VisSim

User's Guide

Version 3

Visual Solutions, Inc.

VisSim User's Guide - Version 3

Contents

Introduction xiii

Chapter 1 VisSim Basics 1

Starting VisSim 1
Exploring the VisSim window 2
Choosing commands 4
Using dialog boxes 6
Creating and setting up a new block diagram 7
Opening an existing block diagram 10
Undoing an editing action 10
Repainting the screen 10
Saving a block diagram 11
Previewing before printing 11
Printing 11
Setting up the VisSim environment 13
Exiting VisSim 13

Chapter 2 Inserting, Setting Up, and Wiring Blocks 15

Block basics 15
  Types of blocks 16
  Identifying block parts 16
Inserting blocks 17
Setting up block properties and initial conditions 17
  Entering numeric data 18
  Entering arithmetic expressions 18
  Entering C expressions 18
  Controlling the number of displayed significant decimal digits 19
Wiring basics 19
  Types of wires 19
  Wiring rules 20
  Wiring blocks together 20
  Automatically completing connections 21
  Positioning wires 21
  Coloring wires 21

Hiding wires 22
Deleting wires 22
Connector tab basics 22
Adding and removing connector tabs 22
Unconnected input connector tabs 23
Setting connection classes 23
Displaying connector tabs in a different view 24

Chapter 3 Arranging Blocks 25

Selecting blocks 25
Area Selecting 26
Toggle selecting 26
Unselecting blocks 26
Moving and copying blocks 26
Rules for moving and copying blocks 27
Drag-and-drop editing 27
Copying, cutting, and pasting blocks 28
Copying blocks into other applications 29
Flipping blocks 29
Aligning blocks vertically and horizontally 29
Finding and replacing blocks 30
Finding blocks 30
Replacing blocks 31
Deleting blocks 32

Chapter 4 Setting Simulation Properties 33

Setting up the simulation range 33
Using the Range property sheet 34
Setting up an integration method 35
Using the Integration Method property sheet 36
Setting up simulation preferences 38
Using the Preferences property sheet 39
Setting simulation defaults 41

Chapter 5 Simulating Block Diagrams 43

Simulation basics 43
Continuous system simulation 44
Discrete time system simulation 51
Hybrid system simulation 54

Controlling a simulation ... 54
The Control Panel ... 54
Starting a simulation ... 55
Stopping a simulation ... 55
Continuing a simulation ... 55
Single-stepping a simulation ... 56
Resetting a simulation to initial conditions ... 56
More on controlling a simulation ... 56
Dynamically modifying signal values ... 56
Probing signal values ... 56
Trimming a system ... 57
Resetting error conditions ... 57
Snapping system states ... 57
Troubleshooting ... 58

Chapter 6 Viewing Simulations ... 59

Plots ... 59
Basic time domain plots ... 60
Sizing a plot block ... 60
Zooming ... 61
Changing plot properties ... 61
Strip charts ... 67
Basic time domain strip chart ... 68
Sizing a stripChart block ... 68
Printing a stripChart block ... 68
Changing stripChart properties ... 68
Histograms ... 73
Sizing a histogram block ... 74
Changing histogram properties ... 74
Bar and needle graphs ... 75
Sizing a meter block ... 75
Changing meter properties ... 76
Creating animation ... 77
Animation basics ... 77
Using the animate block ... 78
Using the lineDraw block ... 81
Other ways to create animation ... 82

Chapter 7 Solving Implicit Equations ... 83

Setting up an implicit equation ... 83
Solving an implicit equation ... 84

Using the Implicit Solver property sheet 85
Implicit equation examples 86
Simple nonlinear implicit equation 86
Advanced nonlinear implicit equation 86

Chapter 8 Performing Global Optimization 89

Global optimization basics 89
Cost functions with many local minimum values 89
Cost functions with no minimum values 90
Performing global optimization 90
Using the Optimization Properties dialog box 91
Global optimization examples 91
Optimized paper bag problem 91
Two segment approximation of sin(πt) 93
Five segment approximation of sin(πt) 94
Troubleshooting 96

Chapter 9 Designing Digital Filters 97

Digital filter basics 97
Filter operations 98
Time domain filters with tapped delay 98
Time domain filters with transfer functions 99
Frequency domain filter implementation 99
Comparison of FIR and IIR filters 100
Interactive filter design with the transferFunction block 100
IIR filter design 100
Using the IIR Filter Properties dialog box 101
FIR filter design 104
Discrete and continuous FIR filter design 104
Using the FIR Filter Properties dialog box 105

Chapter 10 Working with Other Applications 109

Importing basics 109
Setting up the input file 109
Importing data 110
Using the Import Properties dialog box 110
Exporting basics 111
Exporting data 111
Using the Export Properties dialog box 112

DDE basics 114
Creating an app-to-VisSim link with DDEreceive 115
Creating a VisSim-to-app link with DDEsend 117
Creating a two-way link with DDE 119
Creating DDE links with applications that do not support Copy Link and Paste Link 122

Chapter 11 Working with Large Diagrams 123

Creating model hierarchy 123
Creating a compound block 124
Drilling into a compound block 124
Hiding compound blocks 125
Configuring pictures on compound blocks 125
Labeling connector tabs on compound blocks 126
Dissolving a compound block 126
Other things you can do with compound blocks 127
Embedding blocks 127
Setting up a diagram to be embedded 127
Embedding a block diagram 127
Editing an embedded block diagram 128
Reconnecting an embedded block diagram 128
Adding block diagrams 128
Using variables to pass signals 129
Creating variables 129
Scoping variables 130
Built-in variables 132
Using path aliases to reference files 132
Creating path aliases 132
Inserting path aliases in blocks 133
Maintaining an edit history 134
Protecting your work 134
Protecting block diagrams 135
Protecting compound blocks 136
Protecting embed blocks 138

Chapter 12 Block Reference 141

* (multiply) 142
-X (negate) 143
/ (divide) 145
< (less than) 147
<= (less than or equal to) 148

== (equal to) ... 149
!= (not equal to) ... 152
> (greater than) ... 153
>= (greater than or equal to) ... 155
1/X (inverse) ... 156
abs ... 157
acos ... 159
and ... 160
animate ... 161
asin ... 161
atan2 ... 162
bessel ... 163
bezel ... 165
buffer ... 166
button ... 168
case ... 169
comment ... 172
const ... 172
constraint ... 173
convert ... 173
cos ... 173
cosh ... 174
cost ... 174
crossDetect ... 175
date ... 178
DDE ... 178
DDEreceive ... 178
DDEsend ... 178
deadband ... 179
derivative ... 179
display ... 180
dotProduct ... 181
embed ... 181
error ... 181
exp ... 182
export ... 182
expression ... 182
fft ... 184
gain ... 186
gaussian ... 188
globalConstraint ... 188
histogram ... 188
ifft ... 189
import ... 190

index 190
int 192
integrator (1/S) 192
invert 198
label 198
light 199
limit 201
limitedIntegrator (1/S) 202
lineDraw 205
log10 205
ln 206
map 206
max 215
merge 217
meter 218
min 218
multiply 220
neuralNet 221
not 221
or 222
parabola 223
parameterUnknown 224
plot 224
pow 224
PRBS 226
pulseTrain 226
quantize 227
ramp 229
realTime 229
relay 230
resetIntegrator (1/S) 231
rt-DataIn 234
rt-DataOut 234
sampleHold 235
scalarToVec 237
sign 238
sin 239
sinh 240
sinusoid 240
slider 241
sqrt 242
stateSpace 242
step 244
stop 244

stripChart 245
summingJunction 245
tan 247
tanh 248
timeDelay 249
transferFunction 252
transpose 255
uniform 256
unitConversion 256
unitDelay 257
unknown 259
userFunction 260
variable 260
vecToScalar 260
vsum 261
wirePositioner 261
xor 261

Appendix A Customizing VisSim 263

Customizing VisSim start-up 263
Customizing the VisSim window 264
    Customizing the toolbar 265
    Customizing other screen settings 266
Creating custom implicit solvers 267
    Source files for building a custom implicit solver 268
    Using vissimRequest() in a custom implicit solver 268
    Building a custom implicit solver 270
    Using the constraint block with a custom implicit solver 270
Creating custom global optimizers 270
    Source files for building a custom global optimizer 271
    Using vissimRequest() in a custom global optimizer 271
    Building a custom global optimizer 273

Appendix B Extending the Block Set 275

The big picture 275
    Criteria for writing DLLs 276
    Building a DLL 276
How VisSim talks to a DLL 277
    Calling conventions 277
    Simulation level functions 278
    Block level functions 279

Exported functions 284
Debugging hints 287
Binding a DLL to a userFunction block 287
Adding a user-written block to the Blocks menu 288

Appendix C Toolbox and Components Libraries 289

Toolboxes 289
Controls toolbox library 289
Electro-mechanical toolbox library 290
Padé toolbox library 290
Signal generation toolbox library 291
Tools toolbox library 291
Components 292
DSP library 292
Dynamical system library 292
Electro-mechanical library 292
Electrical library 292
Hydraulic libraries 293
Process control library 295
Thermal control library 295
Turbine library 295

Appendix D Sample Block Diagrams 297

Aerospace block diagrams 298
Animation block diagrams 298
Biophysical block diagrams 298
Business block diagrams 298
Chemical engineering block diagrams 299
Control design block diagrams 300
Dynamical systems block diagrams 300
Electro-mechanical block diagrams 301
Environmental block diagrams 301
Fixed-point DSP block diagrams 301
Man-machine interface block diagrams 301
Motion control block diagrams 301
Optimization block diagrams 302
Power block diagrams 302
Process control block diagrams 302
Signal processing block diagrams 303
Logic diagrams and state machines 303

Appendix D Working with Bitmaps 305

VisSim Viewer 313

Distributing VisSim Viewer and your block diagrams to end users 313
VisSim Viewer documentation 313
Installing and starting VisSim Viewer 314
VisSim Viewer End User License Agreement 315
VisSim Viewer Distribution Agreement 316

Index 317

Introduction

Welcome to VisSim 3, the most comprehensive modeling and simulation environment for developing continuous, discrete, multi-rate, and hybrid system models and running dynamic simulations on IBM PCs and compatibles. VisSim 3 contains numerous features for simplifying system design, enhancing modeling capabilities, and strengthening its simulation engine. A thumbnail description of each feature can be found on page xvi.

The *VisSim User's Guide* contains a comprehensive description about using VisSim on the Windows 3.1, Windows 95, and Windows NT platforms. If you've purchased Micro-VisSim 3, please read "For Micro-VisSim users," at the bottom of this page for a list of specifics about your version of the software.

Registering your software

Before you begin using VisSim, please fill out the enclosed registration card and mail it to us. As a registered user, you will receive a free subscription to The flexWire, along with discount promotions and VisSim workshop schedules.

For Micro-VisSim users

If you purchased Micro-VisSim, the following limitations apply to your software:

- 100 blocks per diagram
- `userFunction` block is unavailable for use
- VisSim Viewer is unavailable for use

In addition, you also receive a compact version of VisSim/Analyze that allows you to linearize systems containing up to seven states.

Conventions used in this book

The following typographical conventions are used in this manual:

Convention	Where it's used
Shortcut key combinations	Shortcut key combinations are joined with the plus sign (+). For example, the command CTRL+C means to hold down the CTRL key while you press the C key.
Hot keys	Hot keys are the underlined keys in VisSim's menus, commands, and dialog boxes. To use a hot key, press ALT and then the key for the underlined character. For instance, to execute the File menu's Save command, hold down the ALT key while you press the F key, then release both keys and press the S key.
SMALL CAPS	To indicate the names of the keys on the keyboard.
ALL CAPS	To indicate directory names, file names, and acronyms.
Initial Caps	To indicate menu names, commands names, and dialog box options.

In addition, unless specifically stated otherwise, when you read "click the mouse…" or "click on…," it means click the left mouse button.

Getting help

To help you get the most out of VisSim, the following online information is available:

- **Online help.** The online help contains step-by-step instructions for using VisSim features.
- **Online release notes.** A file named README.TXT is installed in your main VisSim directory. This file contains last minute information and changes that were discovered after this manual went to print. For your convenience, you should read this file immediately and print a copy of it to keep with this manual.

You may also find it helpful to browse through the sample block diagrams included with VisSim. These diagrams, which are listed in Appendix D, "Sample Block Diagrams," demonstrate how VisSim is used to solve a broad spectrum of engineering and scientific problems.

Online help

VisSim's Help program provides online instructions for using VisSim.

▶ **To open Help**

- Do one of the following:

To	Do this
Access the top level of help	Select Help from the menu bar or press ALT+H.
Access help on the selected block	Click on the Help command button in the dialog box for the block.

▶ **To close Help**

- In the Help window, choose File > Exit, or press ALT+F4.

Technical support

When you need assistance with a Visual Solutions product, first look in the manual, read the README.TXT file, and consult the online Help program. If you cannot find the answer, contact the Technical Support group via toll call between 9:00 am and 6:00 pm Eastern Standard Time, Monday through Friday, excluding holidays. The phone number is **978-392-0100.**

When you call in, please have the following information at hand:

- The version of VisSim and the version of the software operating environment that you're using
- All screen messages
- What you were doing when the problem happened
- How you tried to solve the problem

Visual Solutions also has the following fax and email addresses:

Address/Number	What it's for
978-692-3102	Fax number
bugs@vissol.com	Bug report
doc@vissol.com	Documentation errors and suggestions
sales@vissol.com	Sales, pricing, and general information
tech@vissol.com	Technical support

What's new in VisSim 3

Feature	Function	Benefit	Application
Enhanced vector and matrix operations	Handles vector and matrix algebra (buffer, dotProduct, invert, multiply, transpose, and vsum)	Simplifies model design	6 DOF aerospace simulations, state-space system simulation and control
C expression block	Allows C command or expression to be part of a VisSim diagram	Simplifies model design	Any simulation where arithmetic expressions need to be simplified
derivative block	Calculates the change in function value with respect to time.	Extends modeling capabilities	All simulations
Windows Explorer-like interface	Visually depicts organizational hierarchy of a diagram	Improves ability to navigate block diagram models	Large, complex models with multiple hierarchical levels
Probe data value	Displays data entering and exiting blocks	Extends data visualization capabilities	All simulations
Data conversion	Converts units of measurement of data (supports temperature, capacitance, speed, mass, energy, and power conversions and more)	Extends modeling capabilities	All simulations
Connector labels	Adds connector labels to compound blocks	Enhances model readability	All simulations
Connection class	Categorizes connections by class name and color for easy recognition of subcomponents	Enhances model readability	All simulations

Feature	Function	Benefit	Application
3-D mapping	Provides simultaneous mapping for three independent variables	Extends modeling capabilities	All simulations
Floating labels	Displays labels on top of other blocks when they overlap	Enhances model readability	All simulations
Multiple data types	Supports char, unsigned char, short, unsigned short, int, long, unsigned long, float, and double.	Extends modeling capabilities	Prototyping and development of fixed point and mixed (fixed and floating point) system simulations and automatic code generation
Data type propagation	Follows ANSI C data propagation rules when mixing data types and propagates the correct data type	Extends modeling capabilities	Same as above
Color coding of data types	Displays block data types by color	Enhances model readability	Same as above
Print preview	Displays a diagram as it will look when printed	Enhances printing capabilities	All simulations
Auto-pan	Scrolls when the mouse nears window edge	Enhances user interface	All simulations
Customizable headers and footers	Allows print macros in user-defined header and footer text strings	Creates customized reports	All simulations
Path macros	Allows user-defined names in file paths	Aids multi-platform installations	Large models
Floating Find and Replace Block commands	Navigates models by selecting layer of matched block name	Extends modeling and debugging capabilities	Large models

Feature	Function	Benefit	Application
Multi-XY traces	Allow two simultaneous XY traces on one plot (for example, “seeker *vs*. target”)	Extends modeling capabilities	Multi-body simulations
Enhanced simulation warnings	Warn if time delay or pulse interval is non-integral multiple of base step size	Speeds model debugging	All simulations
Tool tips	Pop-up toolbutton descriptions	Enhances user interface	All simulations
Tear-off toolbars	Move toolbars anywhere on screen	Enhances user interface	All simulations

References to other books

For information on	Refer to
Block diagram modeling and simulation	Karayanakis, Nicholas M., *Computer-Assisted Simulation of Dynamic Systems with Block Diagram Languages.* CRC Press, 1993.
Scientific computing	Abramowitz, M.; Stegun, I. A. *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55, Washington: National Bureau of Standards; reprinted Dover Publications, New York, 1968.
	D’Azzo, John J.; Houpis, Constantine H. *Linear Control System Analysis & Design - Conventional and Modern.* McGraw-Hill Book Company, 1988.
	Fitzgerald, A. E.; Kingsley, Charles Jr.; Umans, Stephen D. *Electric Machinery.* McGraw-Hill Book Company, 1983.
	Flannery, B. P.; Press, W. H.; S. A.; Vetterling, W. T. *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, 1989.

