy blocks by using the **Copy** and **Paste** commands from the **Edit** menu:

1. Select the block you want to copy.
2. Choose **Copy** from the **Edit** menu.
3. Make the target model window the active window.
4. Choose **Paste** from the **Edit** menu.

Simulink assigns a name to each copied block. If it is the first block of its type in the model, its name is the same as its name in the source window. For example, if you copy the Gain block from the Math library into your model window, the name of the new block is Gain. If your model already contains a block named Gain, Simulink adds a sequence number to the block name (for example, Gain1, Gain2). You can rename blocks; see "Manipulating Block Names" on page 3–16.

When you copy a block, the new block inherits all the original block's parameter values.

Simulink uses an invisible five-pixel grid to simplify the alignment of blocks. All blocks within a model snap to a line on the grid. You can move a block slightly up, down, left, or right by selecting the block and pressing the arrow keys.

You can display the grid in the model window by typing the following command in the MATLAB window:

```
set_param('<model name>','showgrid','on')
```

To change the grid spacing, type:

```
set_param('<model name>','gridspacing',<number of pixels>)
```

For example, to change the grid spacing to 20 pixels, type:

```
set_param('<model name>','gridspacing',20)
```

For either of the above commands, you can also select the model, and then type `gcs` instead of `<model name>`.

You can copy or move blocks to compatible applications (such as word processing programs) using the **Copy**, **Cut**, and **Paste** commands. These commands copy only the graphic representation of the blocks, not their parameters.

Moving blocks from one window to another is similar to copying blocks, except that you hold down the **Shift** key while you select the blocks.

You can use the **Undo** command from the **Edit** menu to remove an added block.

Moving Blocks in a Model

To move a single block from one place to another in a model window, drag the block to a new location. Simulink automatically repositions lines connected to the moved block.

To move more than one block, including connecting lines:

1. Select the blocks and lines. If you need information about how to select more than one block, see “Selecting More than One Object” on page 3–7.

2. Drag the objects to their new location and release the mouse button.

Duplicating Blocks in a Model

You can duplicate blocks in a model as follows. While holding down the **Ctrl** key, select the block with the left mouse button, then drag it to a new location. You can also do this by dragging the block using the right mouse button. Duplicated blocks have the same parameter values as the original blocks. Sequence numbers are added to the new block names.

Specifying Block Parameters

The Simulink user interface lets you assign values to block parameters. Some block parameters are common to all blocks. Use the **Block Properties** dialog box to set these parameters. To display the dialog box, select the block whose

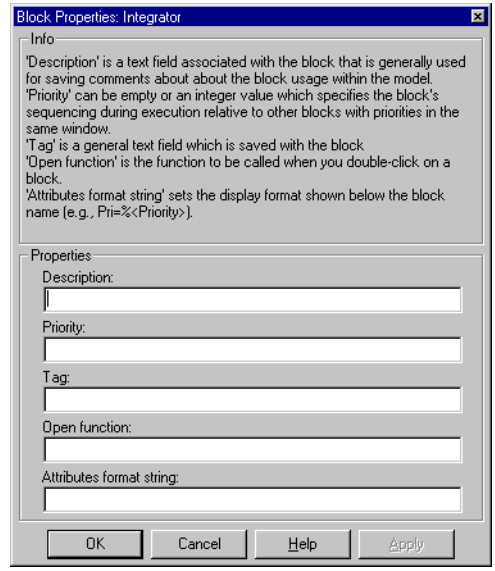
properties you want to set. Then select **Block Properties...** from Simulink's **Edit** menu. See "Block Properties Dialog Box" on page 3-13 for more information.

Other block parameters are specific to particular blocks. Use a block's block-specific parameter dialog to set these parameters. Double-click on the block to open its dialog box. You can accept the displayed values or change them. You can also use the `set_param` command to change block parameters. See `set_param` on page 10-24 for details.

Some block dialogs allow you to specify a data type for some or all of their parameters.The reference material that describes each block (in Chapter 8) shows the dialog box and describes the block parameters.

Block Properties Dialog Box

The **Block Properties** dialog box lets you set some common block parameters.



The dialog box contains the following fields:

Description

Brief description of the block's purpose.

Priority

Execution priority of this block relative to other blocks in the model. See "Assigning Block Priorities" on page 3-19 for more information.

Tag

A general text field that is saved with the block.

Open function

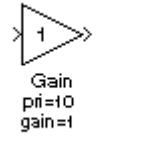
MATLAB (m-) function to be called when a user opens this block.

Attributes format string

Current value of the block's `AttributesFormatString` parameter. This parameter specifies which parameters to display beneath a block's icon. The attributes format string can be any text string with embedded parameter names. An embedded parameter name is a parameter name preceded by `%<` and followed by `>`, for example, `%<priority>`. Simulink displays the attributes format string beneath the block's icon, replacing each parameter name with the corresponding parameter value. You can use line feed characters (`\n`) to display each parameter on a separate line. For example, specifying the attributes format string

```
pri=%<prior  y>\ngain=%<Gain>
```

for a Gain block splays



If a parameter's value is not a string or an integer, Simulink displays `N/S` (not supported) for the parameter's value. If the parameter name is invalid, Simulink displays "`???`".

Deleting Blocks

To delete one or more blocks, select the blocks to be deleted and press the **Delete** or **Backspace** key. You can also choose **Clear** or **Cut** from the **Edit** menu. The **Cut** command writes the blocks into the clipboard, which enables

you to paste them into a model. Using the **Delete** or **Backspace** key or the **Clear** command does not enable you to paste the block later.

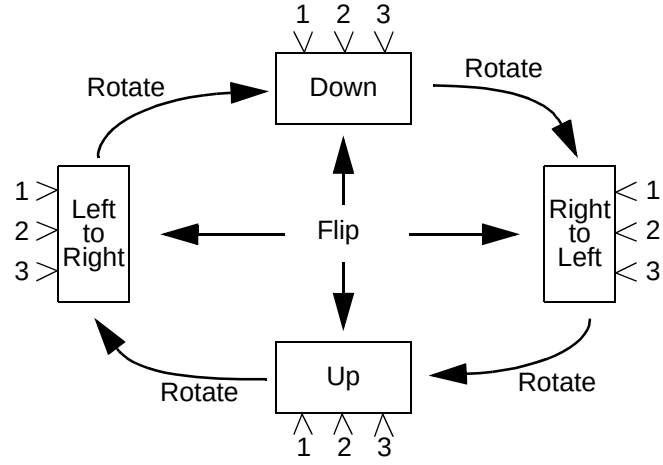
You can use the **Undo** command from the **Edit** menu to replace a deleted block.

Changing the Orientation of Blocks

By default, signals flow through a block from left to right. Input ports are on the left, and output ports are on the right. You can change the orientation of a block by choosing one of these commands from the **Format** menu:

- The **Flip Block** command rotates the block 180 degrees.
- The **Rotate Block** command rotates a block clockwise 90 degrees.

The figure below shows how Simulink orders ports after changing the orientation of a block using the **Rotate Block** and **Flip Block** menu items. The text in the blocks shows their orientation.

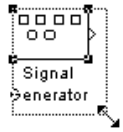


Resizing Blocks

To change the size of a block, select it, then drag any of its selection handles. While you hold down the mouse button, a dotted rectangle shows the new block size. When you release the mouse button, the block is resized.

For example, the figure below shows a Signal Generator block being resized. The lower-right handle was selected and dragged to the cursor position. When

the mouse button is released, the block takes its new size. This figure shows a block being resized.



Manipulating Block Names

All block names in a model must be unique and must contain at least one character. By default, block names appear below blocks whose ports are on the sides, and to the left of blocks whose ports are on the top and bottom, as this figure shows.

Changing Block Names

You can edit a block name in one of these ways:

- To replace the block name on a Microsoft Windows or UNIX system, click on the block name, then double-click or drag the cursor to select the entire name. Then, enter the new name.
- To insert characters, click between two characters to position the insertion point, then insert text.
- To replace characters, drag the mouse to select a range of text to replace, then enter the new text.

When you click the pointer someplace else in the model or take any other action, the name is accepted or rejected. If you try to change the name of a block to a name that already exists or to a name with no characters, Simulink displays an error message.

You can modify the font used in a block name by selecting the block, then choosing the **Font** menu item from the **Format** menu. Select a font from the **Set Font** dialog box. This procedure also changes the font of text on the block icon.

You can cancel edits to a block name by choosing **Undo** from the **Edit** menu.

Note If you change the name of a library block, all links to that block will become unresolved.

Changing the Location of a Block Name

You can change the location of the name of a selected block in two ways:

- By dragging the block name to the opposite side of the block
- By choosing the **Flip Name** command from the **Format** menu. This command changes the location of the block name to the opposite side of the block.

For more information about block orientation, see "Changing the Orientation of Blocks" on page 3–15.

Changing Whether a Block Name Appears

To change whether the name of a selected block is displayed, choose a menu item from the **Format** menu:

- The **Hide Name** menu item hides a visible block name. When you select **Hide Name**, it changes to **Show Name** when that block is selected.
- The **Show Name** menu item shows a hidden block name.

Displaying Parameters Beneath a Block's Icon

You can cause Simulink to display one or more of a block's parameters beneath the block's icon in a block diagram. You specify the parameters to be displayed in the following ways:

- By entering an attributes format string in the **Attributes format string** field of the block's **Block Properties** dialog box (see "Block Properties Dialog Box" on page 3-13)
- By setting the value of the block's `AttributesFormatString` property to the format string, using `set_param` (see `set_param` on page 10-24)

Disconnecting Blocks

To disconnect a block from its connecting lines, hold down the **Shift** key, then drag the block to a new location.

Vector Input and Output

Almost all Simulink blocks accept scalar or vector inputs, generate scalar or vector outputs, and allow you to provide scalar or vector parameters. These blocks are referred to in this manual as being *vectorized*.

You can determine which lines in a model carry vector signals by choosing **Wide Vector Lines** from the **Format** menu. When this option is selected, lines that carry vectors are drawn thicker than lines that carry scalars. The figures in the next section show scalar and vector lines.

If you change your model after choosing **Wide Vector Lines**, you must explicitly update the display by choosing **Update Diagram** from the **Edit** menu. Starting the simulation also updates the block diagram display.

Block descriptions in Chapter 8 discuss the characteristics of block inputs, outputs, and parameters.

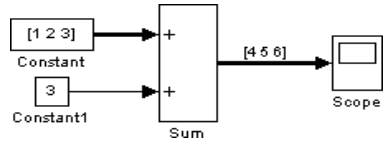
Scalar Expansion of Inputs and Parameters

Scalar expansion is the conversion of a scalar value into a vector of identical elements. Simulink applies scalar expansion to inputs and/or parameters for most blocks. Block descriptions in Chapter 8 indicate whether Simulink applies scalar expansion to a block's inputs and parameters.

Scalar Expansion of Inputs

When using blocks with more than one input port (such as the Sum or Relational Operator block), you can mix vector and scalar inputs. When you do this, the scalar inputs are expanded into vectors of identical elements whose widths are equal to the width of the vector inputs. (If more than one block input is a vector, they must have the same number of elements.)

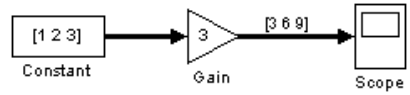
This model adds scalar and vector inputs. The input from block Constant1 is scalar expanded to match the size of the vector input from the Constant block. The input is expanded to the vector `[3 3 3]`.



Scalar Expansion of Parameters

You can specify the parameters for vectorized blocks as either vectors or scalars. When you specify vector parameters, each parameter element is associated with the corresponding element in the input vector(s). When you specify scalar parameters, Simulink applies scalar expansion to convert them automatically into appropriately sized vectors.

This example shows that a scalar parameter (the Gain) is expanded to a vector of identically valued elements to match the size of the block input, a three-element vector.



Assigning Block Priorities

You can assign evaluation priorities to nonvirtual blocks in a model. Higher priority blocks evaluate before lower priority blocks, though not necessarily before blocks that have no assigned priority.

You can assign block priorities interactively or programmatically. To set priorities programmatically, use the command

```
set_param(b,'Priority','n')
```

where `b` is a block path and `n` is any valid integer. (Negative numbers and 0 are valid priority values.) The lower the number, the higher the priority; that is, 2 is higher priority than 3. To set a block's priority interactively, enter the priority in the **Priority** field of the block's **Block Properties** dialog box (see "Block Properties Dialog Box" on page 3-13).

Using Drop Shadows

You can add a drop shadow to a block by selecting the block, then choosing **Show Drop Shadow** from the **Format** menu. When you select a block with a drop shadow, the menu item changes to **Hide Drop Shadow**. The figure below shows a Subsystem block with a drop shadow.

Libraries

Libraries enable users to copy blocks into their models from external libraries and automatically update the copied blocks when the source blocks change. Using libraries 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.

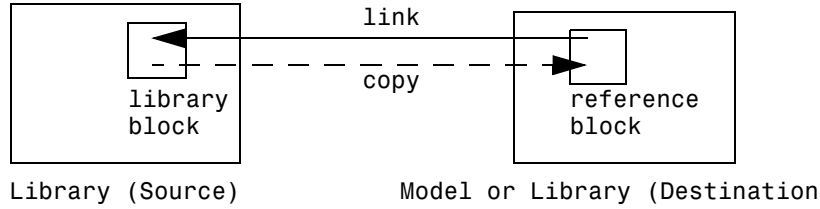
Library block – A block in a library.

Reference block – A copy of a library block.

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 this terminology.



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.

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 10.

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

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

You can copy a block from a library into a model by copying and pasting or dragging the block from the library window to the model window (see "Copying and Moving Blocks from One Window to Another" on page 3-10) or by dragging the block from the Library Browser (see "Browsing Block Libraries" on page 3-25) into the model window.

When you copy a library block into a model or another library, Simulink creates a link to the library block. The reference block is a copy of the library block. You can modify block parameters in the reference block 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.
- Add the directory that contains the required library to the MATLAB path and select **Update Diagram** from the **Edit** menu.
- Double-click on the reference block. On the dialog box that appears, correct the pathname and click on **Apply** or **Close**.

All blocks have a `LinkStatus` parameter that indicates whether the block is a reference block. The parameter can have these values:

- `'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 out-of-date 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
- When you query the `LinkStatus` parameter of a block using the `get_param` command (see “Getting Information About Library Blocks” on page 3-24)
- 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 a simple copy of the library block, unlinked to the

library block. 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

Use the `libinfo` command to get information about reference blocks in a system. The format for the command is

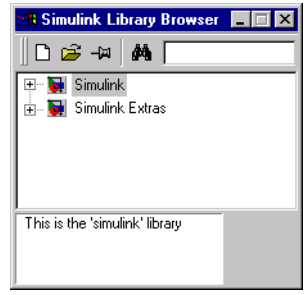
```
libdata = libinfo(sys)
```

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'`

Browsing Block Libraries

The Library Browser lets you quickly locate and copy library blocks into a model.



Note The Library Browser is available only on Microsoft Windows platforms.

You can locate blocks either by navigating the Library Browser's library tree or by using the Library Browser's search facility.

Navigating the Library Tree

The library tree displays a list of all the block libraries installed on the system. You can view or hide the contents of libraries by expanding or collapsing the tree using the mouse or keyboard. To expand/collapse the tree, click the +/- buttons next to library entries or select an entry and press the +/- or right/left arrow key on your keyboard. Use the up/down arrow keys to move up or down the tree.

Searching Libraries

To find a particular block, enter the block's name in the edit field next to the Library Browser's **Find** button and then click the **Find** button.

Opening a Library

To open a library, right-click the library's entry in the browser. Simulink displays an **Open Library** button. Select the **Open Library** button to open the library.

Creating and Opening Models

To create a model, select the **New** button on the Library Browser's toolbar. To open an existing model, select the **Open** button on the toolbar.

Copying Blocks

To copy a block from the Library Browser into a model, select the block in the browser, drag the selected block into the model window, and drop it where you want to create the copy.

Displaying Help on a Block

To display help on a block, right-click the block in the Library Browser and select the button that subsequently pops up.

Pinning the Library Browser

To keep the Library Browser above all other windows on your desktop, select the **PushPin** button on the browser's toolbar.

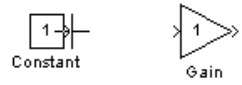
Lines

Lines carry signals. Each line can carry a scalar or vector signal. A line can connect the output port of one block with the input port of another block. A line can also connect the output port of one block with input ports of many blocks by using branch lines.

Drawing a Line Between Blocks

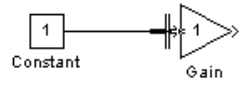
To connect the output port of one block to the input port of another block:

1. Position the cursor over the first block's output port. It is not necessary to position the cursor precisely on the port. The cursor shape changes to a cross hair.

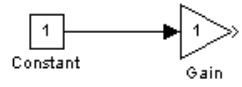


2. Press and hold down the mouse button.

3. Drag the pointer to the second block's input port. You can position the cursor on or near the port, or in the block. If you position the cursor in the block, the line is connected to the closest input port. The cursor shape changes to a double cross hair.



4. Release the mouse button. Simulink replaces the port symbols by a connecting line with an arrow showing the direction of the signal flow. You can create lines either from output to input, or from input to output. The arrow is drawn at the appropriate input port, and the signal is the same.

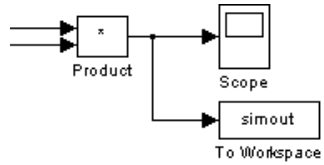


Simulink draws connecting lines using horizontal and vertical line segments. To draw a diagonal line, hold down the **Shift** key while drawing the line.

Drawing a Branch Line

A *branch line* is a line that starts from an existing line and carries its signal to the input port of a block. Both the existing line and the branch line carry the same signal. Using branch lines enables you to cause one signal to be carried to more than one block.

In this example, the output of the Product block goes to both the Scope block and the To Workspace block.



To add a branch line, follow these steps:

1. Position the pointer on the line where you want the branch line to start.
2. While holding down the **Ctrl** key, press and hold down the left mouse button.
3. Drag the pointer to the input port of the target block, then release the mouse button and the **Ctrl** key.

You can also use the right mouse button instead of holding down the left mouse button and the **Ctrl** key.

Drawing a Line Segment

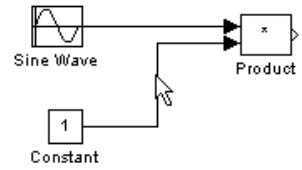
You may want to draw a line with segments exactly where you want them instead of where Simulink draws them. Or, you might want to draw a line before you copy the block to which the line is connected. You can do either by drawing line segments.

To draw a line segment, you draw a line that ends in an unoccupied area of the diagram. An arrow appears on the unconnected end of the line. To add another line segment, position the cursor over the end of the segment and draw another segment. Simulink draws the segments as horizontal and vertical lines. To draw diagonal line segments, hold down the **Shift** key while you draw the lines.

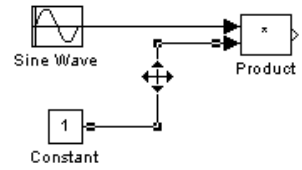
Moving a Line Segment

To move a line segment, follow these steps:

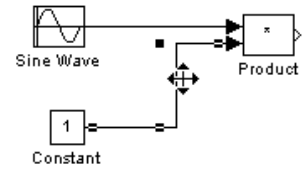
1 Position the pointer on the segment you want to move.



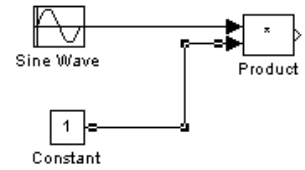
2 Press and hold down the left mouse button.



3 Drag the pointer to the desired location.



4 Release the mouse button.

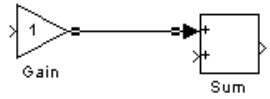


You cannot move the segments that are connected directly to block ports.

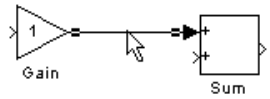
Dividing a Line into Segments

You can divide a line segment into two segments, leaving the ends of the line in their original locations. Simulink creates line segments and a vertex that joins them. To divide a line into segments, follow these steps:

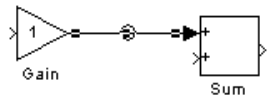
1 Select the line.



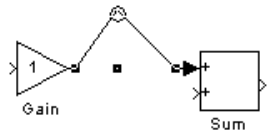
2 Position the pointer on the line where you want the vertex.



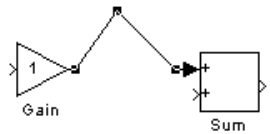
3 While holding down the **Shift** key, press and hold down the mouse button. The cursor shape changes to a circle that encloses the new vertex.



4 Drag the pointer to the desired location.



5 Release the mouse button and the **Shift** key.

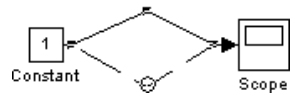


Moving a Line Vertex

To move a vertex of a line, follow these steps:

1. Position the pointer on the vertex, then press and hold down the mouse button. The cursor changes to a circle that encloses the vertex.

2. Drag the pointer to the desired location.



3. Release the mouse button.

Displaying Line Widths

You can display the widths of vector lines in a model by turning on **Vector 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. You can cause Simulink to use a thick line to display vector lines by selecting **Wide Vector Lines** from the **Format** menu.

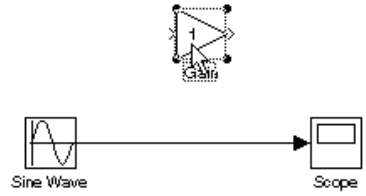
When you start a simulation or update the diagram and Simulink detects a mismatch of input and output ports, it displays an error message and shows line widths in the model.

Inserting Blocks in a Line

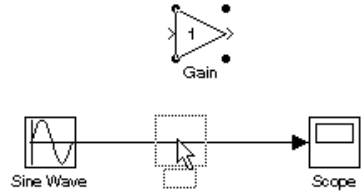
You can insert a block in a line by dropping the block on the line. Simulink inserts the block for you at the point where you drop the block. The block that you insert can have only one input and one output.

To insert a block in a line:

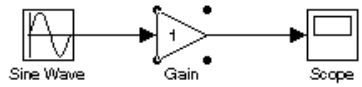
1 Position the pointer over the block and press the left mouse button.



2 Drag the block over the line in which you want to insert the block.



3 Release the mouse button to drop the block on the line. Simulink inserts the block where you dropped it.



Signal Labels

You can label signals to annotate your model. Labels can appear above or below horizontal lines or line segments, and left or right of vertical lines or line segments. Labels can appear at either end, at the center, or in any combination of these locations.

Using Signal Labels

To create a signal label, double-click on the line segment and type the label at the insertion point. When you click on another part of the model, the label fixes its location.

Note When you create a signal label, take care to double-click *on* the line. If you click in an unoccupied area close to the line, you will create a model annotation instead.

To move a signal label, drag the label to a new location on the line. When you release the mouse button, the label fixes its position near the line.

To copy a signal label, hold down the **Ctrl** key while dragging the label to another location on the line. When you release the mouse button, the label appears in both the original and the new locations.

To edit a signal label, select it:

- To replace the label, click on the label, then double-click or drag the cursor to select the entire label. Then, enter the new label.
- To insert characters, click between two characters to position the insertion point, then insert text.
- To replace characters, drag the mouse to select a range of text to replace, then enter the new text.

To delete all occurrences of a signal label, delete all the characters in the label. When you click outside the label, the labels are deleted. To delete a single occurrence of the label, hold down the **Shift** key while you select the label, then press the **Delete** or **Backspace** key.

To change the font of a signal label, select the signal, choose **Font** from the **Format** menu, then select a font from the **Set Font** dialog box.

Signal Label Propagation

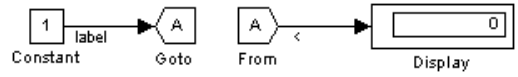
Signal label propagation is the automatic labeling of a line emitting from a connection block. Blocks that support signal label propagation are the Demux, Enable, From, Inport, Mux, Selector, and Subsystem blocks. The labeled signal

must be on a line feeding a connecting block and the propagated signal must be on a line coming from the same connecting block or one associated with it.

To propagate a signal label, create a signal label starting with the "<" character on the output of one of the listed connection blocks. When you run the simulation or update the diagram, the actual signal label appears, enclosed within angle brackets. The actual signal label is obtained by tracing back through the connection blocks until a signal label is encountered.

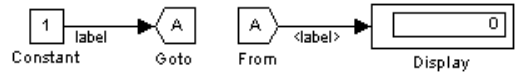
This example shows a model with a signal label and the propagated label both before and after updating the block diagram. In the first figure, the signal entering the Goto block is labeled `label` and the signal leaving the associated From block is labeled with a single <. The second figure shows the same model after choosing **Update Diagram** from the **Edit** menu.

1



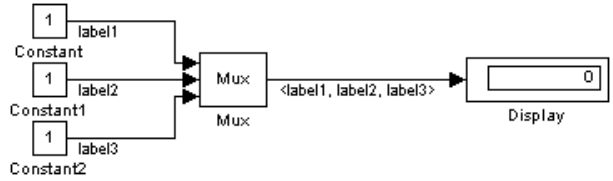
The signal label and propagated label before updating the diagram.

2



The same signal labels after updating the diagram.

In the next example, the propagated signal label shows the contents of a vector signal. This figure shows the label only *after* updating the diagram.