For information on	Refer to
Scientific computing	Franklin, Gene F.; Powell, David J. *Digital Control of Dynamic Systems*. Addison-Wesley Publishing Company, 1980.
	Gear, C. W. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, 1971.
	Stoer, J.; Bulirsh, R. *Introduction to Numerical Analysis.* New York: Springer-Verlag, 1980.
Computer Programming	Darnell, Peter A.; Margolis, Philip E. *C: A Software Engineering Approach*. Springer-Verlag, 1990.

Chapter 1

VisSim Basics

This chapter covers the following information:

- Starting VisSim
- Exploring the VisSim window
- Choosing commands and using dialog boxes
- Creating block diagrams
- Opening block diagrams
- Undoing an editing action
- Repainting the screen
- Saving block diagrams
- Previewing and printing block diagrams
- Setting up VisSim
- Quitting VisSim

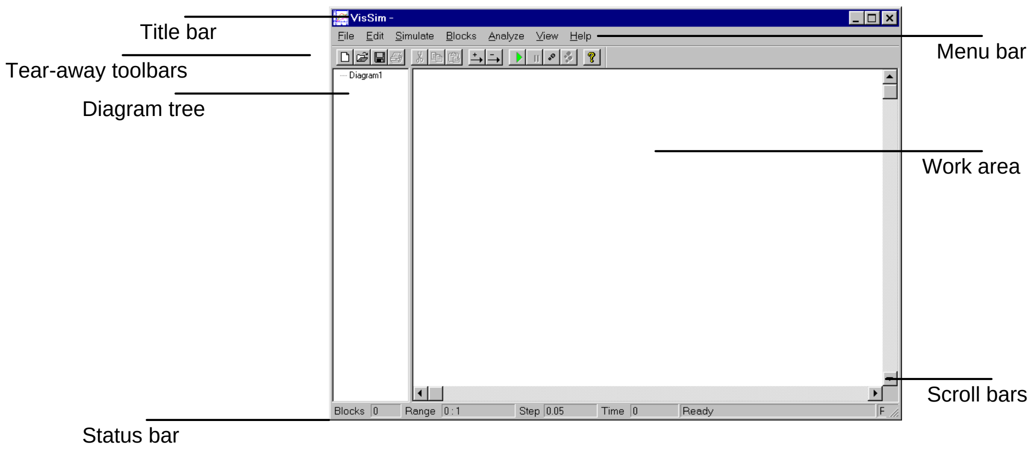
Starting VisSim

The table below describes the start-up methods for Professional VisSim and Micro-VisSim. You can customize how VisSim starts up by editing the start-up command line, as described on page 263.

Platform	Start-up method
Windows 3.1+	Start up the Program Manager; then double-click on the VisSim icon in the VisSim group window.
Windows 95 and Windows NT	Click on Start > Programs > VisSim; then double-click on the VisSim icon.

Exploring the VisSim window

When you start VisSim, a new, empty block diagram, like the one shown below, is automatically created for you.

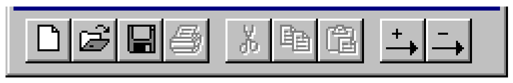


Title bar: Lists the application name and currently opened block diagram. Unnamed diagrams are titled *Diagram1*. The title bar also contains the Minimize, Maximize, and Close buttons. The Minimize button shrinks the VisSim window to an icon; the Maximize button enlarges the VisSim window to fill your entire screen; and the Close button closes the VisSim window.

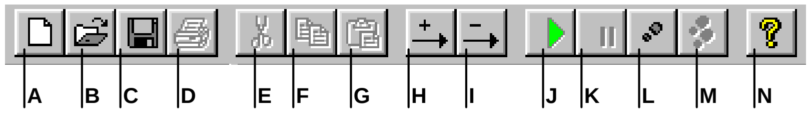
Menu bar: Lists the six basic menus available in VisSim: File, Edit, Simulate, View, Blocks, and Help. If you have installed a VisSim add-on, for example VisSim/Analyze, additional menus may appear on the menu bar. Clicking on a menu name displays a list of VisSim commands or blocks.

Tear-away toolbars: The buttons in the toolbars represent commonly used VisSim commands. To select a toolbar button, click on it.

Each cluster of buttons represents a tear-away toolbar. For example, the Main toolbar consists of the following buttons:



There are eight tear-away toolbars: Main, Sim Control, Annotation Blocks, Arithmetic Blocks, Boolean Blocks, Consumer Blocks, Producer Blocks, and User. By default, the Main and Sim Control toolbars appear when you start up VisSim. These toolbar buttons are described on the next page.



A	File > New command	**H**	Edit > Add Connector command
B	File > Open command	**I**	Edit > Remove Connector command
C	File > Save command	**J**	Simulate > Go command
D	File > Print command	**K**	Simulate > Stop command
E	Edit > Cut command	**L**	Simulate > Single Step command
F	Edit > Copy command	**M**	Simulate > Continue command
G	Edit > Paste command	**N**	Help command

As their names imply, the Annotation, Arithmetic, Boolean, Consumer, and Producer Blocks toolbars represent blocks in each of the corresponding categories. For descriptions of these blocks, see Chapter 12, “Block Reference.” The User toolbar allows you to create your own toolbar buttons. For more information, see page 265.

If a toolbar restricts your view of your work, drag on its background to move it to a new location, or click on its background to display a menu from which to close it. You can also use the View > Toolbar command to close toolbars. For more information, see page 264.

Dimmed toolbar buttons

Sometimes, when a toolbar button is dimmed, it is because the last cursor position was in the left window pane. Click the mouse anywhere in the right window pane to activate all available toolbar buttons.

Status bar: Provides simulation information about the current diagram, including the block count, simulation range, integration algorithm, step size, and implicit solver. When you run a simulation, the elapsed simulation time is also displayed.

Displaying menu command and toolbar descriptions in status bar

When you drag the mouse over a menu, menu command, or toolbar button, VisSim displays a brief description of the item in the status bar.

You can show or hide the status bar at any time using the View > Status Bar command.

Scroll bars: Pans the current viewing window. There are three ways to pan with the scroll bars. Click on the scroll arrows to scroll in small increments; click on the scroll bar to scroll in screen increments; or drag the scroll box to a location on the scroll bar that approximates a location in the block diagram.

You can show or hide the scroll bars at any time using the Edit > Preferences command, as described on page 264.

Auto-panning

Whenever you drag a block or draw a wire beyond the visible portion of the working area, VisSim will automatically scroll the work area.

Diagram tree: The VisSim window is divided into two panes. The left pane displays a *diagram tree*; that is, an outline of the diagram's compound blocks. At the top of the diagram tree is the Block Diagram icon, which represents the highest level of the currently opened block diagram. Its name appears next to the icon.

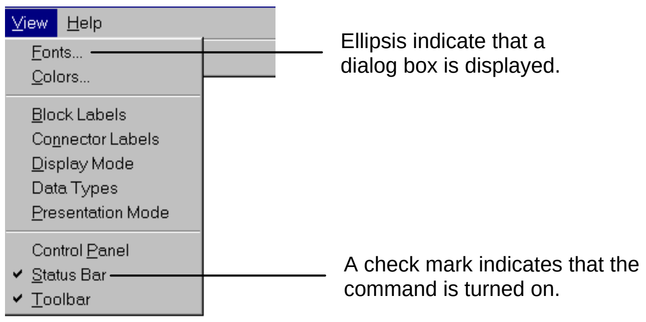
Beneath Block Diagram are the names of the compound blocks encapsulated in the block diagram. You can expand and collapse the diagram tree to display more or less detail by clicking on the plus or minus signs that appear next to the diagram and compound block names.

Whatever you select in the diagram tree is displayed in the right window pane. For example, if you select Block Diagram, the top level of the block diagram is displayed in the right window pane. You can jump to a specific compound block without wading through block diagram hierarchy by simply selecting the compound block name in the diagram tree.

If the diagram tree takes up too much space or if you cannot see all the hierarchical information in the tree, you can change its width by dragging its right edge.

Choosing commands

You can choose menu commands using the mouse or the keyboard. To choose a menu command with the mouse, click on the menu, then click on the command. To choose a menu command with the keyboard, press ALT to activate the menu bar, then press the key corresponding to the underlined letter in the menu, and finally press the key corresponding to the underlined letter in the command.



For commonly used menu commands, you can either:

- Press shortcut keys, which are listed on the menu to the right of the commands. For example, press CTRL+C to execute Edit > Copy.
- Press a corresponding toolbar button. For example, press ▶ to execute Simulate > Go.

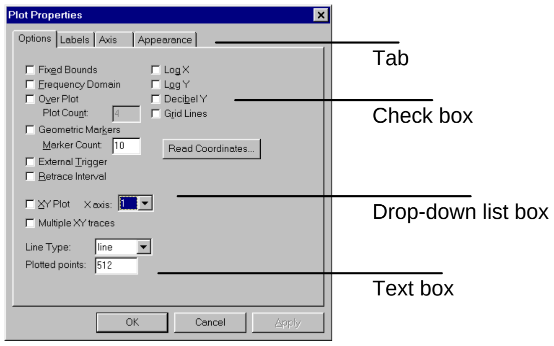
If a menu command is dimmed, it is unavailable for use.

Missing dialog boxes

If you choose a command with an ellipsis (for example, the File > Print command and the dialog box for the command is not displayed, click the mouse anywhere in the right window pane and re-select the command.

Using dialog boxes

VisSim uses dialog boxes to gather and display information about a command or block.

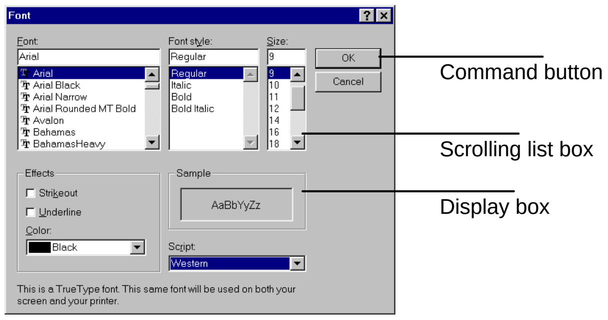


Tab: Allows similar options to be grouped together. When you click on a tab, a corresponding property sheet is brought to the front.

Check box: Sets or clears a particular option. When a ✓appears in the box, the option is activated.

Drop-down list box: Provides a list of several options. Click on the DOWN ARROW to select from a list of options.

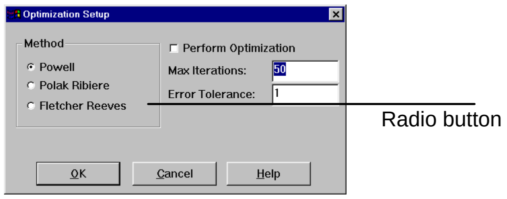
Text box: Allows you to enter text strings. Move the pointer over a text box until it changes into an I beam; then type in the text.



Command button: Causes an action to happen. Command buttons with ellipsis invoke another dialog box. Command buttons with a darkened rim are the default action. You can press the ENTER key to execute the default command button.

Scrolling list: Allows you to select from a list. Click on the scroll bar, scroll box, or scroll arrows to scroll through the list.

Display box: Provides a visual representation of your selection.

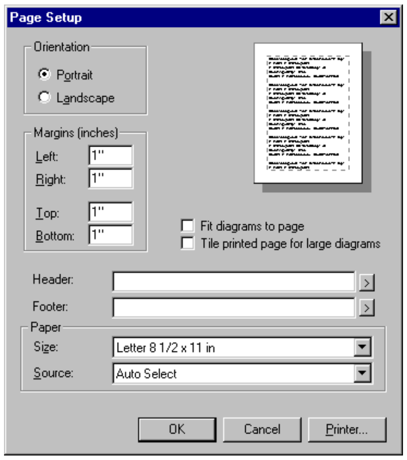


Radio button: Used to present two or more mutually exclusive options. You must pick one of the choices by clicking on it. When a black dot appears in the radio button, it is selected.

Creating and setting up a new block diagram

To open a new diagram, choose the File > New command, or click on 🗋 in the toolbar. If you're working on a different diagram and haven't yet saved your changes, VisSim prompts you to save them, then creates a new diagram. VisSim temporarily names the diagram *Diagram1*. The first time you attempt to save it, VisSim asks for a new name.

When you begin working on a new diagram, you usually start by setting up the page with the File > Page Setup command. As you choose options in the Page Setup dialog box, a sample of your selections is displayed in the top right-hand corner of the dialog box.



Orientation: You have the choice of Portrait or Landscape. Activate Portrait for a page that is taller than it is wide. If you want the opposite, activate Landscape.

Margins (Inches): The margins control the distance between diagram elements (blocks and wires) and the edge of the paper. VisSim does not display margins unless you are in print preview mode. In this mode, they appear as blue, nonprinting lines. Headers and footers, if specified, appear inside the margins.

Paper: Click on the DOWN ARROW in the Size box and select a standard paper size from the list; then click on the DOWN ARROW in the Source box to select the paper source (that is, the tray the printer uses to print the diagram).

Fit Diagram to Page: When Fit Diagram to Page is activated, VisSim prints each level of the block diagram on a separate page. When necessary, VisSim reduces diagram text so the level will fit on a single page within the specified margins. Because VisSim prints each level individually with the minimal reduction possible, the levels of a multi-level diagram may be sized differently.

VisSim may not be able to print extremely large block diagrams when Fit Diagram to Page is activated. In these cases, VisSim gives you the opportunity to abort the print operation. If you choose to continue printing, VisSim prints as much of the diagram as will fit on the page.