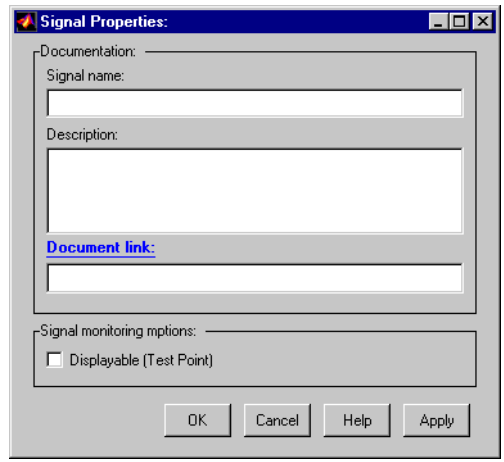


Setting Signal Properties

Signals have properties. Use Simulink's **Signal Properties** dialog box to view or set a signal's properties. To display the dialog box, select the line that carries the signal and choose **Signal Properties** from the Simulink **Edit** menu.

Signal Properties Dialog Box

The **Signal Properties** dialog box allows you to view and edit signal properties.



The dialog box includes the following controls.

Signal Name

Name of signal.

Description

Enter a description of the signal in this field.

Document Link

Enter a MATLAB expression in the field that displays documentation for the signal. To display the documentation, click the field's label (that is, "Document Link"). For example, entering the expression

```
web(['file:///' which('foo_signal.html')])
```

in the field causes MATLAB's default Web browser to display `foo_signal.html` when you click the field's label.

Displayable (Test Point)

Check this option to indicate that the signal can be displayed during simulation.

Note The next two controls are used to set properties used by the Real-Time Workshop to generate code from the model. You can ignore them if you do not plan to generate code from the model.

RTW storage class

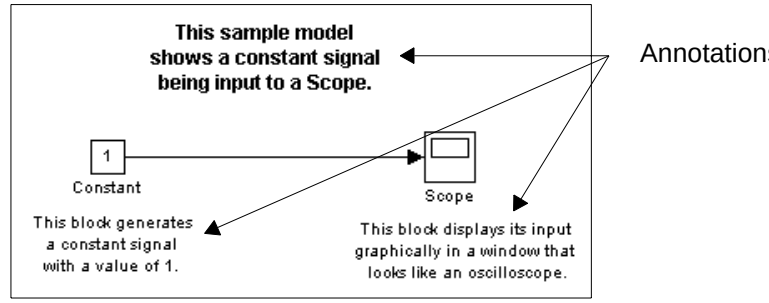
Select the storage class of this signal from the list. See the *Real-Time Workshop User's Guide* for an explanation of the listed options.

RTW storage type qualifier

Select the storage type of this signal from the list. See the *Real-Time Workshop User's Guide* for more information.

Annotations

Annotations provide textual information about a model. You can add an annotation to any unoccupied area of your block diagram.



To create a model annotation, double-click on an unoccupied area of the block diagram. A small rectangle appears and the cursor changes to an insertion point. Start typing the annotation contents. Each line is centered within the rectangle that surrounds the annotation.

To move an annotation, drag it to a new location.

To edit an annotation, select it:

- To replace the annotation on a Microsoft Windows or UNIX system, click on the annotation, then double-click or drag the cursor to select it. Then, enter the new annotation.
- To insert characters, click between two characters to position the insertion point, then insert text.
- To replace characters, drag the mouse to select a range of text to replace, then enter the new text.

To delete an annotation, hold down the **Shift** key while you select the annotation, then press the **Delete** or **Backspace** key.

To change the font of all or part of an annotation, select the text in the annotation you want to change, then choose **Font** from the **Format** menu. Select a font and size from the dialog box.

Working with Data Types

The term *data type* refers to the way in which a computer represents numbers in memory. A data type determines the amount of storage allocated to a number, the method used to encode the number's value as a pattern of binary digits, and the operations available for manipulating the type. Most computers provide a choice of data types for representing numbers, each with specific advantages in the areas of precision, dynamic range, performance, and memory usage. To enable you to take advantage of data typing to optimize the performance of MATLAB programs, MATLAB allows you to specify the data type of MATLAB variables. Simulink builds on this capability by allowing you to specify the data types of Simulink signals and block parameters.

The ability to specify the data types of a model's signals and block parameters is particularly useful in real-time control applications. For example, it allows a Simulink model to specify the optimal data types to use to represent signals and block parameters in code generated from a model by automatic code-generation tools, such as the Real-Time Workshop available from The MathWorks. By choosing the most appropriate data types for your model's signals and parameters, you can dramatically increase the performance and decrease the size of the code generated from the model.

Simulink performs extensive checking before and during a simulation to ensure that your model is *typesafe*, that is, that code generated from the model will not overflow or underflow and thus produce incorrect results. Simulink models that use Simulink's default data type (`double`) are inherently typesafe. Thus, if you never plan to generate code from your model or use a nondefault data type in your models, you can skip the remainder of this section.

On the other hand, if you plan to generate code from your models and use nondefault data types, read the remainder of this section carefully, especially the section on data type rules (see "Data Typing Rules" on page 3-44). In that way, you can avoid introducing data type errors that prevent your model from running to completion or simulating at all.

Data Types Supported by Simulink

Simulink supports all built-in MATLAB data types. The term *built-in data type* refers to data types defined by MATLAB itself as opposed to data types defined by MATLAB users. Unless otherwise specified, the term data type in the

Simulink documentation refers to built-in data types. The following table lists MATLAB's built-in data types.

Name	Description
`double`	Double-precision floating point
`single`	Single-precision floating point
`int8`	Signed eight-bit integer
`uint8`	Unsigned eight-bit integer
`int16`	Signed 16-bit integer
`uint16`	Unsigned 16-bit integer
`int32`	Signed 32-bit integer
`uint32`	Unsigned 32-bit integer

Besides the built-in types, Simulink defines a `boolean` (1 or 0) type, instances of which are represented internally by `uint8` values.

Block Support for Data and Numeric Signal Types

All Simulink blocks accept signals of type `double` by default. Some blocks prefer `boolean` input and others support multiple data types on their inputs. The following table lists Simulink blocks that prefer `boolean` or support multiple data types. The table also lists blocks that support complex signals.

Block	Comments
Abs	Inputs a real or complex signal of type `double`. Outputs a real signal of type `double`.
Combinatorial Logic	Input and output data type is `boolean`, if Boolean mode is enabled (see "Enabling Strict Boolean Type Checking" on page 3-45); otherwise, `double`.
Constant	Outputs a real or complex signal of any data type.

Block	Comments
Data Type Conversion	Inputs and outputs any real or complex data type.
Demux	Accepts mixed-type signal vectors.
Display	Accepts signals of any complex or real data type.
Dot Product	Inputs and outputs real or complex values of type `double`.
Enable	The corresponding subsystem enable port accepts signals of type `boolean` or `double`.
From	Outputs the data type (or types) of the signal connected to the corresponding Goto block.
From Workspace	Outputs type of corresponding workspace values.
Gain	Input can be a real- or complex-valued signal or vector of any data type except `boolean`.
Goto	Input can be of any type.
Ground	Outputs a 0 signal of the same type as the port to which it is connected.
Hit Crossing	Inputs a `double` signal. Outputs `boolean`, if Boolean mode is enabled (see "Enabling Strict Boolean Type Checking" on page 3-45); otherwise, `double`.
Inport	An inport accepts real- or complex-valued signals of any data type. The elements of an input signal vector must be of the same type if the inport is a root-level inport or the inport is directly connected to an outport of the same subsystem.
Integrator	An Integrator block accepts and outputs signals of type `double` on its data ports. Its external reset port accepts signals of type `double` or `boolean`.

Block	Comments
Logical Operator	Inputs and outputs real signals of type `boolean`, if Boolean mode is enabled (see "Enabling Strict Boolean Type Checking" on page 3-45); otherwise, real signals of type `double`.
Manual Switch	Accepts real- or complex-valued signals of any type. All inputs must have the same signal and data type.
Math Function	Inputs and outputs real or complex values of type `double`.
MATLAB Function	Inputs and outputs real or complex values of type `double`.
Memory	Inputs real or complex signals of any data type.
Merge	Inputs and outputs any real or complex data type.
Multiport Switch	The control input of a Multiport Switch block accepts a real-valued signal of any type except `boolean`. The other inputs accept real- or complex-valued inputs of any type. All inputs must of the same data and numeric type. The block outputs the type of signal on its inputs.
Mux	Accepts any supported Simulink data type, including mixed-type vectors, on each input.
Out	Accepts any Simulink data type as input. Accepts mixed-type vectors as input, if the outport is in a subsystem and no initial condition is specified.
Product	Accepts real- or complex-valued signals of any data type except `boolean`. All inputs must be of the same data type.
Relational Operator	Accepts any supported data type as inputs. Both inputs must be of the same type. Outputs `boolean`, if Boolean mode is enabled (see "Enabling Strict Boolean Type Checking" on page 3-45); otherwise, `double`.

Block	Comments
Rounding Function	Accepts and outputs real or complex values of type `double`.
Scope	Accepts real or complex signals of any data type.
Selector	Outputs the data types of the selected input signals.
Sum	Accepts any Simulink data type as input. All inputs must be of the same type. Outputs the same type as the input.
Switch	Accepts real- or complex-valued signals of any data type as switched inputs (inputs 1 and 3). Both switched inputs must be of the same type. The block output signal has the data type of the input. The data type of the threshold input must be `boolean` or `double`.
Terminator	Accepts any Simulink type.
To Workspace	Accepts any Simulink data type as input.
Trigger	The corresponding subsystem control port accepts signals of type `boolean` or `double`.
Trigonometric Function	Inputs and outputs real- or complex-valued signals of type `double`.
Unit Delay	Accepts and outputs real- or complex-valued signals of any data type.
Width	Accepts real- or complex-valued signals of any data type, including mixed-type signal vectors.
Zero-Order Hold	Accepts any Simulink data type as input.

See Chapter 8, “Block Reference” for more information on the data types supported by specific blocks for parameter and input and output values. If the documentation for a block does not specify a data type, the block inputs or outputs only data of type `double`.

Specifying Block Parameter Data Types

When entering block parameters whose data type is user-specifiable, use the syntax

```
type(value)
```

to specify the parameter, where type is the name of the data type and value is the parameter value. The following examples illustrate this syntax.

`single(1.0)`	Specifies a single-precision value of 1.0
`int8(2)`	Specifies an eight-bit integer of value 2
`int32(3+2i)`	Specifies a complex value whose real and imaginary parts are 32-bit integers

Creating Signals of a Specific Data Type

You can introduce a signal of a specific data type into a model in any of the following ways:

- Load signal data of the desired type from the MATLAB workspace into your model via a root-level inport or a From Workspace block.
- Create a Constant block in your model and set its parameter to the desired type.
- Use a Data Type Conversion block to convert a signal to the desired data type.

Displaying Port Data Types

To display the data types of ports in your model, select **Port Data Types** from Simulink's **Format** menu. Simulink does not update the port data type display when you change the data type of a diagram element. To refresh the display, type **Ctrl-D**.

Data Type Propagation

Whenever you start a simulation, enable display of port data types, or refresh the port data type display, Simulink performs a processing step called data type propagation. This step involves determining the types of signals whose

type is not otherwise specified and checking the types of signals and input ports to ensure that they do not conflict. If type conflicts arise, Simulink displays an error dialog that specifies the signal and port whose data types conflict. Simulink also highlights the signal path that creates the type conflict.

Note You can insert typecasting (data type conversion) blocks in your model to resolve type conflicts. See"Typecasting Signals" on page 3-45 for more information.

Data Typing Rules

Observing the following rules will help you to create models that are typesafe and therefore execute without error:

- Signal data types generally do not affect parameter data types, and vice versa.

  A significant exception to this rule is the Constant block whose output data type is determined by the data type of its parameter.
- If the output of a block is a function of an input and a parameter and the input and parameter differ in type, Simulink converts the parameter to the input type before computing the output.

  See "Typecasting Parameters" on page 3-45 for more information.
- In general, a block outputs the data type that appears at its inputs.

  Significant exceptions include constant blocks and data type conversion blocks whose output data types are determined by block parameters.
- Virtual blocks accept signals of any type on their inputs.

  Examples of virtual blocks include Mux and Demux blocks and unconditionally executed subsystems.
- The elements of a signal vector connected to a port of a nonvirtual block must be of the same data type.
- The signals connected to the input data ports of a nonvirtual block cannot differ in type.
- Control ports (for example, Enable and Trigger ports) accept `boolean` or `double` signals.

- Solver blocks accept only `double` signals.
- Connecting a non`double` signal to a block disables zero-crossing detection for that block.

Enabling Strict Boolean Type Checking

By default, Simulink detects but does not signal an error when it detects that double signals are connected to block that prefer `boolean` input. This ensures compatibility with models created by earlier versions of Simulink that support only `double` data type. You can enable strict boolean type checking by unchecking the **Relax boolean type checking** option on the **Diagnostics** page of the **Simulation Parameters** dialog box (see "The Diagnostics Page" on page 4-24).

Typecasting Signals

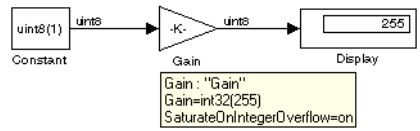
Simulink signals an error whenever it detects that a signal is connected to a block that does not accept the signal's data type. If you want to create such a connection, you must explicitly typecast (convert) the signal to a type that the block does accept. You can use Simulink's Data Type Conversion block to perform such conversions (see `Data Type Conversion` on page 8-41).

Typecasting Parameters

In general, during simulation, Simulink silently converts parameter data types to signal data types (if they differ) when computing block outputs that are a function of an input signal and a parameter. The following exceptions occur to this rule:

- If the signal data type cannot represent the parameter value, Simulink halts the simulation and signals an error.

  Consider, for example, the following model.



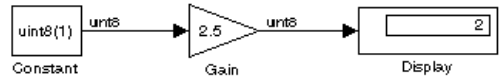
  This model uses a Gain block to amplify a constant input signal. Computing the output of the Gain block requires computing the product of the input

signal and the gain. Such a computation requires that the two values be of the same data type. However, in this case, the data type of the signal, `uint8` (unsigned 8-bit word), differs from the data type of the gain parameter, int32 (signed 32-bit integer). Thus computing the output of the gain block entails a type conversion.

When making such conversions, Simulink always casts the parameter type to the signal type. Thus, in this case, Simulink must convert the Gain block's gain value to the data type of the input signal. Simulink can make this conversion only if the input signal's data type (uint8) can represent the gain. In this case, Simulink can make the conversion because the gain is 255, which is within the range of the uint8 data type (0 to 255). Thus, this model simulates without error. However, if the gain were slightly larger (for example, 256), Simulink would signal an out-of-range error if you attempted to simulate the model.

- If the signal data type can represent the parameter value but only at reduced precision, Simulink issues a warning message and continues the simulation.

  Consider, for example, the following model.



  In this example, the signal type accommodates only integer values while the gain value has a fractional component. Simulating this model causes Simulink to truncate the gain to the nearest integral value (2) and issue a loss-of-precision warning. On the other hand, if the gain were 2.0, Simulink would simulate the model without complaint because in this case the conversion entails no loss of precision.

Note Conversion of an `int32` parameter to a `float` or `double` can entail a loss of precision. The loss can be severe if the magnitude of the parameter value is large. If an `int32` parameter conversion does entail a loss of precision, Simulink issues a warning message.

Working with Complex Signals

By default, the values of Simulink signals are real numbers. However, models can create and manipulates signals that have complex numbers as values.

You can introduce a complex-valued signal into a model in any of the following ways:

- Load complex-valued signal data from the MATLAB workspace into the model via a root-level inport.
- Create a Constant block in your model and set its value to a complex number.
- Create real signals corresponding to the real and imaginary parts of a complex signal and then combine the parts into a complex signal, using Real-Imag to Complex conversion block.

You can manipulate complex signals via blocks that accept them. Most Simulink blocks accept complex signals as input. If you are not sure whether a block accepts complex signals, refer to the documentation for the block in Chapter 8, “Block Reference.”

Summary of Mouse and Keyboard Actions

These tables summarize the use of the mouse and keyboard to manipulate blocks, lines, and signal labels. LMB means press the left mouse button; CMB, the center mouse button; and RMB, the right mouse button.

The first table lists mouse and keyboard actions that apply to blocks.

Table 3-2: Manipulating Blocks

Task	Microsoft Windows	UNIX
Select one block	LMB	LMB
Select multiple blocks	**Shift** + LMB	**Shift** + LMB; or CMB alone
Copy block from another window	Drag block	Drag block
Move block	Drag block	Drag block
Duplicate block	**Ctrl** + LMB and drag; or RMB and drag	**Ctrl** + LMB and drag; or RMB and drag
Connect blocks	LMB	LMB
Disconnect block	**Shift** + drag block	**Shift** + drag block; or CMB and drag

The next table lists mouse and keyboard actions that apply to lines.

Table 3-3: Manipulating Lines

Task	Microsoft Windows	UNIX
Select one line	LMB	LMB
Select multiple lines	**Shift** + LMB	**Shift** + LMB; or CMB alone
Draw branch line	**Ctrl** + drag line; or **RMB** and drag line	**Ctrl** + drag line; or RMB + drag line

Table 3-3: Manipulating Lines (Continued)

Task	Microsoft Windows	UNIX
Route lines around blocks	**Shift** + draw line segments	**Shift** + draw line segments; or CMB and draw segments
Move line segment	Drag segment	Drag segment
Move vertex	Drag vertex	Drag vertex
Create line segments	**Shift** + drag line	**Shift** + drag line; or CMB + drag line

The next table lists mouse and keyboard actions that apply to signal labels.

Table 3-4: Manipulating Signal Labels

Action	Microsoft Windows	UNIX
Create signal label	Double-click on line, then type label	Double-click on line, then type label
Copy signal label	**Ctrl** + drag label	**Ctrl** + drag label
Move signal label	Drag label	Drag label
Edit signal label	Click in label, then edit	Click in label, then edit
Delete signal label	**Shift** + click on label, then press **Delete**	**Shift** + click on label, then press **Delete**

The next table lists mouse and keyboard actions that apply to annotations.

Table 3-5: Manipulating Annotations

Action	Microsoft Windows	UNIX
Create annotation	Double-click in diagram, then type text	Double-click in diagram, then type text
Copy annotation	**Ctrl** + drag label	**Ctrl** + drag label

Table 3-5: Manipulating Annotations (Continued)

Action	Microsoft Windows	UNIX
Move annotation	Drag label	Drag label
Edit annotation	Click in text, then edit	Click in text, then edit
Delete annotation	**Shift** + select annotation, then press **Delete**	**Shift** + select annotation, then press **Delete**

Creating Subsystems

As your model increases in size and complexity, you can simplify it by grouping blocks into subsystems. Using subsystems has these advantages:

- It helps reduce the number of blocks displayed in your model window.
- It allows you to keep functionally related blocks together.
- It enables you to establish a hierarchical block diagram, where a Subsystem block is on one layer and the blocks that make up the subsystem are on another.

You can create a subsystem in two ways:

- Add a Subsystem block to your model, then open that block and add the blocks it contains to the subsystem window.
- Add the blocks that make up the subsystem, then group those blocks into a subsystem.

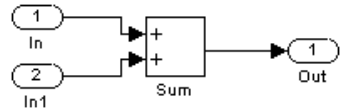
Creating a Subsystem by Adding the Subsystem Block

To create a subsystem before adding the blocks it contains, add a Subsystem block to the model, then add the blocks that make up the subsystem:

1. Copy the Subsystem block from the Signals & Systems library into your model.

2. Open the Subsystem block by double-clicking on it.

3. In the empty Subsystem window, create the subsystem. Use Inport blocks to represent input from outside the subsystem and Outport blocks to represent external output. For example, the subsystem below includes a Sum block and Inport and Outport blocks to represent input to and output from the subsystem:

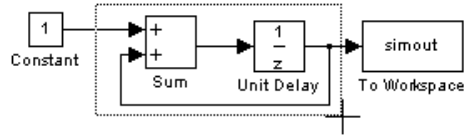


Creating a Subsystem by Grouping Existing Blocks

If your model already contains the blocks you want to convert to a subsystem, you can create the subsystem by grouping those blocks:

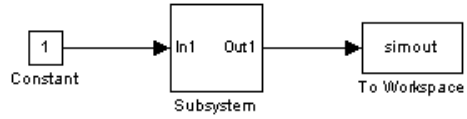
1. Enclose the blocks and connecting lines that you want to include in the subsystem within a bounding box. You cannot specify the blocks to be grouped by selecting them individually or by using the **Select All** command. For more information, see "Selecting Multiple Objects Using a Bounding Box" on page 3–7.

   For example, this figure shows a model that represents a counter. The Sum and Unit Delay blocks are selected within a bounding box.

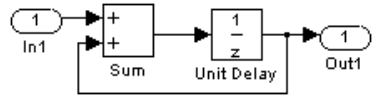


   When you release the mouse button, the two blocks and all the connecting lines are selected.

2. Choose **Create Subsystem** from the **Edit** menu. Simulink replaces the selected blocks with a Subsystem block. This figure shows the model after choosing the **Create Subsystem** command (and resizing the Subsystem block so the port labels are readable).



If you open the Subsystem block, Simulink displays the underlying system, as shown below. Notice that Simulink adds Inport and Outport blocks to represent input from and output to blocks outside the subsystem.



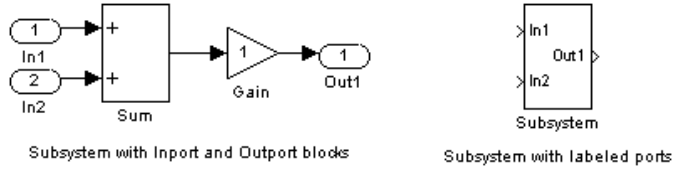
As with all blocks, you can change the name of the Subsystem block. Also, you can customize the icon and dialog box for the block using the masking feature, described in Chapter 6.

Labeling Subsystem Ports

Simulink labels ports on a Subsystem block. The labels are the names of Inport and Outport blocks that connect the subsystem to blocks outside the subsystem through these ports.

You can hide the port labels by selecting the Subsystem block, then choosing **Hide Port Labels** from the **Format** menu. You can also hide one or more port labels by selecting the appropriate Inport or Outport block in the subsystem and choosing **Hide Name** from the **Format** menu.

This figure shows two models. The subsystem on the left contains two Inport blocks and one Outport block. The Subsystem block on the right shows the labeled ports.



Using Callback Routines

You can define MATLAB expressions that execute when the block diagram or a block is acted upon in a particular way. These expressions, called *callback routines*, are associated with block or model parameters. For example, the callback associated with a block's `OpenFcn` parameter is executed when the model user double-clicks on that block's name or path changes.

To define callback routines and associate them with parameters, use the `set_param` command (see `set_param` on page 10-24).

For example, this command evaluates the variable `testvar` when the user double-clicks on the Test block in `mymodel`:

```
set_param('mymodel/Test', 'OpenFcn', testvar)
```

You can examine the `clutch` system (`clutch.mdl`) for routines associated with many model callbacks.

These tables list the parameters for which you can define callback routines, and indicate when those callback routines are executed. Routines that are executed before or after actions take place occur immediately before or after the action.

Table 3-6: Model Callback Parameters

Parameter	When Executed
`CloseFcn`	Before the block diagram is closed.
`PostLoadFcn`	After the model is loaded. Defining a callback routine for this parameter might be useful for generating an interface that requires that the model has already been loaded.
`InitFcn`	Called at start of model simulation.
`PostSaveFcn`	After the model is saved.
`PreLoadFcn`	Before the model is loaded. Defining a callback routine for this parameter might be useful for loading variables used by the model.
`PreSaveFcn`	Before the model is saved.
`StartFcn`	Before the simulation starts.
`StopFcn`	After the simulation stops. Output is written to workspace variables and files before the `StopFcn` is executed.

Table 3-7: Block Callback Parameters

Parameter	When Executed
`CloseFcn`	When the block is closed using the `close_system` command.
`CopyFcn`	After a block is copied. The callback is recursive for Subsystem blocks (that is, if you copy a Subsystem block that contains a block for which the `CopyFcn` parameter is defined, the routine is also executed). The routine is also executed if an `add_block` command is used to copy the block.
`DeleteFcn`	Before a block is deleted. This callback is recursive for Subsystem blocks.
`DestroyFcn`	When block has been destroyed.
`InitFcn`	Before the block diagram is compiled and before block parameters are evaluated.
`LoadFcn`	After the block diagram is loaded. This callback is recursive for Subsystem blocks.
`ModelCloseFcn`	Before the block diagram is closed. This callback is recursive for Subsystem blocks.
`MoveFcn`	When block is moved or resized.
`NameChangeFcn`	After a block's name and/or path changes. When a Subsystem block's path is changed, it recursively calls this function for all blocks it contains after calling its own `NameChangeFcn` routine.

Table 3-7: Block Callback Parameters (Continued)

Parameter	When Executed
`OpenFcn`	When the block is opened. This parameter is generally used with Subsystem blocks. The routine is executed when you double-click on the block or when an `open_system` command is called with the block as an argument. The `OpenFcn` parameter overrides the normal behavior associated with opening a block, which is to display the block's dialog box or to open the subsystem.
`ParentCloseFcn`	Before closing a subsystem containing the block or when the block is made part of a new subsystem using the new_system command (see `new_system` on page 10-19).
`PreSaveFcn`	Before the block diagram is saved. This callback is recursive for Subsystem blocks.
`PostSaveFcn`	After the block diagram is saved. This callback is recursive for Subsystem blocks.
`StartFcn`	After the block diagram is compiled and before the simulation starts.
`StopFcn`	At any termination of the simulation.
`UndoDeleteFcn`	When a block delete is undone.

Tips for Building Models

Here are some model-building hints you might find useful:

- Memory issues

  In general, the more memory, the better Simulink performs.

- Using hierarchy

  More complex models often benefit from adding the hierarchy of subsystems to the model. Grouping blocks simplifies the top level of the model and can make it easier to read and understand the model. For more information, see "Creating Subsystems" on page 3–51. The Model Browser (see "The Model Browser" on page 3-66) provides useful information about complex models.