Tile Printed Page for Large Diagrams: Tile Printed Page for Large Diagrams causes VisSim to print each level using as many pages as necessary to print it without resizing. The margin settings are honored for each page.

Header and Footer: You can create headers and footers by entering text in the Header or Footer box. To view headers and footers, you must be in print preview mode. Headers and footers appear within the established page margins on each printed page of the diagram.

Using fields to enter header and footer information

By using fields, you can automatically insert information into a header or footer. For example, you can use fields to insert the file name of a diagram, the date the diagram was created, and so on.

To enter a field, click on the DOWN ARROW in the Header or Footer box and select one or more fields from the list. When you select a field, it appears as a field code in the Header or Footer box.

Field	**Field code**
File name	$f
File path	$F
Block path	$H
Date	$D
Integration method	$I
Optimization	$O
Page number	$p
Range	$G
Step size	$S
Left justify	$L
Center	$C
Right justify	$R

Using File > Print for page setup

You can use the File > Print command to reset the orientation, paper size, paper source, tiling, and fit-to-page options.

Opening an existing block diagram

You can easily open any of the last 12 block diagrams you worked on. When you click on the File menu, VisSim displays their names at the bottom of the menu.

To open any block diagram, choose the File > Open command. If another diagram is currently opened and contains unsaved changes, VisSim asks you if you want them saved before it closes the diagram and displays the File Open dialog box.

When you assign a title to a block diagram using the Diagram Information command, the title appears in the File Open dialog box when you select the block diagram.

▶ **To open a block diagram**

1. Do one of the following:
   - From the toolbar, choose .
   - Choose File > Open.
2. In the File Name box, type or select the name of the block diagram you want to open. If you do not see the block diagram you want, select a new drive or directory.
3. To open the block diagram for viewing only, activate the Read Only parameter. Although you can edit the block diagram, you must save the diagram under a new name to retain your edits.
4. Click on the OK button, or press ENTER.

Undoing an editing action

If you make a change to a block diagram then decide against the change, use the Edit > Undo command to erase it. If the Undo command is dimmed, the effect of the command cannot be undone.

Repainting the screen

Choosing Repaint Screen under the Edit menu redraws blocks and wires, and clears the screen of remnants left over from earlier VisSim manipulations.

Saving a block diagram

When you open a block diagram, VisSim reads the diagram into your computer's memory. As you work on the diagram, the changes you make are temporary. To make the changes permanent, you must save them to disk.

> ***Retaining diagram appearance on different graphic resolutions***
>
> If you activate Snap To Grid under Preferences in the dialog box for the Edit > Preferences command, VisSim saves block positions in units of ½ the average character size of the currently selected font. This results in a more consistent appearance of the block diagram over different graphic resolutions.

▶ **To save an existing block diagram**

- Do one of the following:
  - From the toolbar, choose 💾.
  - Choose File > Save.

You can use File > Save As to save the block diagram under a new name or to a different directory or device. This command comes in handy when you want to alter the current diagram but keep its original version.

Previewing before printing

Use the File > Print Preview command to display a block diagram as it will look when printed. Headers and footers, if specified, appear at the top and bottom of the pages according to the specifications established with the File > Page Setup command. Similarly, margins, if specified with File > Page Setup, are displayed in nonprinting, blue ink.

You can zoom in and out of the page using the Zoom buttons in the Print Preview toolbar.

Printing

The File > Print command lets you choose a printer and select printing options, such as the number of copies, the layers to be printed, and so on.

▶ **To print a block diagram on Windows**

- Do one of the following:
  - From the toolbar, choose 🖨.

- Choose File > Print.

▶ **To set printing options**

1. Do one of the following:
   - From the toolbar, choose 🖨.
   - Choose File > Print.
2. Do one or more of the following:

To print	Do this
Multiple copies	In the Copies box, enter a number.
The current level of the diagram	Under Print Range, activate Current Level.
The current level and below	Under Print Range, activate Current Level and Below.
All levels of the diagram	Under Print Range, activate All.
Each level of the block diagram on a separate page, and when necessary, reduce diagram text so the level fits on a single page	Activate Fit to Page. Because VisSim prints each level with the minimal reduction possible, the levels of a multi-level diagram may be sized differently. VisSim may not be able to print extremely large block diagrams when Fit to Page is activated, In these cases, VisSim gives you the opportunity to abort the print operation. If you choose to continue printing, VisSim prints as much of the diagram as will fit on the page.
Each level using as many pages as necessary to print it without resizing	Activate Tile Pages. The margin settings are honored for each page.
A version of the block diagram to a file to be printed at a later date or to be used in another program	Activate Print to File and then click on the OK button, or press enter. In the ensuing dialog box, specify a name for the block diagram you want to print.

Selecting a printer: The currently selected printer appears in the Printer box when you choose the File > Print command. To select a different printer, choose the Setup command button. If the printer you want to use is not listed, you must install the printer driver software on your system via the system Control Panel. To invoke the system Control Panel from VisSim, use the File > Printer & System Config command; then see the *Microsoft Windows User's Guide* for installation procedures.

Selecting additional printing options: You can also specify a paper size, orientation, and paper source for the printed diagram by clicking on the Setup command button. The selections you make here override selections you made previously with the File > Page Setup command.

Setting up the VisSim environment

You can customize VisSim to suit the way you work. You can, for example, use commands in the View menu to change the color of the work area, plotting area, and wires, specify diagram fonts, and switch between presentation modes. Other preferences are set with the Edit > Preferences command. These include coloring compound blocks, alternating between black and white, and color displays, and using training mode.

For more information on setting up the VisSim environment, see page 266.

Exiting VisSim

When you're finished working and decide to exit VisSim, use the File > Exit command, or press ALT+F4 to end your VisSim session. VisSim checks that all your work has been saved. If there are any unsaved changes, VisSim asks if you want them saved before exiting.

Chapter 2

Inserting, Setting Up, and Wiring Blocks

This chapter covers the following information:

- Inserting blocks
- Setting up block properties and initial conditions
- Wiring blocks
- Positioning, hiding, deleting, and coloring wires
- Using variables to pass signals
- Manipulating connector tabs

Block basics

In VisSim, you build system models in the form of block diagrams. Blocks are your basic design component. Each block represents a specific mathematical function. The function can be as simple as a sin function or as complex as a 15th order transfer function.

Types of blocks

VisSim offers over 90 blocks for linear, nonlinear, continuous, discrete-time, time varying, and hybrid system design. Blocks are categorized under the Blocks menu as follows:

- Animation
- Annotation
- Arithmetic
- Boolean
- DDE
- Integration
- Linear Systems
- Matrix Operations
- Nonlinear
- Optimization
- Random Generator
- Signal Consumer
- Signal Producer
- Time Delay
- Transcendental

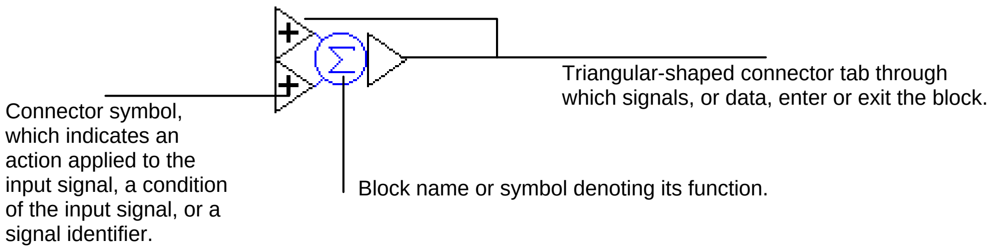
In addition, VisSim supplies three special-purpose blocks: `embed`, `expression`, and `userFunction`.

Custom blocks

If your design requirements extend beyond the blocks supplied by VisSim, you can create custom blocks in C, Fortran, or Pascal, as described in Appendix B, “Extending the Block Set.”

Identifying block parts

When you insert a block into a diagram, various symbols and text appear on it.


Triangular-shaped connector tab through which signals, or data, enter or exit the block.
Connector symbol, which indicates an action applied to the input signal, a condition of the input signal, or a signal identifier.
Block name or symbol denoting its function.

You can display additional information on the blocks in your diagram using the View > Block Labels command, as described on page 267.

Inserting blocks

You insert blocks into a diagram by selecting them from the Blocks menu and placing them in the work area. When you click on the Blocks menu, a list of blocks and block categories appears. Block categories are depicted by filled triangles (▶). When you click on a block category, a cascading menu appears listing the additional blocks.

▶ **To insert a block from the Blocks menu**

1. Choose Blocks from the menu bar.
2. Point to the block category and click the mouse. For example, point to Nonlinear and click the mouse to display the nonlinear blocks.
3. Point to a block and click the mouse. For example, point to `crossDetect` and click the mouse to choose the `crossDetect` block.

   The Blocks menu closes and a rectangular box appears with the pointer anchored in the upper left-hand corner of the box.
4. Point to the location in the diagram where you want to insert the block and click the mouse.

Setting up block properties and initial conditions

Most blocks have user-settable properties associated with them that allow you to set simulation invariant parameters of the blocks' functions. You define and change property values for a block through its Properties dialog box. When you change a property while the simulation is running, VisSim immediately updates the simulation to reflect the change. Initial conditions, which are supplied to the system at the start of a simulation, are also set in the blocks via their Properties dialog boxes.

▶ **To set up block properties**

1. Choose Edit > Block Properties.
2. Point to the block whose parameters you want to define or change and click the mouse.
3. In the Properties dialog box, enter or select the new parameter values and options, and then choose the OK button, or press ENTER.

Shortcuts for accessing Properties dialog boxes

A shortcut for accessing Properties dialog boxes for most blocks is to click the right mouse button over the block. For `button`, compound, `embed`, `label`, `userFunction`, and `variable` blocks, hold down the CTRL key while you click the right mouse button to access their Properties dialog boxes.

Entering numeric data

When entering numeric data, VisSim displays values greater than 10^6 or less than 10^{-6} in exponential notation. VisSim uses the letter "e" to indicate exponential notation; however, on input, it also recognizes the letter "E." For example, you can enter 6,000,000 in the following ways: 6e6 or 6E6.

Entering arithmetic expressions

Most numeric block parameters can be expressed using the arithmetic operators "+," "-," "*," "/" and the usual rules of precedence. For example:

2 * (5 + 4) = 18

2 * 5 + 4 = 14

Entering C expressions

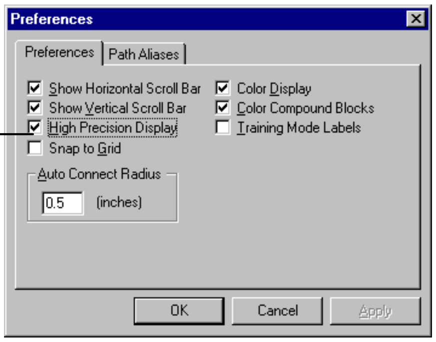
VisSim also recognizes C expressions for numeric data. This means you can build elementary mathematical functions using acos, asin, atan2, cos, cosh, exp, fabs, log, log10, pow, sin, sinh, sqrt, tan, and tanh. For example, if you enter pow (2,3) to the Gain parameter on the `gain` block, VisSim calculates 8. VisSim also interprets the universal constant *pi* as π.

Learning C

If you are unfamiliar with the C language and want to learn how to enter mathematical functions in C format, see *C: A Software Engineering Approach*, (P. Darnell and P. Margolis, Springer-Verlag, 1990).

Controlling the number of displayed significant decimal digits

Numeric block properties are always calculated in up to 15 significant decimal digits; however, you have the choice of displaying them in up to 6 or 15 significant decimal digits. The High Precision Display option under Preferences in the dialog box for the Edit > Preferences command controls the display.



Preferences

Preferences | Path Aliases

[✓] Show Horizontal Scroll Bar
[✓] Show Vertical Scroll Bar
[✓] High Precision Display
[ ] Snap to Grid
[✓] Color Display
[✓] Color Compound Blocks
[ ] Training Mode Labels

Auto Connect Radius: 0.5 (inches)

A check activates 15 significant decimal digit display.

Wiring basics

By wiring blocks together, VisSim is able to pass signals among blocks during a simulation. Signals are simply data. Input signals (x_n) represent data entering blocks; output signals (y_n) represent data exiting blocks.

Wireless transmission of signals

A `variable` block lets you name and transmit a signal throughout a block diagram without using wires. Typically, you use a `variable` block for system-wide variables or signals that would be laborious or visually messy to represent as wires. For more information, see page 129.

Types of wires

VisSim offers two types of wires:

- flexWires
- vector wires

A flexWire is a thin wire that allows a single signal to pass through it. A vector wire, on the other hand, is a thick wire that contains multiple flexWires. Typically, you use vector wires when performing vector or matrix operations, or to reduce wiring clutter at top-level diagram design.

The table below lists the blocks that accept vector wires:

Block category	Block name
Annotation	`index`, `scalarToVec`, `variable`, `vecToScalar`, `wirePositioner`
Arithmetic	`1/X`, `-X`, `*`, `/`, `abs`, `convert`, `gain`, `power`, `sign`, `summingJunction`, `unitConversion`
Matrix Operations	`buffer`, `dotProduct`, `fft`, `ifft`, `inverse`, `multiply`, `transpose`, `vsum`
Nonlinear	`case`, `merge`

You can manually bundle and unbundle flexWires using the `scalarToVec` and `vecToScalar` blocks.

Wiring rules

You attach flexWires and vector wires to blocks through their connector tabs. Once you have attached a wire to a block, VisSim maintains the connection even as you move the block around the screen.

When you wire blocks, the following rules are in effect:

- Wires can only be drawn between an input and output connector tab pair. The triangular shape of the connector tab lets you easily distinguish inputs from outputs.
- Input connector tabs can only have one wire attached to them; output connector tabs can have any number of wires attached to them. To change the number of connector tabs on a block, follow the procedures on page 23.
- If you draw multiple wires between two blocks, VisSim automatically skews them.

Wiring blocks together

▶ **To wire together blocks**

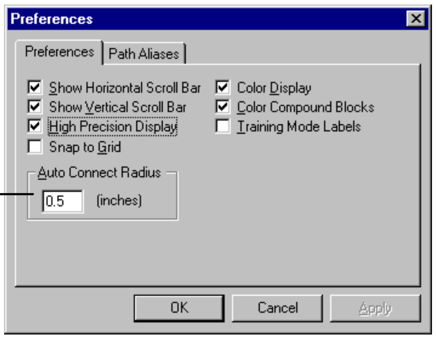
1. Point to a connector tab on one of the blocks to be wired. The pointer becomes an ⇧.
2. Hold down the mouse button and drag the pointer over the connector tab on the destination block.

As you drag the pointer, VisSim generates a colored line, which represents the wire. Because VisSim draws lines vertically and horizontally, the path of the line may not mimic the path of the cursor.

3. Release the mouse button.

Automatically completing connections

You can control how close the pointer must be to a connector tab to automatically complete a connection with the Auto Connect Radius option under Preferences in the dialog box for the Edit > Preferences command.



[✓] Show Horizontal Scroll Bar
[✓] Show Vertical Scroll Bar
[✓] High Precision Display
[ ] Snap to Grid
[✓] Color Display
[✓] Color Compound Blocks
[ ] Training Mode Labels

Auto Connect Radius: 0.5 (inches)

Positioning wires

Using `wirePositioner` blocks, you can perform a connect-the-dot method of wiring. That is, you insert `wirePositioner` blocks and then manually route the wire through them. Since you control the placement of the `wirePositioner` blocks, it's easy to draw a precise wiring path.