- Cleaning up models

  Well organized and documented models are easier to read and understand. Signal labels and model annotations can help describe what is happening in a model. For more information, see "Signal Labels" on page 3–32 and "Annotations" on page 3–37.

- Modeling strategies

  If several of your models tend to use the same blocks, you might find it easier to save these blocks in a model. Then, when you build new models, just open this model and copy the commonly used blocks from it. You can create a block library by placing a collection of blocks into a system and saving the system. You can then access the system by typing its name in the MATLAB command window.

  Generally, when building a model, design it first on paper, then build it using the computer. Then, when you start putting the blocks together into a model, add the blocks to the model window before adding the lines that connect them. This way, you can reduce how often you need to open block libraries.

Modeling Equations

One of the most confusing issues for new Simulink users is how to model equations. Here are some examples that may improve your understanding of how to model equations.

Converting Celsius to Fahrenheit

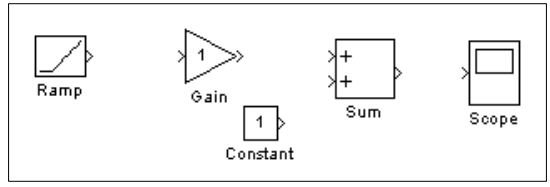
To model the equation that converts Celsius temperature to Fahrenheit:

$$T_F = 9/5(T_C) + 32$$

First, consider the blocks needed to build the model:

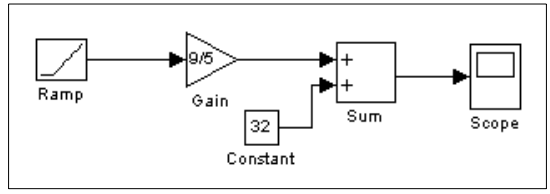
- A Ramp block to input the temperature signal, from the Sources library
- A Constant block, to define a constant of 32, also from the Sources library
- A Gain block, to multiply the input signal by 9/5, from the Math library
- A Sum block, to add the two quantities, also from the Math library
- A Scope block to display the output, from the Sinks library

Next, gather the blocks into your model window.



Assign parameter values to the Gain and Constant blocks by opening (double-clicking on) each block and entering the appropriate value. Then, click on the **Close** button to apply the value and close the dialog box.

Now, connect the blocks.



The Ramp block inputs Celsius temperature. Open that block and change the **Initial output** parameter to 0. The Gain block multiplies that temperature by the constant 9/5. The Sum block adds the value 32 to the result and outputs the Fahrenheit temperature.

Open the Scope block to view the output. Now, choose **Start** from the **Simulation** menu to run the simulation. The simulation will run for 10 seconds.

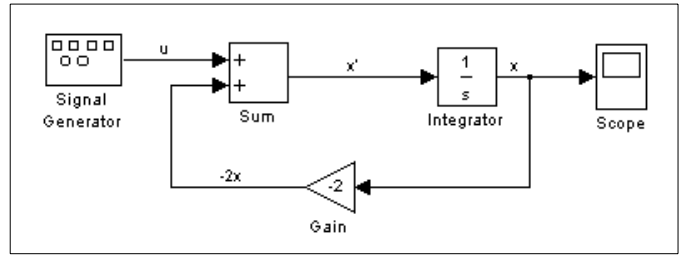
Modeling a Simple Continuous System

To model the differential equation

$$x'(t) = -2x(t) + u(t)$$

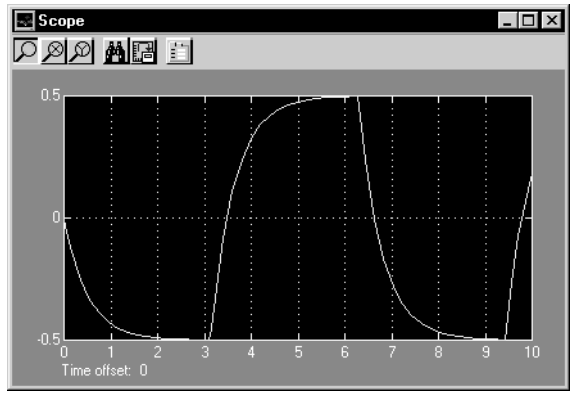
where *u(t)* is a square wave with an amplitude of 1 and a frequency of 1 rad/sec. The Integrator block integrates its input, x', to produce x. Other blocks needed in this model include a Gain block and a Sum block. To generate a square wave, use a Signal Generator block and select the Square Wave form but change the default units to radians/sec. Again, view the output using a Scope block. Gather the blocks and define the gain.

In this model, to reverse the direction of the Gain block, select the block, then use the **Flip Block** command from the **Format** menu. Also, to create the branch line from the output of the Integrator block to the Gain block, hold down the **Ctrl** key while drawing the line. For more information, see "Drawing a Branch Line" on page 3–28. Now you can connect all the blocks.



An important concept in this model is the loop that includes the Sum block, the Integrator block, and the Gain block. In this equation, x is the output of the Integrator block. It is also the input to the blocks that compute x', on which it is based. This relationship is implemented using a loop.

The Scope displays x at each time step. For a simulation lasting 10 seconds, the output looks like this.



The equation you modeled in this example can also be expressed as a transfer function. The model uses the Transfer Fcn block, which accepts u as input and outputs x. So, the block implements x/u. If you substitute sx for x' in the equation above.

$$sx = -2x + u$$

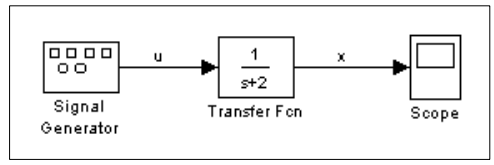
Solving for x gives

$$x = u/(s + 2)$$

Or,

$$x/u = 1/(s + 2)$$

The Transfer Fcn block uses parameters to specify the numerator and denominator coefficients. In this case, the numerator is `1` and the denominator is `s+2`. Specify both terms as vectors of coefficients of successively decreasing powers of `s`. In this case the numerator is `[1]` (or just `1`) and the denominator is `[1 2]`. The model now becomes quite simple:



The results of this simulation are identical to those of the previous model.

Saving a Model

You can save a model by choosing either the **Save** or **Save As** command from the **File** menu. Simulink saves the model by generating a specially formatted file called the *model file* (with the `.mdl` extension) that contains the block diagram and block properties. The format of the model file is described in Appendix B.

If you are saving a model for the first time, use the **Save** command to provide a name and location to the model file. Model file names must start with a letter and can contain no more than 31 letters, numbers, and underscores.

If you are saving a model whose model file was previously saved, use the **Save** command to replace the file's contents or the **Save As** command to save the model with a new name or location.

Simulink follows this procedure while saving a model:

1. If the `mdl` file for the model already exists, it is renamed as a temporary file.
2. Simulink executes all block `PreSaveFcn` callback routines, then executes the block diagram's `PreSaveFcn` callback routine.
3. Simulink writes the model file to a new file using the same name and an extension of `mdl`.
4. Simulink executes all block `PostSaveFcn` callback routines, then executes the block diagram's `PostSaveFcn` callback routine.
5. Simulink deletes the temporary file.

If an error occurs during this process, Simulink renames the temporary file to the name of the original model file, writes the current version of the model to a file with an `.err` extension, and issues an error message. Simulink performs steps 2 through 4 even if an error occurs in an earlier step.

Printing a Block Diagram

You can print a block diagram by selecting **Print** from the **File** menu (on a Microsoft Windows system) or by using the `print` command in the MATLAB command window (on all platforms).

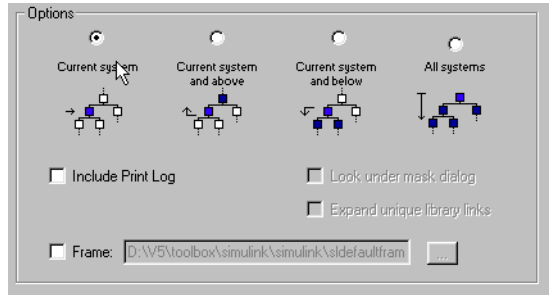
On a Microsoft Windows system, the **Print** menu item prints the block diagram in the current window.

Print Dialog Box

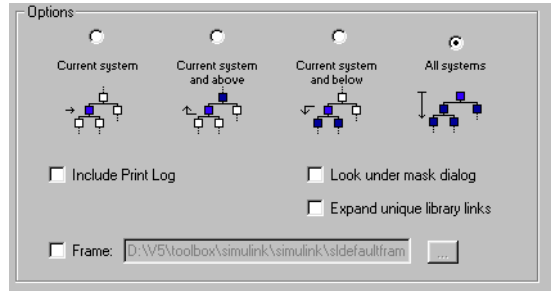
When you select the **Print** menu item, the **Print** dialog box appears. The **Print** dialog box enables you to selectively print systems within your model. Using the dialog box, you can:

- 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
- Print an overlay frame on each diagram

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.



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.



Selecting 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.

Selecting 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 3-21.

The print log lists the blocks and systems printed. To print the print log, select the **Include Print Log** check box.

Selecting the **Frame** check box prints a title block frame on each diagram. Enter the path to the title block frame in the adjacent edit box. You can create a customized title block frame, using MATLAB’s frame editor. See `frameedit` in the online MATLAB reference for information on using the frame editor to create title block frames.

Print Command

The format of the `print` command is

```
print –ssys –device filename
```

`sys` is the name of the system to be printed. The system name must be preceded by the `s` switch identifier and is the only required argument. *`sys`* must be open or must have been open during the current session. If the system name contains spaces or takes more than one line, you need to specify the name as a string. See the examples below.

device specifies a device type. For a list and description of device types, see *Using MATLAB Graphics*.

filename is the PostScript file to which the output is saved. If `filename` exists, it is replaced. If `filename` does not include an extension, an appropriate one is appended.

For example, this command prints a system named `untitled`.

```
print –suntitled
```

This command prints the contents of a subsystem named `Sub1` in the current system.

```
print –sSub1
```

This command prints the contents of a subsystem named `Requisite Friction`.

```
print (['–sRequisite Friction'])
```

The next example prints a system named `Friction Model`, a subsystem whose name appears on two lines. The first command assigns the newline character to a variable; the second prints the system.

```
cr = sprintf('\n');
print (['–sFriction' cr 'Model'])
```

Specifying Paper Size and Orientation

Simulink lets you specify the type and orientation of the paper used to print a model diagram. You can do this on all platforms by setting the model's `PaperType` and `PaperOrientation` properties, respectively (see "Model Parameters" on page A-3), using the `set_param` command. You can set the paper orientation alone, using MATLAB's `orient` command. On Windows, the **Print** dialog box lets you set the page type and orientation properties as well.

Positioning and Sizing a Diagram

You can use a model's `PaperPositionMode` and `PaperPosition` parameters to position and size the model's diagram on the printed page. The value of the `PaperPosition` parameter is a vector of form `[left bottom width height]`. The first two elements specify the bottom left corner of a rectangular area on the page, measured from the page's bottom left corner. The last two elements specify the width and height of the rectangle. When the model's

`PaperPositionMode` is manual, Simulink positions (and scales, if necessary) the model's diagram to fit inside the specified print rectangle. For example, the following commands

```
vdp
set_param('vdp', 'PaperType', 'usletter')
set_param('vdp', 'PaperOrientation', 'landscape')
set_param('vdp', 'PaperPositionMode', 'manual')
set_param('vdp', 'PaperPosition', [0.5 0.5 4 4])
print -svdp
```

print the block diagram of the `vdp` sample model in the lower left corner of a U.S. letter-size page in landscape orientation.

If `PaperPositionMode` is `auto`, Simulink centers the model diagram on the printed page, scaling the diagram, if necessary, to fit the page.

The Model Browser

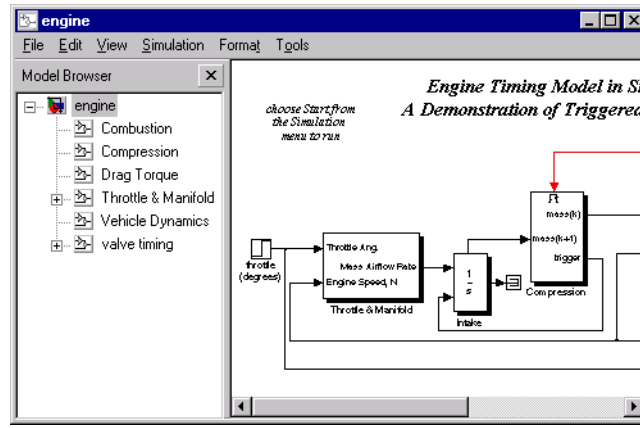
The Model Browser enables you to:

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

The browser operates differently on Microsoft Windows and UNIX platforms.