Additionally, because `wirePositioner` blocks do not take any additional computation time, you won't see a decrease in performance during a simulation.

Both flexWires and vector wires can by routed through `wirePositioner` blocks.

Coloring wires

By default, wires are drawn in black. You can, however, change the default color using the View > Colors command, as described on page 267.

You can also apply color to specific wires by assigning a connection class to the corresponding connector tabs. This wire coloring method is described on page 24.

Hiding wires

When you activate display mode with the View > Display Mode command, VisSim hides all wiring. Typically, you activate display mode when you want to display a control or instrumentation panel without the underlying connections, or when you want to view an animation.

Deleting wires

You delete a wire by detaching it from an input connector tab. Just point to the tab and hold down the mouse button as you drag the pointer away from the tab. When you release the mouse button, VisSim erases the wire.

Connector tab basics

All blocks that operate on signals have connector tabs. VisSim distinguishes between input and output connector tabs. Input connector tabs enable signals to enter a block; output connector tabs enable signals to exit a block. The triangular shape of the connector tab lets you easily see the direction in which the signals travel.

Some blocks have symbols on their connector tabs that indicate how the block acts on the data or the type of data the block is expecting. For example, the "-" on the `summingJunction` block means that the input is negated. See the descriptions of the individual blocks in Chapter 12, "Block Reference," for connector tab symbol definitions.

Adding and removing connector tabs

You can add or delete connector tabs on most VisSim blocks. If you delete a connector tab with an attached wire, the wire is also deleted.

Connector tabs on compound blocks

Because additional connector tabs are unconnected in compound blocks, make sure you verify the input and output connections after you complete this procedure.

▶ **To change the number of connector tabs on a block**

1. Do one of the following:
   - From the toolbar, choose +→ or −→.
   - Choose Edit > Add Connector or Edit > Remove Connector.

2. Do one of the following:

To	Do this
Add a connector tab	Point to where you want the tab. The short black line indicates tab placement. Then click the mouse.
Delete a connector tab	Point to the tab to be deleted. The selected tab has a short black line over it. Then click the mouse.

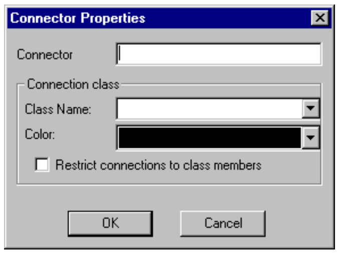
3. Repeat step 2 for as many tabs that you want to add or delete.
4. Click the mouse on empty screen space to exit this command.

Unconnected input connector tabs

Except on * blocks, all unconnected inputs are fed zeros, by default. Unconnected inputs on * blocks are fed ones.

Setting connection classes

Connection classes provide an easy method of organizing your calculations by name and color. You assign connection classes through the Connector Properties dialog box. To access this dialog box, double-click the mouse over a connector tab.



A class connection consists of a class name and corresponding color. The color is applied to the wire attached to the connector. For example, you can assign the class name PRESSURE to all connectors whose input and output signals relate to pressure calculations. All wires entering or exiting those connectors would then be displayed in the same color.

▶ **To assign a class**

1. Point to the connector tab to be classified. The pointer turns into an upward pointing arrow.
2. Double-click the mouse.

3. In the Class Name box, enter a name, or click on the DOWN ARROW to select an existing name.
4. In the Color box, click on the DOWN ARROW and select from the drop-down color list.
5. Click on the OK button, or press ENTER.

▶ **To change a class color**

Changing the color of all the connections in a particular class is simple to do in VisSim.

1. Point to a connector tab whose class color you want to change. The pointer turns into an upward pointing arrow.
2. Double-click the mouse.
3. In the Color box, click on the DOWN ARROW and select from the drop-down color list.
4. Click on the OK button, or press ENTER.

Restricting connections to class members

If you want to prevent wires from being drawn between connector tabs of different classes, activate the Restrict Connections to Class Members box in the Connector Properties dialog box.

Displaying connector tabs in a different view

The View menu's Presentation Mode, Display Mode, and Data Types commands have different affects on how connector tabs are presented. For information on these commands, see page 266.

Chapter 3

Arranging Blocks

This chapter covers the following information:

- Selecting blocks
- Moving and copying blocks
- Flipping blocks
- Aligning blocks
- Finding and replacing blocks
- Deleting blocks

Selecting blocks

Once you have inserted a block into a block diagram, you will probably have to select the block in order to manipulate it. When you select a block, VisSim highlights it in black and outlines it in white.

When you select a compound block, all encapsulated blocks are implicitly selected.

▶ **To select a block**

1. Point to the block.
2. Hold down the SHIFT key and click the mouse.

Area Selecting

A quick way to select one or more blocks is to use area select , which lets you draw a bounding box around the blocks you want to select. If any part of a block is contained in the bounding box, it is automatically selected.

▶ **To perform an area select**

1. Point to one corner of the area you want to select.
2. To anchor the corner, hold down the mouse button.
3. Drag the pointer until the box encloses all the blocks you want selected.
4. Release the mouse button.

Toggle selecting

This action automatically selects all unselected blocks at the current level, and unselects all selected blocks at the current level.

▶ **To toggle select blocks**

1. Point to empty screen space.
2. Hold down the SHIFT key and click the mouse.

Unselecting blocks

You can easily cancel the selection of individual blocks.

▶ **To unselect blocks**

1. Point to the selected block.
2. Hold down the SHIFT key and click the mouse.

When blocks are unselected they are returned to a normal video display.

Moving and copying blocks

Moving and copying blocks are common operations you'll perform in VisSim. Like many operations, there are several ways to move and copy blocks. For instance, you can move blocks by dragging and dropping them into place or you can cut them to the Windows Clipboard. From there, you can paste them back into your diagram or into another VisSim diagram. You can also paste them into other Windows-based applications.

Rules for moving and copying blocks

The following rules are in effect when you're moving and copying blocks:

- Moved and copied blocks retain the parameter values of the original blocks.
- Moved and copied blocks retain their internal wiring. This means that wires connecting blocks within the group of copied or cut blocks are retained.
- Moved and copied blocks lose their peripheral wiring. This means that wires connecting blocks in the group of blocks being copied or cut to other blocks are not retained.
- When moving or copying a compound block containing a global `variable` block with input, VisSim appends a number to the `variable` block name to keep it unique.

Drag-and-drop editing

An easy way to move or copy blocks within the current level of the diagram is with drag-and-drop editing. If you're moving or copying blocks to another level in the diagram, or to a different block diagram, you have to use the Edit menu's Cut, Copy, and Paste commands.

▶ **To move a single block using drag-and-drop editing**

1. Point to the block to be moved and hold down the mouse button.
2. Drag the block to the new location in the diagram.
3. Release the mouse button.

▶ **To move a group of blocks using drag-and-drop editing**

1. Select the blocks to be moved.
2. Point to one of the selected blocks and hold down the mouse button.

   The selected blocks are replaced with an empty box.
3. Drag the box to the desired location in the diagram.
4. Release the mouse button.

▶ **To copy a single block using drag-and-drop editing**

1. Point to the block to be copied.
2. Press CTRL+SHIFT while you simultaneously click the mouse.

   As you move the pointer away from the block, a box appears. The box shows where the copy will be placed.

3. Point to the location where you want the copy inserted and click the mouse.

If you want to copy a group of blocks, see the description below.

Copying, cutting, and pasting blocks

The Copy, Cut, and Paste commands use the Windows Clipboard to transfer blocks to another block diagram level or to a different block diagram. You can also use the Clipboard to paste blocks into other applications.

The Clipboard can only hold one selection of cut or copied blocks at a time. If you place a new selection in the Clipboard, it overwrites whatever was already there.

▶ **To copy or move selected blocks within VisSim**

1. Select the blocks.
2. To copy the blocks, do one of the following:
   - From the toolbar, choose .
   - Choose Edit > Copy.
   - Press CTRL+C.
3. To move the blocks, do one of the following:
   - From the toolbar, choose .
   - Choose Edit > Cut.
   - Press CTRL+X.

   At this point, the blocks are in the Clipboard.
4. Move to where you want the Clipboard contents inserted. If the location is in a different block diagram, use the File > Open command to open the proper block diagram and do one of the following:
   - From the toolbar, choose .
   - Choose Edit > Paste.
   - Press CTRL+V.

   A rectangular box appears.
5. Position the box where you want the Clipboard contents inserted.
6. Click the mouse.

If blocks and wires overlap as a result of this procedure, you can easily reposition them using drag-and-drop editing.

Copying blocks into other applications

You can use the Copy command to copy pictures of blocks into other Windows-based applications. Common Window elements (like title bars, control-menu boxes, scroll bars, and minimize and maximize boxes) are not copied when you use the Copy command.

Using the PRINT SCRN key to copy block diagrams

You can alternatively press PRINT SCRN to copy a picture of the entire VisSim window into the Clipboard. From there, you can paste it into another Windows-based application using the application's paste command.

Flipping blocks

By allowing you to flip blocks 180^{o}, VisSim can present a more logical representation of right-to-left signal flow. When you flip blocks, VisSim redraws all flexWires attached to the blocks.

▶ **To flip a block**

1. Select the blocks to be flipped.
2. Choose Edit > Flip Horizontal.
3. Click the mouse on empty screen space to unselect the blocks.

Aligning blocks vertically and horizontally

The Snap to Grid parameter under Preferences in the dialog box for the Edit > Preferences command forces blocks to stay on an invisible grid. When you create block diagrams where you want blocks to line up horizontally or vertically, or where you want them to be spaced equally, activate Snap to Grid. When you move a block with Snap to Grid active, the block is forced to the nearest grid point. Blocks that have been inserted into your block diagram before Snap to Grid is active are also affected by this parameter.

Finding and replacing blocks

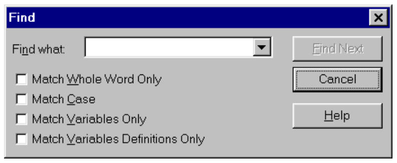
Using the Edit menu's Find and Replace commands, you can search for certain occurrences of blocks with user-defined names and text strings. These blocks include:

- `comment` blocks
- compound blocks
- `DDE`, `DDEreceive`, and `DDEsend` blocks
- `export` blocks
- `import` blocks
- `label` blocks
- `rtDataIn` and `rtDataOut` blocks
- `variable` blocks

Once you find what you're looking for, you can have VisSim change it to something else. VisSim searches the entire block diagram for the search item, regardless of your current location in the diagram.

Finding blocks

When you choose the Find command, VisSim displays a dialog box you can use to specify the block you want to find. If you want to search for variables, you can also click on the DOWN ARROW next to the Find What box and select a variable from the entries. All `variable` blocks in the diagram are listed in the drop-down list.



Once VisSim finds the search item, you can make a change in the diagram and then continue the search by choosing the Find Next button. The dialog box stays open so you can edit the diagram. To move the dialog box out of the way, drag on its title bar.

▶ **To find a block**

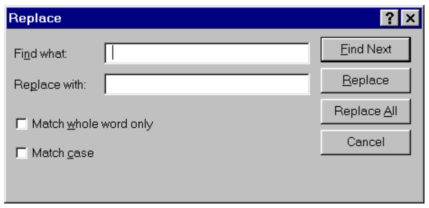
1. Choose Edit > Find.
2. In the Find What box, enter the search item. If you're searching for a `variable`, you can also click on the DOWN ARROW next to the Find What box and select from the variables list.
3. Select any option you want to control the search.

To	Select
Find whole words and not parts of words.	Find Whole Word Only box.
Find item with same capitalization as the word in the Find What box.	Match Case box.
Find only `variable` block names.	Match Variables Only box.
Find only the defining instance of a `variable`; that is, the `variable` block with an input connection.	Match Variable Definitions Only box.

4. Choose the Find Next button. When VisSim finds a match, it highlights the block in black.
5. To cancel a search or close the dialog box, choose the Cancel button.

Replacing blocks

You use the Replace command to replace the names of the blocks you find. You specify entries in the Replace dialog box in the same way that you do in the Find dialog box.



▶ **To replace a block**

1. Choose Edit > Replace.
2. In the Find What box, enter the search item.
3. In the Replace With box, enter the item to replace the search item.

4. Select any option you want to control the search.

To	Select
Find whole words and not parts of words.	Find Whole Word Only box.
Find item with same capitalization as the word in the Find What box.	Match Case box.

5. Choose the Find Next button. When VisSim finds a match, it highlights the block in black.
6. Do one of the following:

To	Select
Replace the block name and find the next occurrence.	The Replace button.
Change all occurrences without confirmation.	The Replace All button.
Leave the block name unchanged and find the next occurrence.	The Find Next button.

7. To cancel a search or close the dialog box, choose the Cancel button.

Deleting blocks

When your block diagram contains blocks you no longer need, you can delete them using the Edit > Clear command or the DEL key. When you delete blocks, all wires attached to the deleted blocks are also deleted.

▶ **To clear selected blocks**

1. Select the blocks to be cleared.
2. Choose Edit > Clear or press DEL.

Chapter 4

Setting Simulation Properties

The following information is covered in this chapter:

- Simulation range
- Simulation step size
- Real-time simulations
- Automatic simulation restarts
- Integration algorithms
- Minimum step sizes, maximum truncation errors, and maximum iteration counts for adaptive integration algorithms
- Checkpointing
- Propagating integer types
- Selecting frequency units
- Generating random numbers
- Notification messages for end-of-simulation, incomplete wiring, nonintegral clock, nonintegral delay, and numeric overflow

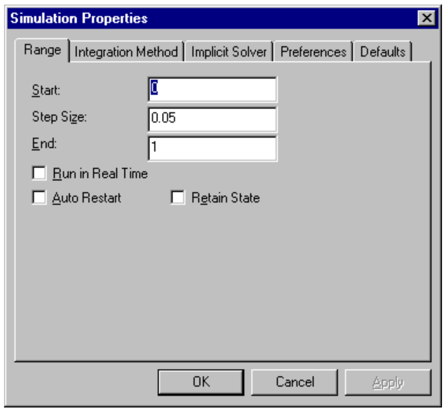
Setting up the simulation range

Setting up the simulation range involves choosing the start and end of the simulation, specifying the step size of the integration algorithm, indicating whether VisSim runs in real-time mode, and indicating whether VisSim automatically restarts the simulation either with or without the last known system states.

▶ **To access the Simulation Range options**

1. Choose Simulate > Simulation Properties.
2. Click on the Range tab.

The Range sheet in the Simulation Properties dialog box appears.



Start: 0
Step Size: 0.05
End: 1
☐ Run in Real Time
☐ Auto Restart
☐ Retain State

3. Choose the options you want, then click on the OK button, or press ENTER. (For more information on the options, see the descriptions below.)

Using the Range property sheet

The Range property sheet options are:

Auto Restart: For real-time control or training neural networks, where multiple data sets must be fed into VisSim repeatedly, you can activate Auto Restart. This parameter restarts and runs the simulation until one of the following conditions is met:

- The signal in the `error` or `stop` block goes to 1.
- You manually stop the simulation.