Using the Model Browser on Windows

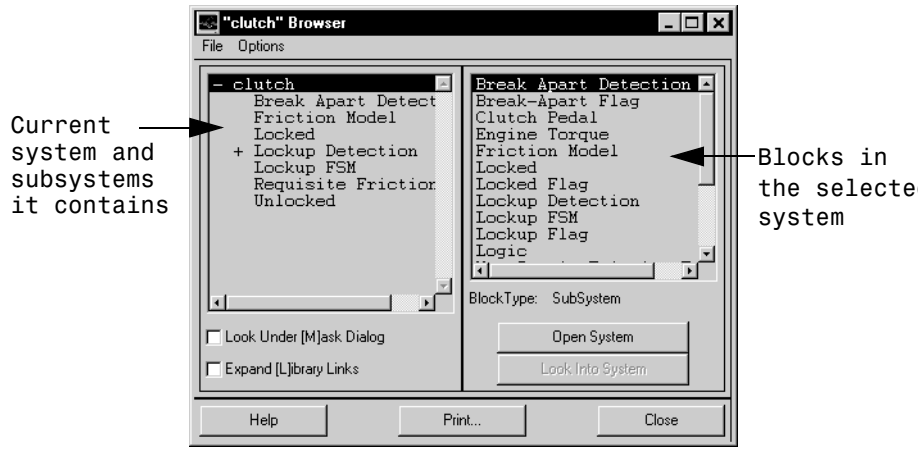
To display the Model Browser pane, select **Model Browser** from the Simulink **View** menu. The model window splits into two panes. The left pane displays the browser, a tree-structured view of the block diagram displayed in the right pane.



Each entry in the tree view corresponds to a subsystem in the model. You can expand/collapse the tree by clicking on the +/- boxes beside each subsystem, or by pressing the left/right arrow or +/- keys on your numeric keypad. You can move up/down the tree by pressing the up/down arrow on your keypad. Click on any subsystem to display its contents in the diagram view. To open a new window on a subsystem, double click the subsystem in the diagram view.

Using the Model Browser on UNIX

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



Contents of the Browser Window

The Model 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. For more information about masking, see Chapter 6.
- [L] indicates that the block is a reference block. For more information, see "Libraries" on page 3-21.
- [O] indicates that an open function (`OpenFcn`) callback is defined for the block. For more information about block callbacks, see "Using Callback Routines" on page 3-53.
- [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 Model 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 Model 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 Model 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 Model 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 Model Browser displays a minus sign (–) before their names. To display systems alphabetically, select the **Display Alphabetical List** menu item on the **Options** menu.

Tracking Model Versions

A Simulink model can go through many versions during its development. Simulink helps you to track the various versions by generating and storing version control information, including the version number, the persons who created and last updated the model, and optionally a change history. The following Simulink features allow you to manage and use the version control information:

- The Simulink **Model Parameters** dialog box allows you to edit some of the version control information stored in the model and to select various version control options.
- The Simulink Model Info block allows you to display version control information, including that maintained by an external version control system, as an annotation block in a model diagram.
- Simulink version control parameters allow you to access version control information from the MATLAB command line or an M-file.

Specifying the Current User

When a user creates or updates a model, Simulink logs the user's name in the model for version control purposes. Simulink assumes that the user's name is specified by at least one of the following environment variables: `USER`, `USERNAME`, `LOGIN`, or `LOGNAME`. If your system does not define any of these variables, Simulink does not update the user name in the model.

UNIX systems always define the `USER` environment variable and set its value to the name you use to log onto your system. Thus, if you are using a UNIX system, you do not have to do anything to enable Simulink to identify you as the current user. Windows systems, on the other hand, may define some or none of the "user name" environment variables that Simulink expects, depending on the version of Windows installed on your system and whether it operates stand-alone or connected to a network. You can use the MATLAB command `getenv` to determine which if any of the environment variables is defined. For example, enter

```
getenv('user')
```

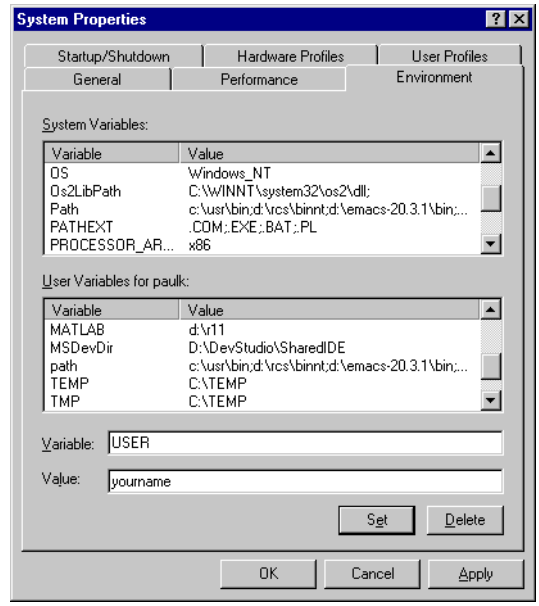
at the MATLAB command line to determine if the `USER` environment variable exists on your Windows system. If not, you must set it yourself. On Windows 95 and 98, set the value by entering the following line

```
set user=yourname
```

in your system's `autoexec.bat` file, where `yourname` is the name by which you want to be identified in a model file. Then, reboot your computer.

Note The `autoexec.bat` file typically is found in the `c:\` directory on your system's hard disk.

On Windows NT, use the **Environment** panel of the Windows NT **System Properties** dialog box to set the `USER` environment variable (if it is not already defined).

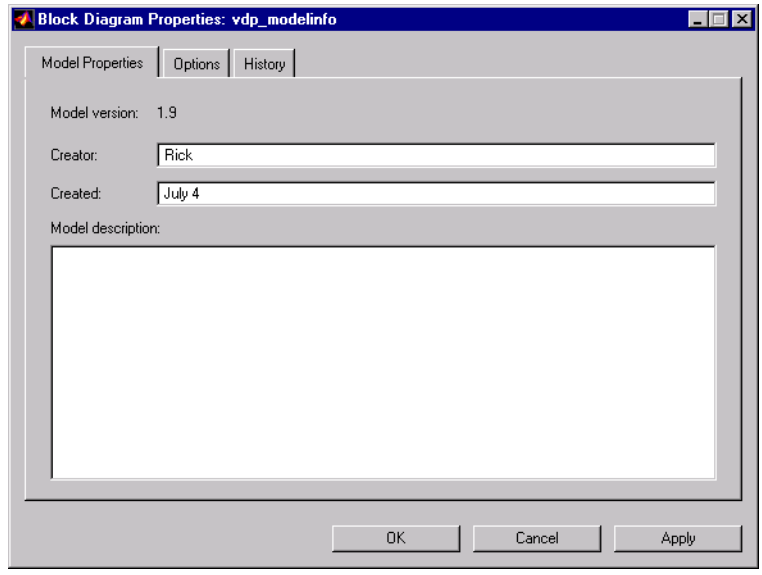


To display the **System Properties** dialog box, select **System** in your system's **Control Panel** folder, which resides in your system's **My Computer** folder, which resides on your Windows NT desktop. To set the `USER` variable, enter

USER in the **Variable** field, your login name in the **Value** field, and select the **Set** button. Then select **OK** to dismiss the dialog box.

Model Properties Dialog

The **Model Properties** dialog box allows you to edit some version control parameters and set some related options. To display the dialog box, choose **Model Parameters** from the Simulink **File** menu.



Model Properties Pane

The **Model Properties** pane lets you edit the following version control parameters.

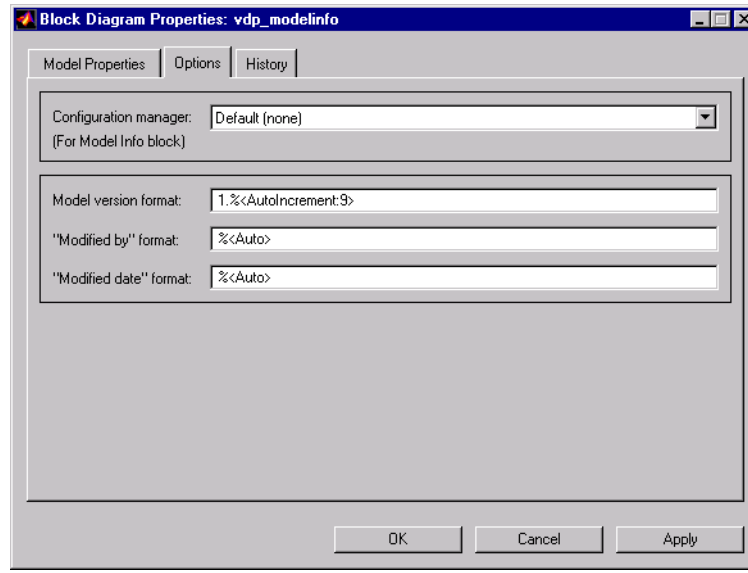
Creator. Name of the person who created this model. Simulink sets this property to the value of the USER environment variable when you create the model. Edit this field to change the value.

Created. Date and time this model was created.

Model description. Description of the model.

Options Pane

The Options pane lets you choose a configuration manager and specify version control information formats.



Configuration manager. External configuration manager used to manage this model. Choosing this option allows you to include information from the configuration manager in a Model Info annotation block. See `Model Info` on page 8-131 for more information.

The file `cmopts.m` in the `MATLABROOT/toolbox/local` directory specifies the default configuration manager for models. The default "default" configuration manager is `none`. You can edit this file to specify another choice.

Model version format. Format used to display the model version number in the **Model Parameters** pane and in Model Info blocks. The value of this parameter can be any text string. The text string can include occurrences of the tag `%<AutoIncrement:#>` where # is an integer. Simulink replaces the tag with # when displaying the model's version number. For example, it displays

```
1.%<AutoIncrement:2>
```

as

```
1.2
```

Simulink increments # by 1 when saving the model. For example,

```
1.%<1.%<AutoIncrement:2>
```

becomes

```
1.%<1.%<AutoIncrement:3>
```

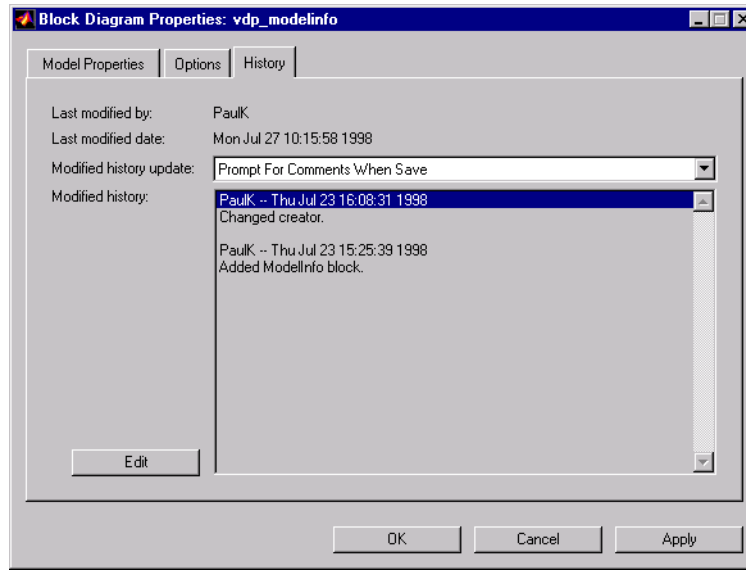
when you save the model.

Modified by format. Format used to display the "Modified By" value in the **History** pane, in the history log, and in Model Info blocks. The value of this field can be any string. The string can include the tag `%<Auto>`. Simulink replaces occurrences of this tag with the current value of the `USER` environment variable.

Modified date format. Format used to display the "Last modified date" in the **History** pane, in the history log, and in Model Info blocks. The value of this field can be any string. The string can contain the tag `%<Auto>`. Simulink replaces occurrences of this tag with the current date and time.

History Pane

The History pane allows you enable, view, and edit this model's change history.



Last modified by. Name of the person who last modified this model. Simulink sets the value of this parameter to the value of the `USER` environment variable when you save a model. You cannot edit this field.

Last modified date. Date that this model was last modified. Simulink sets the value of this parameter to the system date and time when you save a model. You cannot edit this field.

Modified history update. Specifies whether to prompt a user for a comment when this model is saved. If you choose "Prompt for Comments When Save," Simulink prompts you for a comment to store in the model. You would typically use the comment to document any changes you made to the model in the current session. Simulink stores the previous value of this parameter in the model's change history. See "Creating a Model Change History" on page 3–76 for more information.

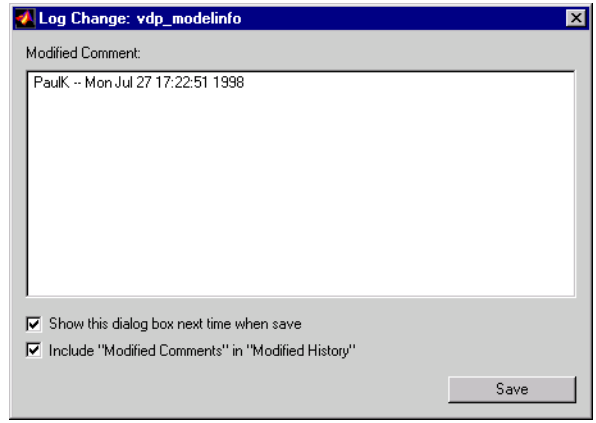
Modified history. History of modifications of this model. Simulink compiles the history from comments entered by users when they update the model. You can edit the history at any time by selecting the adjacent **Edit** button.

Creating a Model Change History

Simulink allows you to create and store a record of changes to a model in the model itself. Simulink compiles the history automatically from comments that you or other users enter when they save changes to a model.

Logging Changes

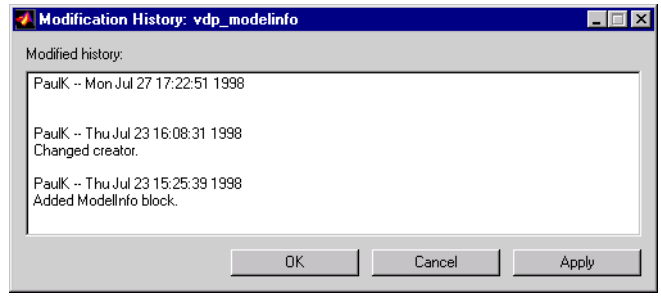
To start a change history, select the "Prompt for Comments When Save" option from the **History** pane on the Simulink **Model Properties** dialog box. The next time you save the model, Simulink displays a **Log Change** dialog box.



If you want to add an item to the model's change history, enter the item in the **Modified Comments** edit field and click the **Save** button. If you do not want to enter an item for this session, uncheck the **Include "Modified Contents" in "Modified History"** option. If you want to discontinue change logging, uncheck the **Show this dialog box next time when save** option.

Editing the Change History

To edit the change history for a model, click the **Edit** button on the Simulink **Model Properties** dialog box. Simulink displays the model's history in a **Modification History** dialog box.



Edit the history displayed in the dialog and select **Apply** or **OK** to save the changes.

Version Control Properties

Simulink stores version control information in a model as model parameters. You can access this information from the MATLAB command line or from an M-file, using the Simulink `get_param` command. The following table describes the model parameters used by Simulink to store version control information.

Property	Description
`Created`	Date created
`Creator`	Name of the person who created this model
`ModifiedBy`	Person who last modified this model
`ModifiedByFormat`	Format of the ModifiedBy parameter. Value can be an string. The string can include the tag `%<Auto>`. Simulink replaces the tag with the current value of the `USER` environment variable.

Property	Description
`ModifiedDate`	Date modified
`ModifiedDateFormat`	Format of the ModifiedDate parameter. Value can be any string. The string can include the tag %<Auto>. Simulink replaces the tag with the current date and time when saving the model.
`ModifiedComment`	Comment entered by user who last updated this model
`ModifiedHistory`	History of changes to this model
`ModelVersion`	Version number
`ModelVersionFormat`	Format of model version number. Can be any string. The string can contain the tag `%<AutoIncrement:#>` where # is an integer. Simulink replaces the tag with # when displaying the version number. It increments # when saving the model.
`Description`	Description of model
`LastModificationDate`	Date last modified.

Ending a Simulink Session

Terminate a Simulink session by closing all Simulink windows.

Terminate a MATLAB session by choosing one of these commands from the **File** menu:

- On a Microsoft Windows system: **Exit MATLAB**
- On a UNIX system: **Quit MATLAB**

4

Running a Simulation

Introduction

You can run a simulation either by using Simulink menu commands or by entering commands in the MATLAB command window.

Many users use menu commands while they develop and refine their models, then enter commands in the MATLAB command window to run the simulation in "batch" mode.

Using Menu Commands

Running a simulation using menu commands is easy and interactive. These commands let you select an ordinary differential equation (ODE) solver and define simulation parameters without having to remember command syntax. An important advantage is that you can perform certain operations interactively while a simulation is running:

- You can modify many simulation parameters, including the stop time, the solver, and the maximum step size.
- You can change the solver.
- You can simulate another system at the same time.
- You can click on a line to see the signal carried on that line on a floating (unconnected) Scope or Display block.
- You can modify the parameters of a block, as long as you do not cause a change in:
  - The number of states, inputs, or outputs
  - The sample time
  - The number of zero crossings
  - The vector length of any block parameters
  - The length of the internal block work vectors

You cannot make changes to the structure of the model, such as adding or deleting lines or blocks, during a simulation. If you need to make these kinds of changes, you need to stop the simulation, make the change, then start the simulation again to see the results of the change.

Running a Simulation from the Command Line

Running a simulation from the command line has these advantages over running a simulation using menu commands:

- You can simulate M-file and MEX-file models, as well as Simulink block diagram models.
- You can run a simulation from an M-file, allowing simulation and block parameters to be changed iteratively.

For more information, see “Running a Simulation from the Command Line” on page 4-29.

Running a Simulation Using Menu Commands

This section discusses how to use Simulink menu commands and the **Simulation Parameters** dialog box to run a simulation.

Setting Simulation Parameters and Choosing the Solver

You set the simulation parameters and select the solver by choosing **Parameters** from the **Simulation** menu. Simulink displays the **Simulation Parameters** dialog box, which uses three "pages" to manage simulation parameters:

- The **Solver** page allows you to set the start and stop times, choose the solver and specify solver parameters, and choose some output options.
- The **Workspace I/O** page manages input from and output to the MATLAB workspace.
- The **Diagnostics** page allows you to select the level of warning messages displayed during a simulation.

Each page of the dialog box, including the parameters you set on the page, is discussed in detail in "The Simulation Parameters Dialog Box" on page 4-8.

You can specify parameters as valid MATLAB expressions, consisting of constants, workspace variable names, MATLAB functions, and mathematical operators.

Applying the Simulation Parameters

After you have set the simulation parameters and selected the solver, you are ready to apply them to your model. Press the **Apply** button on the bottom of the dialog box to apply the parameters to the model. To apply the parameters and close the dialog box, press the **Close** button.

Starting the Simulation

After you have applied the solver and simulation parameters to your model, you are ready to run the simulation. Select **Start** from the **Simulation** menu to run the simulation. You can also use the keyboard shortcut, **Ctrl-T**. When you select **Start**, the menu item changes to **Stop**.

Your computer beeps to signal the completion of the simulation.

Note A common mistake that new Simulink users make is to start a simulation while the Simulink block library is the active window. Make sure your model window is the active window before starting a simulation.

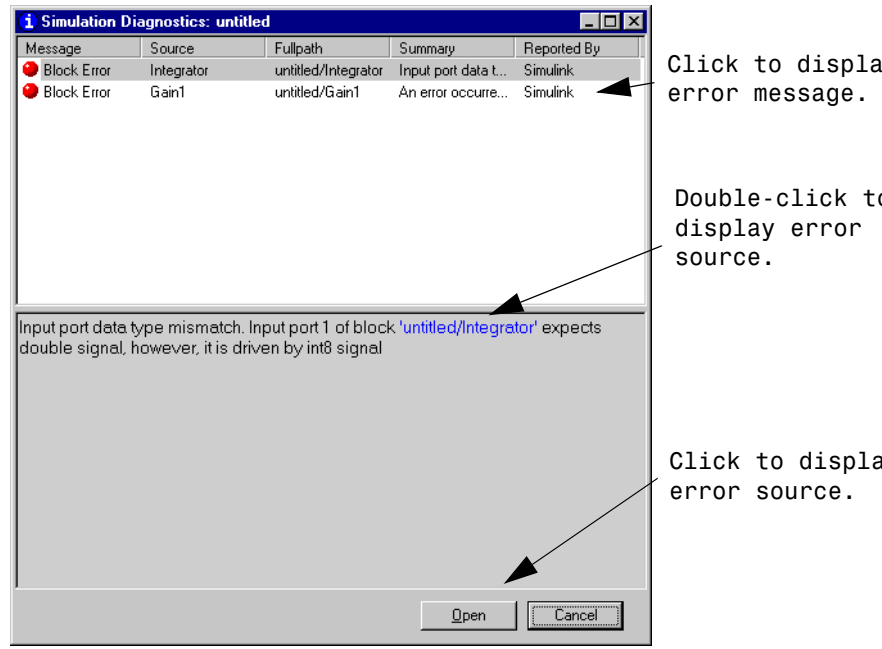
To stop a simulation, choose **Stop** from the **Simulation** menu. The keyboard shortcut for stopping a simulation is **Ctrl-T**, the same as for starting a simulation.

You can suspend a running simulation by choosing **Pause** from the **Simulation** menu. When you select **Pause**, the menu item changes to **Continue**. You proceed with a suspended simulation by choosing **Continue**.

If the model includes any blocks that write output to a file or to the workspace, or if you select output options on the **Simulation Parameters** dialog box, Simulink writes the data when the simulation is terminated or suspended.

Simulation Diagnostics Dialog Box

If errors occur during a simulation, Simulink halts the simulation and displays the errors in the **Simulation Diagnostics** dialog box.



The dialog box has two panes. The upper pane consist of columns that display the following information for each error.

Message. Message type (for example, block error, warning, log)

Source. Name of the model element (for example, a block) that caused the error.

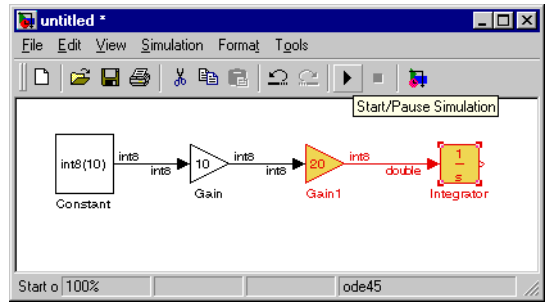
Fullpath. Path of the element that caused the error.

Summary. Error message abbreviated to fit in the column.

Reported by. Component that reported the error (for example, Simulink, Stateflow, Real-Time Workshop, etc).

The lower pane initially contains the full content of the first error message listed in the top pane. You can display the content of other messages by single-clicking on their entries in the upper pane.

In addition to displaying the **Simulation Diagnostics** dialog box, Simulink also opens (if necessary) the diagram that contains the error source and highlights the source.



You can similarly display other error sources by double-clicking on the corresponding error message in the top pane, by double-clicking on the name of the error source in the error message (highlighted in blue), or by selecting the **Open** button on the dialog box.

The Simulation Parameters Dialog Box

This section discusses the simulation parameters, which you specify either on the **Simulation Parameters** dialog box or using the `sim` (see `sim` on page 4-30) and `simset` (see `simset` on page 4-32) commands. Parameters are described as they appear on the dialog box pages.

This table summarizes the actions performed by the dialog box buttons, which appear on the bottom of each dialog box page.

Table 4-1: Simulation Parameters Dialog Box Buttons

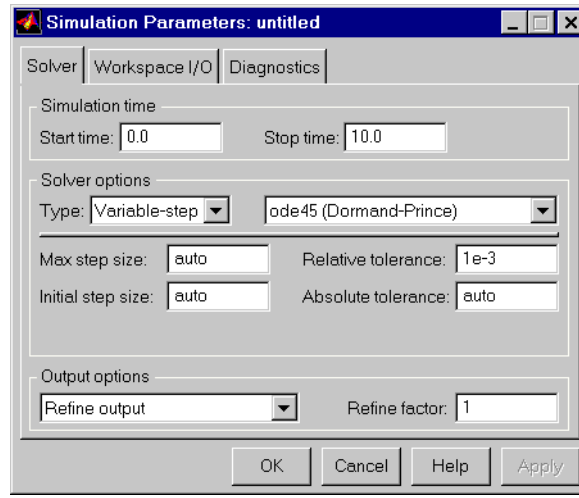
Button	Action
Ok	Applies the parameter values and closes the dialog box. During a simulation, the parameter values are applied immediately.
Cancel	Changes the parameter values back to the values they had when the dialog box was most recently opened and closes the dialog box.
Help	Displays help text for the dialog box page.
Apply	Applies the current parameter values and keeps the dialog box open. During a simulation, the parameter values are applied immediately.

The Solver Page

The **Solver** page appears when you first choose **Parameters** from the **Simulation** menu or when you select the **Solver** tab.

The **Solver** page allows you to:

- Set the simulation start and stop times
- Choose the solver and specify its parameters
- Select output options



Simulation Time

You can change the start time and stop time for the simulation by entering new values in the **Start time** and **Stop time** fields. The default start time is 0.0 seconds and the default stop time is 10.0 seconds.

Simulation time and actual clock time are not the same. For example, running a simulation for 10 seconds will usually not take 10 seconds. The amount of time it takes to run a simulation depends on many factors, including the model's complexity, the solver's step sizes, and the computer's clock speed.

Solvers

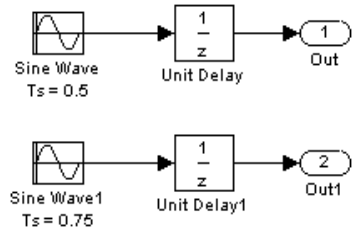
Simulation of Simulink models involves the numerical integration of sets of ordinary differential equations (ODEs). Simulink provides a number of solvers for the simulation of such equations. Because of the diversity of dynamic system behavior, some solvers may be more efficient than others at solving a particular problem. To obtain accurate and fast results, take care when choosing the solver and setting parameters.