You can keep track of the number of the run by wiring a `$runCount variable` block into your diagram.

To retain the states of blocks each time VisSim automatically restarts a simulation, activate the Retain States parameter, as described below. Blocks that are time based (for example, Signal Producer blocks) are reset to their restart time. For a smooth transition between auto-restart simulation runs, you need to remove the Signal Producer blocks from your diagram. For instance, if your diagram contains a `sinusoid` block, replace it with an `integrator` block with its derivative set to the sinusoidal frequency and feed it to a `sin` block.

Retain States: For a smooth transition between simulation iterations, activate Retain State. When activated, VisSim retains the states of the `integrator`, `stateSpace`, `transferFunction`, and `unitDelay` blocks each time it restarts a simulation.

The Retain State parameter can only be activated when Auto Restart is already activated.

Run in Real Time: With Run in Real Time, VisSim simulates in real-time mode, which has the effect of retarding a simulation so that one simulation second equals one clocked second. This mode comes in handy when a system is exhibiting rapidly varying behavior. In real-time mode, the behavior can be slowed down and more easily analyzed.

Typically, you use real-time mode for hardware-in-the-loop control situations. For this, however, you also need the VisSim/Real-Time software and a PC D/A-A/D card. The VisSim/Real-Time driver lets you configure different analog and digital channels and insert them into a block diagram for reading and writing.

Start/End: Using Start and End, you can set independent variables that indicate when VisSim starts and stops a simulation, as well as when VisSim starts and stops logging data points in the Signal Consumer blocks wired into the block diagram.

You can also set defaults for the start and end, as described on page 41.

Step Size: The step size is the fundamental unit of integration. It indicates the interval at which the integration algorithm computes the integral of the input function and generates a data point in the Signal Consumer blocks wired into the block diagram. You specify the step size in the Step Size box. The default is 0.05.

For adaptive integration methods (adaptive Runge Kutta 5th order and adaptive Bulirsh-Stoer), you can also specify a minimum step size, as described on page 38.

Furthermore, you can also set a default step size for the non-adaptive integration methods, as described on page 41.

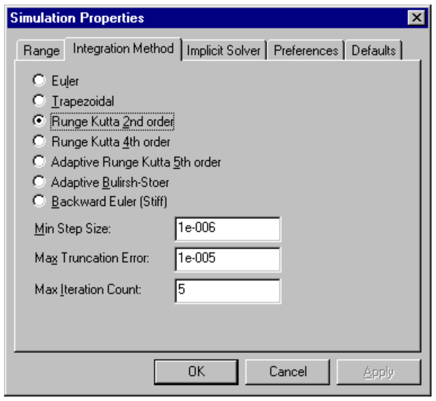
Setting up an integration method

Setting up the integration algorithm involves choosing the algorithm, and, if you choose an adaptive algorithm, specifying the minimum step size, error tolerance, and iteration count.

▶ **To access the integration method options**

1. Choose Simulate > Simulation Properties.
2. Click on the Integration Method tab.

The Integration Method sheet in the Simulation Properties dialog box appears.



Simulation Properties

( ) Euler
( ) Trapezoidal
(●) Runge Kutta 2nd order
( ) Runge Kutta 4th order
( ) Adaptive Runge Kutta 5th order
( ) Adaptive Bulirsh-Stoer
( ) Backward Euler (Stiff)

Min Step Size: 1e-006
Max Truncation Error: 1e-005
Max Iteration Count: 5

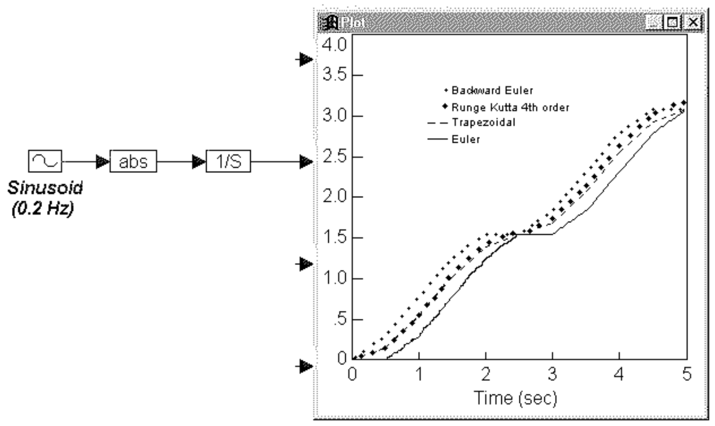
3. Choose the options you want, then click on the OK button, or press ENTER. (For more information on the options, see the descriptions below.)

Using the Integration Method property sheet

VisSim provides seven integration algorithms — Euler, trapezoidal, Runge Kutta 2nd order, Runge Kutta 4th order, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff) — of varying numerical accuracy for the numerical integration of differential and difference equations.

Each algorithm provides a numerical approximation to continuous integration. The approximation is based on a trade-off between speed of execution and accuracy. Generally speaking, the more complex algorithms yield more stable and numerically correct results; however, they also take longer to run.

For example, the integration of the absolute value of a sinusoid signal with a frequency of 0.2 Hz is plotted below. The output of the `abs` block is a sequence of sinusoid positive half-cycles with a frequency of 0.4 Hz. Since the simulation range is from 0 to 5 seconds, the output of the `integrator` block is the estimate area under the curve of two positive half-cycles.



While the differences due to the integration algorithms are negligible for this example, more dramatic differences can be observed when comparing simulation methods in diagrams containing differential equations.

A good rule of thumb, then, is to use the least complicated algorithm that provides stable and correct results. To achieve this, start with the most complex integration algorithm and work backwards to simpler algorithms until you see a noticeable change in your results.

Setting a default integration algorithm

If you plan on using a particular integration algorithm a lot, you can set it as the default, as described on page 41.

The Integration Method property sheet options are:

Euler: Evaluates once per simulation time step. This method is least affected by singularities, and is fastest for moderate step sizes.

Trapezoidal: Evaluates twice per simulation time step.

Runge Kutta 2d order: Obtains second order accuracy. This method uses a midpoint step derivative to calculate the final integration value. Specify the length of the step in the Step Size box.

Runge Kutta 4th order: Obtains fourth order accuracy. This method evaluates the derivative four times at each time step: once at the initial point, twice at sample midpoints, and once at a sample endpoint. The final integration value is then derived based on these derivatives.

Adaptive Runge Kutta 5th order: Obtains fifth order accuracy. This algorithm automatically takes small step sizes through discontinuities in the input function and large strides through smooth functions.

Adaptive Bulirsh-Stoer: Uses rational polynomials to extrapolate a series of substeps to a final estimate. This algorithm is highly accurate for smooth functions.

Backward Euler (Stiff): Obtains efficiency for systems with high and low frequencies. The other algorithms would require small step sizes to maintain stability.

Min Step Size: The adaptive Runge Kutta 5th order and adaptive Bulirsh-Stoer integration algorithms exert more control over the accuracy of the solution by letting you specify a minimum step size. The step size is continually adjusted in order to meet the error tolerance and iteration count criteria; however, it is never reduced below the minimum step size. Thus, inaccurate results may be produced if the minimum step size is too large, the error tolerance is too large, or the iteration count is too small.

The default value for the minimum step size is 1e-006.

Max Truncation Error: When you choose an adaptive integration algorithm, you can specify the maximum error between the results of two successive adaptive iterations. VisSim uses the truncation error to determine the adaptive step size. The larger the error you're willing to tolerate, the larger the step size. The default value for the maximum truncation error is 1e-005.

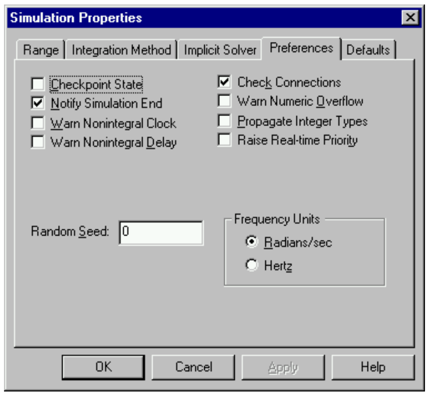
Max Iteration Count: When you choose an adaptive integration algorithm, you can also specify the maximum number of times the integration algorithm will vary its time step attempting to meet the maximum truncation error criterion. The default value for the maximum iteration count is 5.

Setting up simulation preferences

▶ **To access the preferences options**

1. Choose Simulate > Simulation Properties.
2. Click on the Preferences tab.

The Preferences property sheet in the Simulation Properties dialog box appears.



☐ Checkpoint State
✓ Notify Simulation End
☐ Warn Nonintegral Clock
☐ Warn Nonintegral Delay
✓ Check Connections
☐ Warn Numeric Overflow
☐ Propagate Integer Types
☐ Raise Real-time Priority

Random Seed: 0

Frequency Units
● Radians/sec
○ Hertz

3. Choose the options you want, then click on the OK button, or press ENTER. (For more information on the options, see the descriptions below.)

Using the Preferences property sheet

The Preferences property sheet provides options for checking for unconnected blocks, checkpointing a simulation, selecting the frequency units, and more. The property sheet options are:

Check Connections: Warns you at the start of a simulation if the diagram contains unconnected blocks. Unconnected blocks are highlighted in red. A Warning dialog box identifies the unconnected input tab and provides two choices:

- **Abort or Retry** Finishes checking the diagram and then stops the simulation.
- **Ignore** Finishes checking the diagram and then completes the simulation.

Checkpoint State: Saves a temporary copy of your block diagram "as is" at the time you stopped the simulation. To do so, Checkpoint States must be activated before you begin the simulation.

If you activated Checkpoint States, VisSim saves the current values of all system parameters and block outputs, and elapsed simulation time when you stop the simulation. If you close the block diagram and then re-open it, you can continue the simulation from where you left off.

Checkpointing is useful for long simulations because it allows you to stop and save a simulation in the event that you must shut down your computer for a lengthy period of time.

Frequency Units: Sets the frequency units to either radians per second or hertz.

Notify Simulation End: Broadcasts an “End of Simulation” message to your computer when the simulation completes.

Propagate Integer Types: Uses C semantics to propagate integer data types. For example, if you add two integers, the result is an integer value.

Raise Real-Time Priority: Gives your process the higher priority to let you achieve reliable real-time sampling without interruptions from other processes running simultaneously. This option is for real-time control applications.

Random Seed: Generates numbers by a random process. The `gaussian` and `uniform` blocks are affected by this option. That is, the numbers exiting these blocks are derived from the value of Random Seed.

Typically, you use Random Seed when an input is required to be unpredictable. For example, when modeling the descent path of an airplane, it is impossible to predict the force or direction of the wind. Consequently, you represent it as a function of a random number.

The value of Random Seed ranges from 0 to 65,536. The default is 0.

Altering the sequence of generated random numbers

VisSim generates the sequence of random numbers for each simulation differently depending upon whether the Auto Restart parameter in the dialog box for the Simulate menu’s Simulation Setup command is activated. When Auto Restart is on, VisSim generates a new sequence of random numbers for each simulation. Conversely, when it’s turned off, VisSim generates the same sequence of random numbers for each simulation. To change the sequence, you must explicitly enter a new Random Seed value at the start of the simulation.

Warn Nonintegral Clock: Warns you if a pulse is chosen that is not an integral multiple of the simulation step size. This option should always be activated; otherwise, a simulation that inaccurately represents the system you’re modeling may go undetected.

Warn Nonintegral Delay: Warns you if a delay is chosen that is not an integral multiple of the simulation step size. This option should always be activated; otherwise, a simulation that inaccurately represents the system you’re modeling may go undetected.

Warn Numeric Overflow: Warns you if a `convert` block causes data truncation resulting in the value loosing precision.

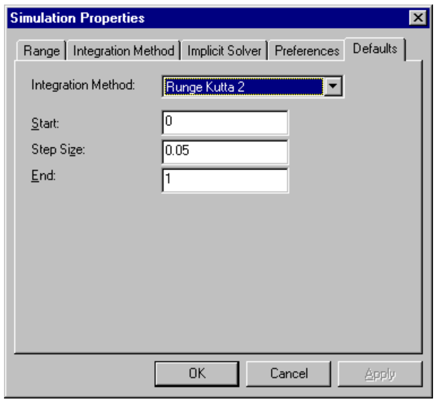
Setting simulation defaults

You can specify default settings for the range, integration algorithm, and fixed step size for non-adaptive integration algorithms that are in effect whenever you create a new block diagram or start a new VisSim session.

▶ **To set simulation defaults**

1. Choose Simulate > Simulation Properties.
2. Click on the Defaults tab.

   The Default property sheet in the Simulation Properties dialog box appears.



Integration Method: Runge Kutta 2
Start: 0
Step Size: 0.05
End: 1

3. Make the selections you want to put into effect.
4. Click on the OK button, or press ENTER.

Chapter 5

Simulating Block Diagrams

The following information is covered in this chapter:

- Types of system simulations
- Controlling a simulation with Simulate menu commands, toolbar buttons, and the Control Panel
- Dynamically modifying signal values
- Probing signal values
- Trimming a system
- Resetting error conditions
- Snapping system states
- Troubleshooting a simulation

Simulation basics

VisSim can simulate linear, nonlinear, continuous, and discrete systems. VisSim can also simulate systems containing both continuous and discrete transfer functions, as well as systems containing multi-rate sampling for discrete transfer functions.

When you initiate a simulation, VisSim first evaluates Signal Producer blocks, like `consts` and `ramps`, then sends the data to intermediate blocks that have both inputs and outputs, like `gains` and `summingJunctions`. Lastly, it sends data to Signal Consumer blocks that have only inputs, such as `plots` and `meters`.

VisSim simulates a system according to:

- Simulation parameters set in the dialog box for the Simulate > Simulation Properties command
- Initial conditions for the system set in the applicable blocks

If the status bar is turned on, VisSim displays current settings for the simulation range, step size, elapsed simulation time, integration algorithm, and implicit solver.

Continuous system simulation

Because integration is a more numerically stable operation than differentiation, you need to transform your ordinary differential equations into ones that use integration operators.

To enter an ordinary differential equation in VisSim, first algebraically solve the equation for the highest derivative. Then, in VisSim, insert the number of `integrator` blocks that equals the order of the highest derivative. Most continuous systems contain one or more differential equations. For example, if you're solving a third order differential equation, insert three `integrator` blocks and supply the equation for the highest order derivative as input to the first integrator block. The output of the first integrator is:

$$\frac{d^{n-1}x}{dt^{n-1}}$$

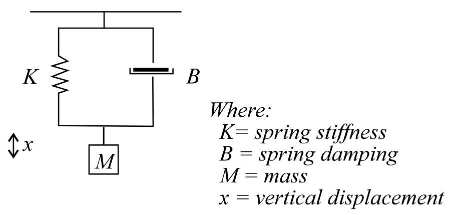
which is the next lower order derivative. The output of the second integrator block is:

$$\frac{d^{n-2}x}{dt^{n-2}}$$

and so on. The outputs of the lower order derivatives can be fed back into the calculation of the highest derivative.

Simulating a spring-damper arm

The following example steps you through the process of converting a second order differential equation into VisSim block diagram form. This example involves the ubiquitous damped harmonic oscillator, where a mass *M* is suspended from the ceiling by a spring-damper arm. The mass is attracted back toward the origin by an elastic restoring force proportional to its vertical displacement and is damped by an opposing force that acts in proportion to its velocity.



Based on Newton's Second Law, the definition of the equation of motion for the damped harmonic oscillator is:

$$M\frac{d^2x}{dt^2} = -Kx - B\frac{dx}{dt}$$

where:

$\frac{d^2x}{dt^2}$ = acceleration

$\frac{dx}{dt}$ = velocity

Because integration is inherently more numerically stable than differentiation, the equation must be expressed in terms of integrals. By definition of the derivative:

$$\frac{dx}{dt} = \int_0^t \frac{d^2x}{dt^2} + v(0)$$

and

$$x = \int_0^t \frac{dx}{dt} + x(0)$$

where:

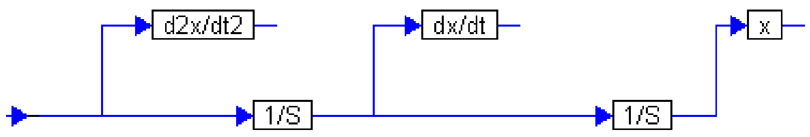
$v(0)$ = initial velocity of the mass

$x(0)$ = initial starting position of the mass

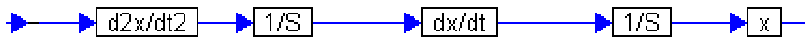
Employing 1/*s* as the operator for integration, and making the initial conditions implicit in 1/*s*, yields the following relationship:

$$x = \frac{1}{s}\left(\frac{dx}{dt}\right) = \frac{1}{s^2}\left(\frac{d^2x}{dt^2}\right)$$

The relationship can be expressed in VisSim block diagram form as:



A more concise representation of this relationship is:

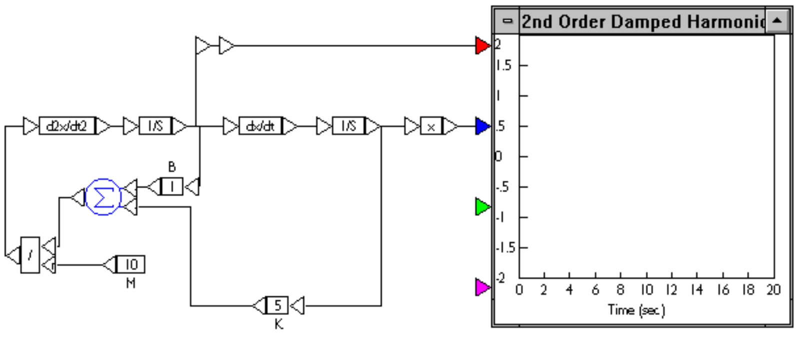


The three `variable` blocks hold the quantities d^2x/dt^2, dx/dt, and x at each instant of time. The `variable` blocks are actually extraneous, because the wires alone can carry the data forward to the next block.

Returning to the original equation of motion and solving for the acceleration yields:

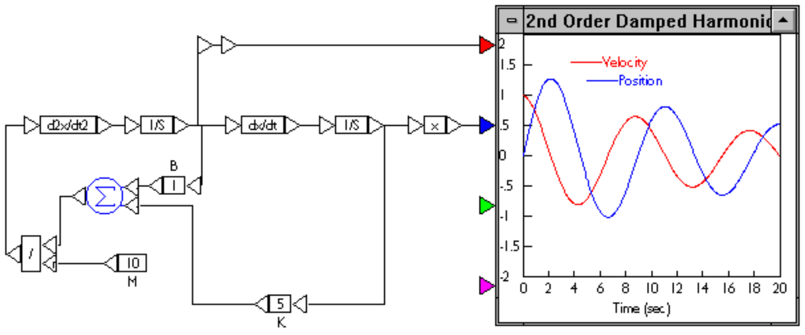
$$\frac{d^2x}{dt^2} = \frac{1}{M}\left(-Kx - B\frac{dx}{dt}\right)$$

To model this system in VisSim, wire the outputs of the *x* and *dx*/*dt* `variable` blocks through two `gain` blocks (which represent *K* and *B*) and into a `summingJunction` block, with inputs negated. By dividing the output of the `summingJunction` block by *M* (which is represented by a constant block), you produce d^2x/dt^2. Letting $K = 5$, $B = 1$, and $M = 10$, for example, results in the diagram shown on the next page.



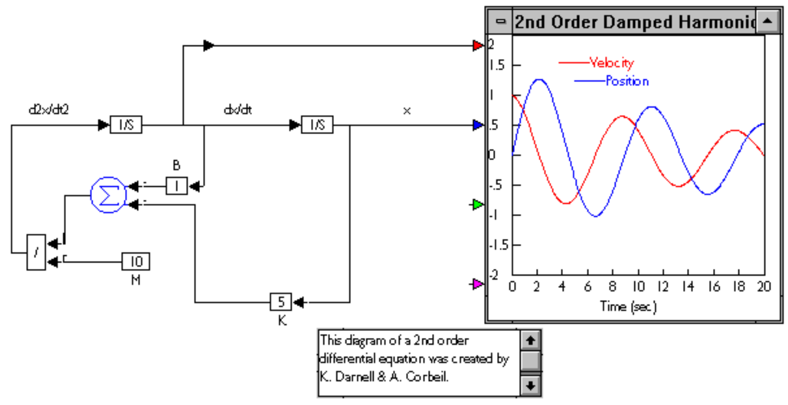
This diagram represents a closed-loop system from which the values for position, velocity, and/or acceleration can be displayed in a `plot` block, as was done here. The initial conditions of starting position *x*(0) and velocity *v*(0) of *M* are specified within the `integrator` blocks preceding the respective `variable` blocks.

Letting *x*(0) = 0 and *v*(0) = 1, and setting the simulation range from 0 to 20 and the step size to 0.05 yields the following results:



Note that the characteristic decay that is observed depends on the parameters *M*, *K*, and *B*. Different values for these quantities and initial conditions can be entered into the appropriate blocks to simulate any system.

You can simplify the diagram by replacing the `variable` blocks that denote *x*, *dx*/*dt*, and d^2x/dt^2 with optional `label` blocks.



This diagram of a 2nd order differential equation was created by K. Darnell & A. Corbeil.

Other physical effects can now be added, such as static and sliding friction, or external driving forces.

A coupled system can also be modeled by interconnecting two separate block diagrams. This permits extremely complex systems to be modeled without the need for a closed mathematical solution.

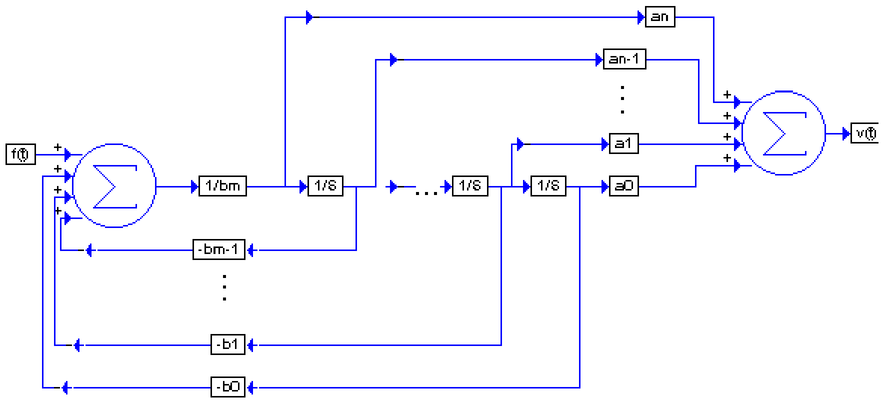
Entering continuous time transfer functions

A transfer function is a ratio of polynomials in the Laplacian *s* operator that models the ratio of the output signals divided by the input signals. There are two ways to enter a transfer function in VisSim. The more common method is via the `transferFunction` block, which you use when entering coefficients as numeric constants.

When the coefficients are polynomial constants, begin by defining the transfer function in operator notation. The transfer function should be proper; that is, the highest degree of the denominator polynomial *m* must be greater or equal to that of the numerator *n*. The general transfer function representation is:

$$\frac{N(s)}{D(s)} = \frac{a_n s^n + a_{n-1} s^{n-1} \ldots + a_1 s + a_0}{b_m s^m + b_{m-1} s^{m-1} \ldots + b_1 s + b_0}$$

You represent this in VisSim as follows:



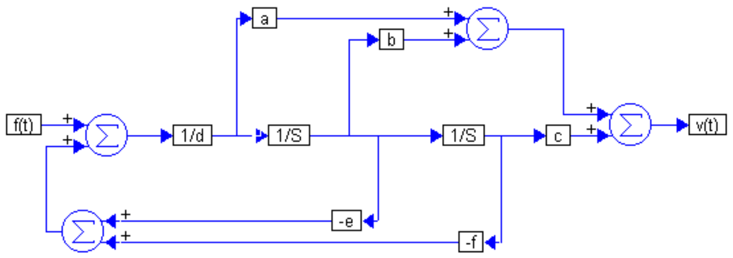
This diagram represents the condition in which the numerator and denominator degrees are equal (*m* = *n*). When the numerator is of a degree less than the denominator, the output paths are removed from the left. For example, if *n* is two less than *m*, the a_n and a_{n-1} output paths would be removed.

Note also that for each k^{th} polynomial term, you add an `integrator` and a corresponding a_k, b_k set of gains flow to the upper and lower `summingJunction` blocks in the diagram.

The second degree transfer function is:

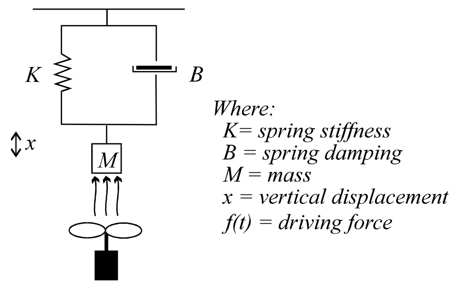
$$\frac{N(s)}{D(s)} = \frac{a\,s^2 + b\,s + c}{d\,s^2 + e\,s + f}$$

The diagram for this transfer function is:



Applying an external driving force to a spring-damper arm

By modifying the damped harmonic oscillator, created earlier, to include an external driving force *f*(*t*), you can create a system that contains a transfer function. An illustration of the modified system is shown below:



In this system, a vertical draft is produced by a strong fan positioned below the mass. This driving force sustains the motion of the damped oscillator and represents an input to the system. The output is the instantaneous velocity *v* of the mass. To derive the transfer function for this simple system, you will use the Laplace transform. The modified equation of motion for this system is:

$$M\frac{d^2x}{dt^2} = -\ Kx - \ B\frac{dx}{dt} + f(t)$$

Accounting for non-zero initial conditions, the Laplace transform becomes:

$$Ms^2x(s) - Msx(0) - M\frac{dx}{dt}(0) = -Kx(s) - Bsx(s) + Bx(0) + F(s)$$

Regrouping the equation yields:

$$x(s)\left(Ms^2 + Bs + K\right) - x(0)(Ms - B) - \frac{dx}{dt}(0)M = F(s)$$

Transfer function representation requires all initial conditions be equal to zero, specifically:

$$x(0) = \frac{dx}{dt}(0) = 0$$

The equation reduces to:

$$x(s)\left(Ms^2 + Bs + K\right) = F(s)$$

whose transfer function is:

$$\frac{x(s)}{F(s)} = \frac{1}{Ms^2 + Bs + K}$$

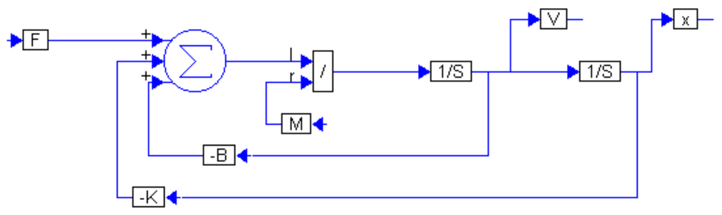
Since velocity rather than displacement is the desired output, the substitution:

$$V(s) = s\,x(s)$$

is made to produce the transfer function:

$$\frac{V(s)}{F(s)} = \frac{s}{Ms^2 + Bs + K}$$

The denominator remains unchanged; however, the numerator coefficients are different. The block diagram becomes:



The non-zero initial conditions can be easily included by specifying their values on each of the two `integrator` blocks. For example, suppose that the spring was initially stretched one inch. An initial condition of one would be placed on the rightmost `integrator`. Assuming an initial velocity of zero, the initial condition on the leftmost `integrator` would still be zero.

Discrete time system simulation

You can simulate models of discrete time systems using `unitDelay`, `transferFunction`, and `stateSpace` blocks. These discrete blocks have built-in samplers on their inputs and zero-order holds on their outputs.

You set the sample time of a `transferFunction` and `stateSpace` blocks in the dT parameters of their Properties dialog boxes. The dT parameter sets the sample time at which the blocks' states are updated. The `unitDelay` block has a Boolean clock at its input to set the sample time.

Simulating multi-rate systems: Discrete time systems in VisSim can be formulated as multi-rate systems. This means that a single model can contain blocks with different sampling rates. This capability is particularly useful in the simulation of discrete Multiple-Input-Multiple-Output (MIMO) systems. For a system with significant differences in its time constants in some natural modes or control loops, you can achieve improved performance by sampling different subsystems at different rates.

To specify multi-rate subsystems, use different sample times in the corresponding discrete `transferFunction` or discrete `stateSpace` blocks. The simulation time step must be set to a value equal to or less than the smallest value of all the sample times used in the discrete blocks.

Entering difference equations

A difference equation (DE) is similar to an ordinary differential equation, but instead of continuous functions, functions in a difference equation take on values only at discrete instances of time. Just as the operator in an ordinary differential equation is the integrator, the operator in the difference equation is the unit delay.

To understand how to represent a difference equation in block diagram form, consider the following example of the trapezoidal integration algorithm in difference equation form:

$$Y_k = Y_{k-1} + \frac{dt\left(R_k + R_{k-1}\right)}{2}$$

where:

$R = input$

$Y = output$

Here, *dt* is the fixed discrete update time and the subscript *k* and *k*-1 denote time in integer multiples of *dt.* Thus:

$$R_k = R\left(k\ dt\right)$$

and

$$R_{k-1} = R\left((k-1)dt\right)$$

A DE is converted to a transfer function in terms of the *Z* operator by replacing occurrences of F_{k-n} with $F\left(z^{-n}\right)$. Thus:

$$Y_k \to Y\left(z^0\right) \to Y$$

$$Y_{k-1} \to Y\left(z^{-1}\right)$$

$$R_k \to R\left(z^0\right) \to R$$

$$R_{k-1} \to R\left(z^{-1}\right)$$

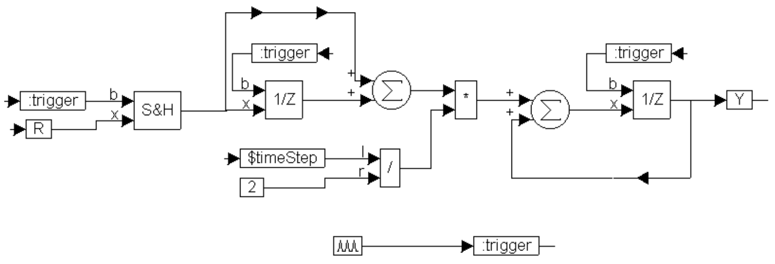
Performing the replacement and solving for $\frac{Y}{R}$ yields:

$$\frac{Y}{R} = \frac{dt}{2}\frac{1+z^{-1}}{1-z^{-1}}$$

Since transfer functions are conventionally expressed in positive powers of *z*, you must multiply the right-hand side of the equation by *z/z* to produce:

$$\frac{Y}{R} = \frac{dt}{2}\frac{z+1}{z-1}$$

To create a VisSim block diagram, the procedure is similar to that used for continuous time transfer functions. However, the `unitDelay` block replaces the `integrator` block. The resulting block diagram becomes:



The continuous input signal, R, is made a discrete function by passing it through a `sampleHold` block to effectively sample and hold its value every time the trigger is activated. The trigger is activated every *dt* seconds using the `pulseTrain` block, and must be fed into every `unitDelay` block to synchronize the VisSim data flow.