You can choose between variable-step and fixed-step solvers. *Variable-step solvers* can modify their step sizes during the simulation. They provide error control and zero crossing detection. *Fixed-step solvers* take the same step size

during the simulation. They provide no error control and do not locate zero crossings. For a thorough discussion of solvers, see *Using MATLAB*.

Default solvers. If you do not choose a solver, Simulink chooses one based on whether your model has states:

- If the model has continuous states, `ode45` is used. `ode45` is an excellent general purpose solver. However, if you know that your system is stiff and if `ode45` is not providing acceptable results, try `ode15s`. For a definition of stiff, see the note at the end of the section "Variable-step solvers" on page 4-11.
- If the model has no continuous states, Simulink uses the variable-step solver called `discrete` and displays a message indicating that it is not using `ode45`. Simulink also provides a fixed-step solver called `discrete`. This model shows the difference between the two `discrete` solvers.



With sample times of 0.5 and 0.75, the *fundamental sample time* for the model is 0.25 seconds. The difference between the variable-step and the fixed-step `discrete` solvers is the time vector that each generates.

The fixed-step `discrete` solver generates this time vector:

```
[0.0 0.25 0.5 0.75 1.0 1.25 ...]
```

The variable-step `discrete` solver generates this time vector:

```
[0.0 0.5 0.75 1.0 1.5 2.0 2.25 ...]
```

The step size of the fixed-step `discrete` solver is the fundamental sample time. The variable-step `discrete` solver takes the largest possible steps.

Variable-step solvers. You can choose these variable-step solvers: `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, and `discrete`. The default is `ode45` for systems with states, or `discrete` for systems with no states:

- `ode45` is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a *one-step* solver; that is, in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{n-1})$. In general, `ode45` is the best solver to apply as a "first try" for most problems.
- `ode23` is also based on an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than `ode45` at crude tolerances and in the presence of mild stiffness. `ode23` is a one-step solver.
- `ode113` is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than `ode45` at stringent tolerances. `ode113` is a *multistep* solver; that is, it normally needs the solutions at several preceding time points to compute the current solution.
- `ode15s` is a variable order solver based on the numerical differentiation formulas (NDFs). These are related to but are more efficient than the backward differentiation formulas, BDFs (also known as Gear's method). Like `ode113`, `ode15s` is a multistep method solver. If you suspect that a problem is stiff or if `ode45` failed or was very inefficient, try `ode15s`.
- `ode23s` is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than `ode15s` at crude tolerances. It can solve some kinds of stiff problems for which `ode15s` is not effective.
- `ode23t` is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping.
- `ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like `ode23s`, this solver may be more efficient than `ode15s` at crude tolerances.
- `discrete` (variable-step) is the solver Simulink chooses when it detects that your model has no continuous states.

Note For a *stiff* problem, solutions can change on a time scale that is very short compared to the interval of integration, but the solution of interest changes on a much longer time scale. Methods not designed for stiff problems are ineffective on intervals where the solution changes slowly because they use time steps small enough to resolve the fastest possible change. Jacobian matrices are generated numerically for `ode15s` and `ode23s`. For more information, see Shampine, L. F., *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, 1994.

Fixed-step solvers. You can choose these fixed-step solvers: `ode5`, `ode4`, `ode3`, `ode2`, `ode1`, and `discrete`:

- `ode5` is the fixed-step version of `ode45`, the Dormand-Prince formula.
- `ode4` is RK4, the fourth-order Runge-Kutta formula.
- `ode3` is the fixed-step version of `ode23`, the Bogacki-Shampine formula.
- `ode2` is Heun's method, also known as the improved Euler formula.
- `ode1` is Euler's method.
- `discrete` (fixed-step) is a fixed-step solver that performs no integration. It is suitable for models having no states and for which zero crossing detection and error control are not important.

If you think your simulation may be providing unsatisfactory results, see "Improving Simulation Performance and Accuracy" on page 4-27.

Solver Options

The default solver parameters provide accurate and efficient results for most problems. In some cases, however, tuning the parameters can improve performance. (For more information about tuning these parameters, see "Improving Simulation Performance and Accuracy" on page 4-27). You can tune the selected solver by changing parameter values on the **Solver** panel.

Step Sizes

For variable-step solvers, you can set the maximum and suggested initial step size parameters. By default, these parameters are automatically determined, indicated by the value `auto`.

For fixed-step solvers, you can set the fixed step size. The default is also `auto`.

Maximum step size. The **Max step size** parameter controls the largest time step the solver can take. The default is determined from the start and stop times:

$$h_{max} = \frac{t_{stop} - t_{start}}{50}$$

Generally, the default maximum step size is sufficient. If you are concerned about the solver missing significant behavior, change the parameter to prevent the solver from taking too large a step. If the time span of the simulation is very long, the default step size may be too large for the solver to find the solution. Also, if your model contains periodic or nearly periodic behavior and you know the period, set the maximum step size to some fraction (such as 1/4) of that period.

In general, for more output points, change the refine factor, not the maximum step size. For more information, see "Refine output" on page 4-16.

Initial step size. By default, the solvers select an initial step size by examining the derivatives of the states at the start time. If the first step size is too large, the solver may step over important behavior. The initial step size parameter is a *suggested* first step size. The solver tries this step size but reduces it if error criteria are not satisfied.

Error Tolerances

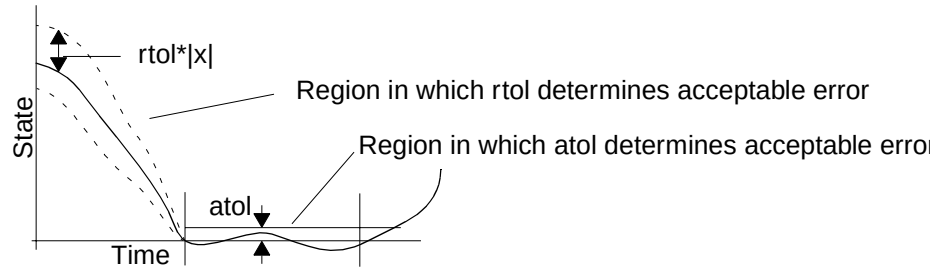
The solvers use standard local error control techniques to monitor the error at each time step. During each time step, the solvers compute the state values at the end of the step and also determine the *local error*, the estimated error of these state values. They then compare the local error to the *acceptable error*, which is a function of the relative tolerance (*rtol*) and absolute tolerance (*atol*). If the error is greater than the acceptable error for *any* state, the solver reduces the step size and tries again:

- *Relative tolerance* measures the error relative to the size of each state. The relative tolerance represents a percentage of the state's value. The default, 1e-3, means that the computed state will be accurate to within 0.1%.
- *Absolute tolerance* is a threshold error value. This tolerance represents the acceptable error as the value of the measured state approaches zero.

The error for the ith state, e_i, is required to satisfy.

$$e_i \le max(rtol \times |x_i|, atol_i)$$

The figure below shows a plot of a state and the regions in which the acceptable error is determined by the relative tolerance and the absolute tolerance:



If you specify `auto` (the default), Simulink sets the absolute tolerance for each state initially to 1e-6. As the simulation progresses, Simulink resets the absolute tolerance for each state to the maximum value that the state has assumed thus far times the relative tolerance for that state. Thus, if a state goes from 0 to 1 and `reltol` is 1e-3, then by the end of the simulation the `abstol` is set to 1e-3 also. If a state goes from 0 to 1000, then the `abstol` is set to 1.

If the computed setting is not suitable, you can determine an appropriate setting yourself. You might have to run a simulation more than once to determine an appropriate value for the absolute tolerance. If the magnitudes of the states vary widely, it might be appropriate to specify different absolute tolerance values for different states. You can do this on the Integrator block's dialog box.

The Maximum Order for ode15s

The `ode15s` solver is based on NDF formulas of order one through five. Although the higher order formulas are more accurate, they are less stable. If your model is stiff and requires more stability, reduce the maximum order to 2 (the highest order for which the NDF formula is A-stable). When you choose the `ode15s` solver, the dialog box displays this parameter.

As an alternative, you might try using the `ode23s` solver, which is a fixed-step, lower order (and A-stable) solver.

Multitasking Options

If you select a fixed-step solver, the **Solver** page of the **Simulation Parameters** dialog box displays a **Mode** options list. The list allows you to select one of the following simulation modes.

MultiTasking. This mode issues an error if it detects an illegal sample rate transition between blocks, that is, a direct connection between blocks operating at different sample rates. In real-time multitasking systems, illegal sample rate transitions between tasks can result in a task's output not being available when needed by another task. By checking for such transitions, multitasking mode helps you to create valid models of real-world multitasking systems, where sections of your model represent concurrent tasks.

Use *rate transition* blocks to eliminate illegal rate transitions from your model. Simulink provides two such blocks: Unit Delay (see `Unit Delay` on page 8-214) and Zero-Order Hold (see `Zero-Order Hold` on page 8-221). To eliminate an illegal slow-to-fast transition, insert a Unit Delay block running at the slow rate between the slow output port and the fast input port. To eliminate an illegal fast-to-slow transition, insert a Zero-Order Hold block running at the slow rate between the fast output port and the slow input port. For more information, see Chapter 7, "Models with Multiple Sample Rates," in the *Real-Time Workshop Users Guide*.

SingleTasking. This mode does not check for sample rate transitions among blocks. This mode is useful when you are modeling a single-tasking system. In such systems, task synchronization is not an issue.

Auto. This option causes Simulink to use single-tasking mode if all blocks operate at the same rate and multitasking mode if the model contains blocks operating at different rates.

Output Options

The **Output options** area of the dialog box enables you to control how much output the simulation generates. You can choose from three popup options:

- Refine output
- Produce additional output
- Produce specified output only

Refine output. The **Refine output** choice provides additional output points when the simulation output is too coarse. This parameter provides an integer number of output points between time steps; for example, a refine factor of 2 provides output midway between the time steps, as well as at the steps. The default refine factor is 1.

To get smoother output, it is much faster to change the refine factor instead of reducing the step size. When the refine factor is changed, the solvers generate additional points by evaluating a continuous extension formula at those points. Changing the refine factor does not change the steps used by the solver.

The refine factor applies to variable-step solvers and is most useful when using `ode45`. The `ode45` solver is capable of taking large steps; when graphing simulation output, you may find that output from this solver is not sufficiently smooth. If this is the case, run the simulation again with a larger refine factor. A value of 4 should provide much smoother results.

Produce additional output. The **Produce additional output** choice enables you to specify directly those additional times at which the solver generates output. When you select this option, Simulink displays an **Ouput Times** field on the **Solver** page. Enter a MATLAB expression in this field that evaluates to an additional time or a vector of additional times. The additional output is produced using a continuous extension formula at the additional times. Unlike the refine factor, this option changes the simulation step size so that time steps coincide with the times that you have specified for additional output.

Produce specified output only. The **Produce specified output only** choice provides simulation output *only* at the specified output times. This option changes the simulation step size so that time steps coincide with the times that you have specified for producing output. This choice is useful when comparing different simulations to ensure that the simulations produce output at the same times.

Comparing Output options. A sample simulation generates output at these times:

```
0, 2.5, 5, 8.5, 10
```

Choosing **Refine output** and specifying a refine factor of 2 generates output at these times:

```
0, 1.25, 2.5, 3.75, 5, 6.75, 8.5, 9.25, 10
```

Choosing the **Produce additional output** option and specifying `[0:10]` generates output at these times:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

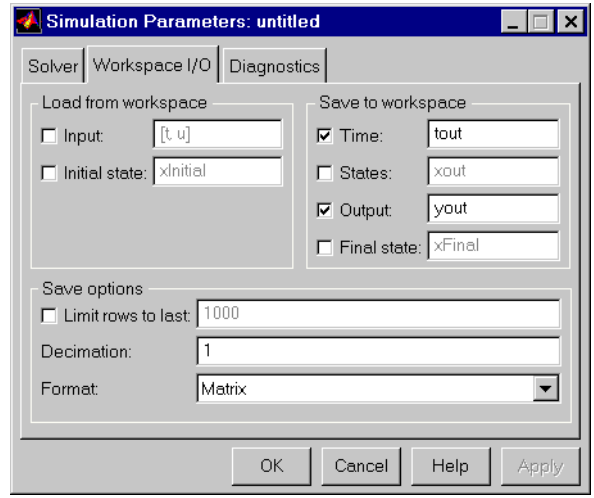
and perhaps at additional times, depending on the step-size chosen by the variable-step solver.

Choosing the **Produce Specified Output Only** option and specifying `[0:10]` generates output at these times:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

The Workspace I/O Page

You can direct simulation output to workspace variables and get input and initial states from the workspace. On the **Simulation Parameters** dialog box, select the **Workspace I/O** tab. This page appears:



Loading Input from the Base Workspace

Simulink can apply input from a model's base workspace to the model's top-level inports during a simulation run. To specify this option, check the **Input** box in the **Load from workspace** area of the **Workspace I/O** page. Then, enter an external input specification (see below) in the adjacent edit box and select **Apply**.