Hybrid system simulation

In VisSim, discrete and continuous time blocks can be used together in a model. Such systems are called hybrid systems. In hybrid systems, the outputs of the discrete blocks are held constant between successive sample times, and updated at times that correspond to the specified discrete sample time. The outputs of continuous blocks are updated at every time step. Similarly, the inputs to the discrete blocks are updated at times that correspond to the discrete time interval while the inputs to continuous blocks are updated at every time step.

Hybrid systems can also be multi-rate. To specify multi-rate subsystems, use different sample times in the corresponding discrete `transferFunction` or discrete `stateSpace` blocks. For hybrid system simulation, the simulation time-step must be set to a value equal to or less than the smallest value of all the sample times used in the discrete blocks.

Controlling a simulation

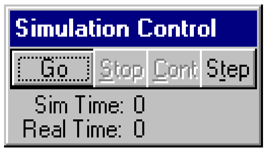
There are three ways to control a simulation:

- Using the Simulate menu Go, Stop, Continue, and Reset commands or corresponding toolbar buttons
- Using the simulation Control Panel
- Specifying simulation parameters on the VisSim command line, as described on page 263

Each way provides the same level of interactive control over the simulation. For short simulations, however, you may have to wire a `stop` block into the diagram if you plan on stopping and single stepping a simulation that has been started from the command line.

The Control Panel

The Control Panel provides fast and easy interactive control over a simulation.



Go, Stop, and Cont pushbuttons: These buttons allow you to start, stop, and continue a simulation. They are equivalent to the Go, Stop, and Continue commands in the Simulate menu, and the ▶, ⏸, and buttons in the toolbar.

Step pushbutton: This button allows you to single-step through a simulation. Each time you press the Step pushbutton, the simulation advances one time step. The Step pushbutton is equivalent to button in the toolbar.

Reset pushbutton: When you're single-stepping or proceeding normally through a simulation, the Go pushbutton is replaced with the Reset pushbutton. If you click on Reset, VisSim resets the system to its initial conditions.

▶ **To activate the Control Panel**

- Choose View > Control Panel.

Starting a simulation

▶ **To start a simulation**

- Do one of the following:
  - From the toolbar, choose ▶.
  - From the Control Panel, press the Go pushbutton.
  - Choose Simulate > Go command.

Stopping a simulation

▶ **To stop a simulation**

- Do one of the following:
  - From the toolbar, choose ⏸.
  - From the Control Panel, press the Stop pushbutton.
  - Choose Simulate > Stop.

Continuing a simulation

▶ **To continue a simulation**

- Do one of the following:
  - From the toolbar, choose .
  - From the Control Panel, press the Cont pushbutton.
  - Choose Simulate > Continue.

Single-stepping a simulation

▶ **To single step**

- Do one of the following:
  - From the toolbar, choose .
  - From the Control Panel, press the Step pushbutton.

Resetting a simulation to initial conditions

▶ **To reset**

- Do one of the following:
  - From the Control Panel, press the Reset pushbutton.
  - Choose Simulate > Reset.

More on controlling a simulation

Dynamically modifying signal values

You can dynamically modify a signal value during a simulation using the slider block. This block lets you set upper and lower bounds in one and 10 percent increments.

▶ **To modify a signal value**

1. Insert and wire a `slider` block into your diagram.
2. Using the scroll bar, adjust the value to be applied to the signal.
3. As the simulation proceeds, re-adjust the value of the `slider` block as necessary.

Probing signal values

There are two ways to probe signal values at each time step of a simulation:

To	**Do this**
Monitor signals entering or exiting a specific block	Hold down the right mouse button over a connector tab on the block.
Monitor signal values emitted from multiple blocks simultaneously	Wire `display` blocks to the output connector tabs of the blocks.

Trimming a system

VisSim's `unknown` and `constraint` blocks can be used to trim a simulation to begin at a desired non-zero point. This technique is especially useful for slow-running simulations in which the interesting region lies later on in the trajectory. By trimming the conditions at the interesting region, you save time.

The initial condition of the integrator can be set externally using a `summingJunction` block. (The actual initial condition on the integrator is set to 0.) The goal is to drive the derivative signal to zero on the first pass of the simulation by adjusting the value of the unknown blocks, which is the integrator initial condition.

Resetting error conditions

If a simulation fails as a result of a math fault — for example, a negative argument to a log function — VisSim displays a dialog box stating the nature of the error and highlights the offending block in red. To reset the error condition, point to the offending block and click the right mouse button. If the offending block is encapsulated within one or more compound blocks, each compound block is also highlighted in red. Note that you'll have to drill into highlighted compound blocks to find the offending block.

If multiple blocks contain errors, use the Edit > Clear Errors command to clear all the errors.

Snapping system states

When you snap states, VisSim overwrites the initial conditions of `unitDelay` blocks, `integrator` blocks, `resetIntegrator`, `limitedIntegrator`, `stateSpace`, and `transferFunction` blocks with their current output states, and renders their initial conditions irretrievable. Snapping states is useful when you want to run a simulation to a stable operating point and, from there, experiment with the system.

State values are saved in memory; to save them to disk, use the File > Save command.

▶ **To snap state values to memory**

1. Run the simulation to a specific point of interest.
2. Choose Simulate > Snap States.

Troubleshooting

What should I do when the input function to an integrator block contains discontinuities?

If the input function to an `integrator` block contains discontinuities, use the adaptive Runge Kutta 5th order or adaptive Bulirsh-Stoer integration algorithm.

How do I stabilize rapidly oscillating behavior?

A simulation that exhibits an oscillating behavior that increases rapidly in amplitude points to unstable integration settings. When this occurs, decrease the integration step size or switch to an integration algorithm that yields more accurate results and produces less accumulated errors over the course of the simulation, such as the adaptive Runge Kutta 5th order or adaptive Bulirsh-Stoer integration algorithm.

For highly nonlinear systems or stiff systems, you should use backward Euler.

How can I speed up my simulations?

When speed is a factor, disconnect all Signal Consumer blocks at the currently displayed level.

Is there an easy way to check for complete wiring?

A faulty simulation can be the result of incomplete wiring. VisSim automatically assigns zeros to all unsatisfied connector tabs (except on `variable` and `*` blocks) before it begins a simulation. To ensure that all blocks are fully connected, activate Check Connections in the dialog box for the Simulate > Preferences command. When this parameter is activated, VisSim warns you at the start of the simulation if the diagram contains unconnected blocks.

Why is my feedback loop causing errors?

If you create a feedback loop that does not contain integration or delay blocks, it is referred to as an *algebraic loop*. VisSim is not equipped to solve algebraic loops. Hence, during simulation, VisSim flags the loop-head block in red and issues a notification message. To fix the error, rework the loop to introduce a delay.

If you are trying to solve an implicit equation, see Chapter 7 “Solving Implicit Equations,” for information on using `unknown` and `constraint` blocks.

Chapter 6

Viewing Simulations

The following information is covered in this chapter:

- Displaying simulation data in customizable plots
- Displaying simulation data in customizable strip charts
- Using histograms, bar graphs, and needle gauges
- Creating animation

Plots

The `plot` block displays data in a two-dimensional time domain plot. You can customize the plot and control how data is presented through the Plotting Properties dialog box for the `plot` block.

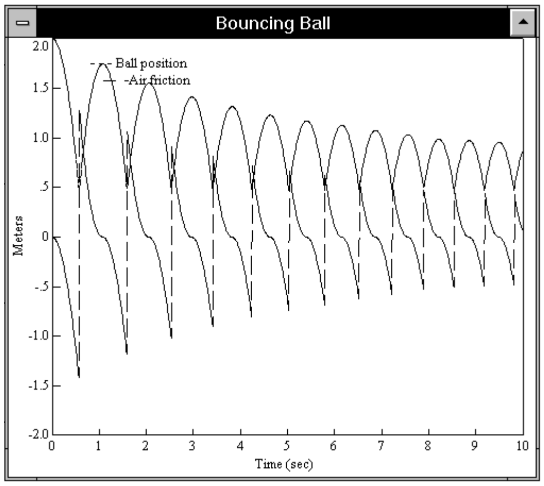
- Choose between XY or frequency domain
- Select logarithmic scaling, fixed axis bounds, or a time axis scale
- Display signal traces as individual data points, line segments, or *stepped* line segments
- Overlay signal traces with geometric markers
- Specify the number of data points to plot

- Use crosshairs and grid lines to determine data point coordinates
- Overlap plots

You can also save simulation data to file in .DAT, .M, .MAT, and .WAV formats.

Basic time domain plots

When you wire a `plot` block into your diagram and run a simulation, the simulation data is initially presented in time domain. All the signals are plotted on the Y axis; the X axis represents time. As data points arrive to be plotted, VisSim dynamically rescales plot bounds and connects the data points with line segments.



In the above plot, ball position and air friction are displayed as functions of time. The peak ball position follows an exponential decay, governed by air viscosity. The signals are distinguished by line patterns, a feature the `plot` block automatically performs when displaying or printing on monochrome devices. To make the plot more meaningful, signal labels and a title were also added.

Sizing a plot block

You might want to change the size or shape of a `plot` block for better viewing. You can expand it to full screen size with the Maximize button in the upper right-hand corner of the block, or you can drag the plot's borders or corners to adjust its size.

Zooming

You can zoom in on data points to view them at a magnified size and zoom back out to display them at their normal size. You can zoom in several times in a row for greater magnification.

If the area you're zooming in on does not contain at least one data point, the magnified area will be blank.

▶ **To zoom in**

1. Point to one corner of the area you want to select.
2. To anchor the corner, hold down the mouse button and CTRL key simultaneously.
3. Drag the pointer until the box encloses the area you want to magnify. A status box in the lower left-hand corner of the plot displays the pointer position.
4. Release the mouse button and CTRL key.

▶ **To zoom out**

- Hold down the CTRL key and click the right mouse button over the plot.

Changing plot properties

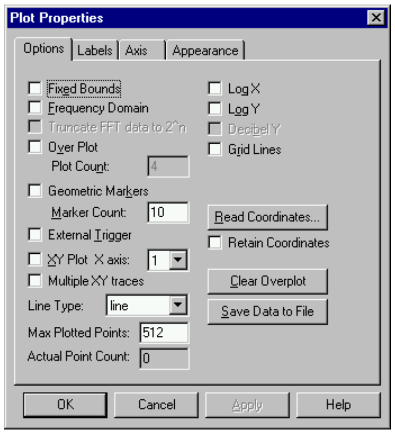
The Plot Properties dialog box controls how simulation data is presented.

▶ **To access the dialog box**

1. Choose Edit > Block Properties.
2. Click the mouse over the `plot` block.
3. Select the plotting parameters. (See the descriptions below for information about each parameter.)
4. Click on the OK button, or press ENTER.

Options property sheet

The Option property sheet lets you choose between XY and frequency domain plots; select logarithmic scaling, fixed axis bounds, or a time axis scale; display signal traces as individual data points, line segments, or stepped line segments; and more.



☐ Fixed Bounds
☐ Frequency Domain
☐ Truncate FFT data to 2^n
☐ Over Plot
Plot Count: 4
☐ Geometric Markers
Marker Count: 10
☐ External Trigger
☐ XY Plot X axis: 1
☐ Multiple XY traces
Line Type: line
Max Plotted Points: 512
Actual Point Count: 0
☐ Log X
☐ Log Y
☐ Decibel Y
☐ Grid Lines
☐ Retain Coordinates

Fixed Bounds: Specifies the region of the plot you want to view by letting you select the plotting bounds. When Fixed Bounds is activated, VisSim uses the values for the X Upper Bound, X Lower Bound, Y Upper Bound, and Y Lower Bound parameters in the Axis property sheet.

Frequency Domain: Obtains the frequency power spectrum through the use of the Fast Fourier Transform (FFT) algorithm.

Do not obtain frequency power spectrum data until after you have run a simulation. If you halt the simulation prematurely, the fidelity of the FFT is diminished.

Unexpected peaks

If your frequency domain plot produces unexpected peaks, check the simulation step size to verify that your sampling rate is adequate for obtaining accurate results. Then, based on the simulation step size and range, check the Plotted Points parameter to verify that you are indeed plotting each time step.

Truncate FFT Data To 2^n: When activated, this option truncates data down to the nearest power of 2. If you do not activate this option, the data buffer is padded with zeros to round up to the nearest power of 2.

This option can be turned on only when the Frequency Domain option is activated.

Over Plot, Plot Count, and Clear Overplot: When activated, Over Plot displays the results of multiple simulation runs in a single plot. This allows for better insight into how small changes can affect overall system performance.

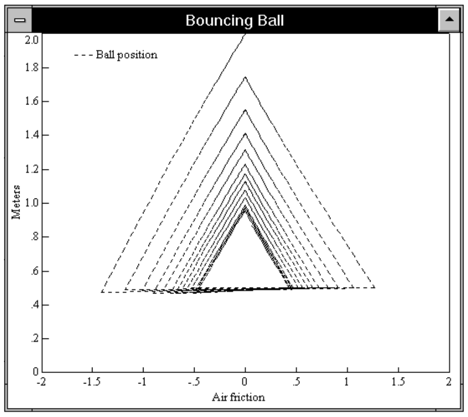
You can select the number of overlapping plots by entering a number into the Plot Count box. To clear all signal traces from a plot, click on the Clear Overplot button.

Geometric Markers and Marker Count: When activated, Geometric Markers overlays signal traces with geometric markers (squares, diamonds, circles, and triangles). These markers are particularly useful for monochrome displays and printers.

By default, VisSim overlays each signal trace with 10 markers; however, if this is not satisfactory, you can enter a new number in the Marker Count box.

External Trigger: Determines whether VisSim displays simulation data in the plot based on the value of an external trigger. When activated, External Trigger causes VisSim to place a round input connector on the `plot` block. When signal values entering the external trigger are 1, simulation data is plotted; when signal values entering the external trigger are 0, simulation data is not plotted.

XY Plot and X Axis: Together, XY Plot and X Axis let you use one input signal to represent X coordinate generation. As time advances, the remaining input signals are plotted relative to the X-axis signal.



In the XY plot above, ball position is plotted against air friction. At time 0, the ball position is at 2 and air friction is at 0. Over the course of the simulation, the ball position moves counter-clockwise, following a three-sided decaying cycle.

▶ **To specify an XY plot**

1. Activate the XY Plot parameter.
2. Under the X Axis parameter, choose the input signal to be used for X coordinate generation: 1 represents the input signal attached to the top input connector tab on the `plot` block, 2 represents the input signal attached to the second to the top input connector tab on the `plot` block, and so on.
3. Click on the OK button, or press ENTER.

▶ **To label the X axis on an XY plot**

In an XY plot, VisSim automatically labels the X axis with the label for the input signal used for X coordinate generation. For example, if you activate XY Plot and choose 2 under X Axis, VisSim uses the label assigned to input signal 2.

1. Click on the Labels tab.
2. Enter a label for the input signal you chose to be used for X coordinate generation. The Trace 1 box corresponds to 1 in the X Axis parameter, the Trace 2 box corresponds to 2 in the X Axis parameter, and so on.
3. Click on the OK button, or press ENTER.

Multiple XY Traces: Creates two independent XY plots, which allows two signals to be superimposed. The XY Plot option must be activated in order to use this option.

Line Type: Click on the DOWN ARROW and choose Point, Line, or Discrete.

- Point displays signal values as individual data points. The primary advantage of point plots is that you can see the separation of data as time advances.
- Line connects data points with solid line segments. On color displays, line segments are keyed to the color to their corresponding input connector tab. On monochrome displays and printers, VisSim automatically uses line patterns (solid, dot, dash, and dot-dash) to distinguish multiple signals. You may have to lower the point count in the Plotted Points parameter to allow enough room between data points for the pattern to be displayed. If this is not satisfactory, you can overlay signal traces with geometric markers.
- Discrete displays signal values as stepped line segments. In a discrete plot, VisSim holds the Y value constant from point to point. A discrete plot is helpful when data points are irregularly spaced and you don't know where the curve is accurate.

Plotted Points: Determines the smoothness and accuracy of a plot. The more data points you plot, the smoother and more accurate the plot. However, increasing the number of plotted data points also increases the time it takes to print and display the plot.

The maximum number of data points that can be plotted is 128,000. For Windows NT and Windows 95, the maximum number is 250 million.

Actual Point Count: Displays the number of data points plotted.

Log X and Log Y: Allow data to be plotted in logarithmic and semi-logarithmic coordinate systems. When you specify a logarithmic or semi-logarithmic plot, you cannot plot negative values on the log axis. Any negative value will be clipped to the low end of the scale. When neither parameter is activated, the plot defaults to linear.

To	Do this
Create a semi-logarithmic plot where the X axis is $\log_{10}$ and the Y axis is linear	Activate the Log X parameter.
Create a semi-logarithmic plot where the Y axis is $\log_{10}$ and the X axis is linear	Activate the Log Y parameter.
Create a plot using $\log_{10}$- $\log_{10}$ scales	Activate the Log X and Log Y parameters.

Decibel Y: Rescales the Y axis to display the values in decibels.

Grid Lines: Extends grid lines from the vertical and horizontal axis coordinates. Grid frequency — that is, the vertical and horizontal spacing of grid lines — is controlled by the spacing of the axis coordinates. VisSim automatically establishes reasonable axis coordinate spacing and hence controls the grid frequency.

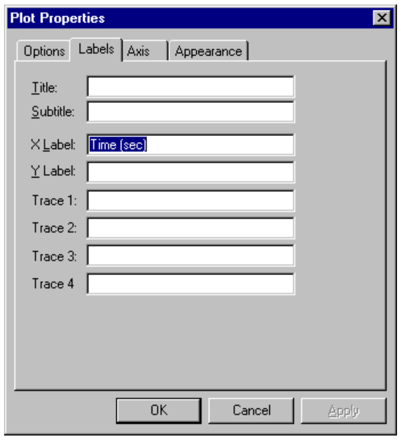
Read Coordinates: Overlays the plot with a set of crosshairs and displays crosshair position at the bottom of the plot.

When you click the left or right mouse button, VisSim freezes the crosshairs. Click the left mouse button again to erase the crosshairs.

Save Data To File: Opens the Select File dialog box to specify a file to which the plot data is to be saved. Click on the DOWN ARROW in the Files of Type box to choose a file format.

Labels property sheet

The Labels property sheet lets you name your plots, label the X and Y axes, and apply names to signal traces.



X Label: Time (sec)

Title and Subtitle: The Title and Subtitle parameters let you provide names for your plots. Titles and subtitles can be up to 80 alphanumeric characters. The title appears in the plot title bar; the subtitle is displayed in the top area of the plot. By default, plots are titled *Plot* and have no subtitles.

X Label and Y Label: The X Label parameter specifies a label for the X axis. To label the X axis on an XY plot, see the description of the XY Plot parameter on page 63. The Y Label parameter specifies a label for the Y axis.

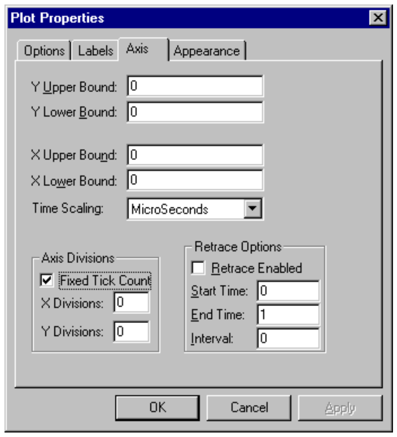
Axis labels can contain up to 80 alphanumeric characters.

Trace 1, Trace 2, Trace 3, and Trace 4: Let you specify labels for up to four input signals. The Trace 1 box corresponds to the top input connector tab, the Trace 2 box corresponds to the next lower tab, and so on.

Signal labels can contain up to 80 alphanumeric characters.

Axis property sheet

The Axis property sheet lets set upper and lower bounds for the X and Y axes, choose a time scale, specify axis divisions, and more.



Y Upper Bound: 0
Y Lower Bound: 0
X Upper Bound: 0
X Lower Bound: 0
Time Scaling: MicroSeconds

☑ Fixed Tick Count
X Divisions: 0
Y Divisions: 0

☐ Retrace Enabled
Start Time: 0
End Time: 1
Interval: 0

Y Upper Bound, Y Lower Bound, X Upper Bound, and X Lower Bound: Specify the upper and lower bounds for the X and Y axes. These bounds are in effect when you activate the Fixed Bounds parameter in the Options property sheet.

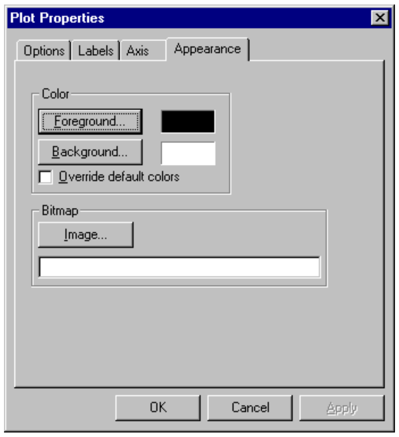
Time Scaling: Specifies X axis scaling in microseconds, milliseconds, seconds, minutes, hours, and days. When you select a different time axis scale, VisSim re-calculates the values in the X Upper Bound and X Lower Bound boxes. When you close the dialog box, the X axis is scaled to the time you chose.

Axis Divisions: You can override the plot’s grid tick division by activating Fixed Tick Count and entering values into the X Divisions and Y Divisions boxes. The numbers you enter indicate the number of grid ticks on each axis.

Retrace Options: Allows you to configure a plot as an *eye* diagram. Activate Retrace Enabled and specify the desired interval in the Interval box. In the Start Time and End Time boxes, enter the start and end times for the eye diagram. Eye diagrams are particularly useful for analyzing digital data waveforms.

Appearance property sheet

With the Appearance property sheet, you can add color and background patterns to your plots.



Color: Click on Foreground to color the axis labels and scaling text; click on Background to color the plotting area. Activate the Override Default Colors to override the color specified in the View > Colors command.

Bitmap: You can specify a bitmap image background for the plotting area. Type the file name directly into the Bitmap box or select one using Image command button. The specified bitmap image file overrides any background color selection.

Strip charts

The `stripChart` block displays up to four signals in a scrolling window. You define the display width and scrollable width of the window. To scroll back and forth through the window, use the horizontal scroll bar at the bottom of the `stripChart` block.

You can customize the strip chart and control how data is presented in the following ways:

- Choose frequency domain strip charts.
- Select Y axis scaling, fixed bounds, or a time axis scale.
- Display signal traces as individual data points, line segments, or stepped line segments.
- Overlay signal traces with geometric markers.

You can also save simulation data to file in .DAT, .M, .MAT, and .WAV formats.

Basic time domain strip chart

Like the `plot` block, a `stripChart` block initially displays data in the time domain. All signals are plotted on the Y axis; X axis represents time. As data points arrive to be plotted, VisSim dynamically rescales the plot bounds and connects the data points with line segments.

Sizing a stripChart block

To change the size or shape of the `stripChart` block for better viewing, drag the `stripChart's` borders or corners to adjust its size.

Printing a stripChart block

To print just a strip chart, click on the control-menu box in the upper left-hand corner of the `stripChart` and select the Print command. VisSim prints the strip chart in horizontal bands, with a maximum of four bands per page. VisSim uses as many pages as necessary to print all the data. VisSim also honors the margin settings specified by the File > Page Setup command.

Changing stripChart properties

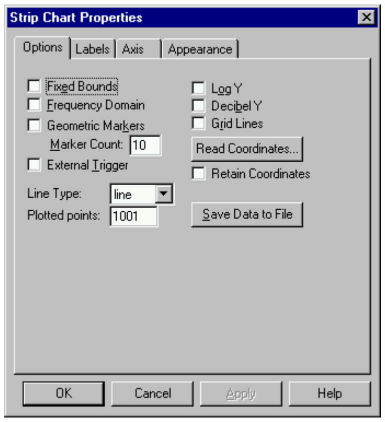
The Strip Chart Properties dialog box controls how simulation data is presented.

▶ **To access the dialog box**

1. Choose Edit > Block Properties.
2. Click the mouse over the `stripChart` block.
3. Select the strip chart parameters. (See the descriptions below for information about each parameter.)
4. Click on the OK button, or press ENTER.

Options property sheet

The Option property sheet lets you define the display width and scrollable width of the window; select fixed axis bounds, logarithmic Y scaling, or decibel Y scaling; activate frequency domain plotting; enable an external trigger; display signal traces as individual data points, line segments, or stepped line segments; and more.



Fixed Bounds: Specifies the region of the strip chart you want to view by letting you select the plotting bounds. When Fixed Bounds is activated, VisSim uses the values for the X Upper Bound, X Lower Bound, Y Upper Bound, and Y Lower Bound parameters in the Axis property sheet. Out-of-range signal values are clipped to the existing strip chart bounds.

Frequency Domain: Obtains the frequency power spectrum through the use of the Fast Fourier Transform (FFT) algorithm.

Do not obtain frequency power spectrum data until after you have run a simulation. If you halt the simulation prematurely, the fidelity of the FFT is diminished.

Unexpected peaks

If your frequency domain plot produces unexpected peaks, check the simulation step size to verify that your sampling rate is adequate for obtaining accurate results. Then, based on the simulation step size and range, check the Plotted Points parameter to verify that you are indeed plotting each time step.

Geometric Markers and Marker Count: The Geometric Markers parameter overlays signal traces with geometric markers (squares, diamonds, circles, and triangles). These markers are particularly useful for monochrome displays and printers.

By default, VisSim overlays each signal trace with 10 markers; however, if this is not satisfactory, you can change the number of markers with the Marker Count parameter.

External Trigger: Determines whether VisSim displays simulation data in the strip chart based on the value of an external trigger. When activated, External Trigger causes VisSim to place a round input connector on the `stripChart` block. When signal values entering the external trigger are 1, simulation data is displayed in the strip chart; when signal values entering the external trigger are 0, simulation data is not displayed.

Line Type: Click on the DOWN ARROW and choose Point, Line, or Discrete.

- Point displays signal values as individual data points. The primary advantage of point plots is that you can see the separation of data as time advances.
- Line connects data points with solid line segments. On color displays, line segments are keyed to the color to their corresponding input connector tab. On monochrome displays and printers, VisSim automatically uses line patterns (solid, dot, dash, and dot-dash) to distinguish multiple signals. You may have to lower the point count in the Plotted Points parameter to allow enough room between data points for the pattern to be displayed. If this is not satisfactory, you can overlay signal traces with geometric markers.
- Discrete displays signal values as stepped line segments. In a discrete plot, VisSim holds the Y value constant from point to point. A discrete plot is helpful when data points are irregularly spaced and you don’t know where the curve is accurate.

Plotted Points: Determines the smoothness and accuracy of a plot. The more data points you plot, the smoother and more accurate the plot. However, increasing the number of plotted data points also increases the time it takes to print and display the plot.

The maximum number of data points that can be plotted is 128,000. For Windows NT and Windows 95, the maximum number is 250 million.

Log Y: Enables a logarithmic Y axis. Note that you cannot plot negative values on a log axis. Any negative value is clipped to the low end of the scale.

Decibel Y: Rescales the Y axis to display the values in decibels.

Grid Lines: Extends grid lines from the vertical and horizontal axis coordinates. Grid frequency — that is, the vertical and horizontal spacing of grid lines — is controlled by the spacing of the axis coordinates. VisSim automatically establishes reasonable axis coordinate spacing and hence controls the grid frequency.

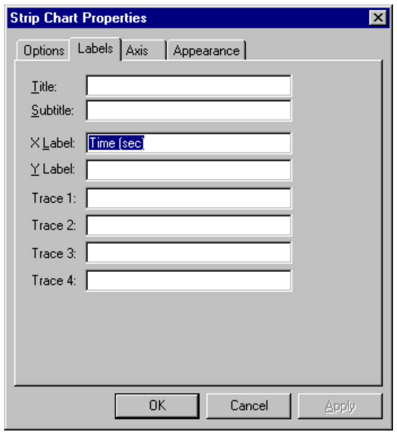
Read Coordinates: Overlays the strip chart with a set of crosshairs and displays crosshair position at the bottom of the chart.

When you click the left or right mouse button, VisSim freezes the crosshairs. Click the left mouse button again to erase the crosshairs.

Save Data To File: Opens the Select File dialog box to specify a file to which the strip chart data is to be saved. Click on the DOWN ARROW in the Files of Type box to choose a file format.

Labels property sheet

The Labels property sheet lets you name your strip charts, label the X and Y axes, and apply names to signal traces.



Title:
Subtitle:
X Label: Time (sec)
Y Label:
Trace 1:
Trace 2:
Trace 3:
Trace 4:

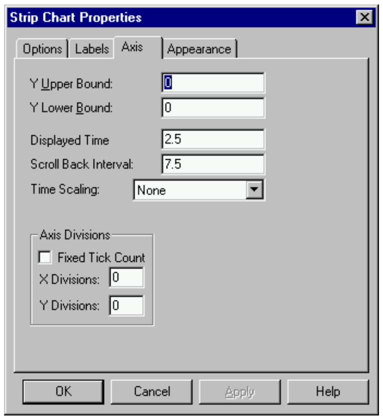
Title and Subtitle: The Title and Subtitle parameters let you provide names for your strip charts. Titles and subtitles can be up to 80 alphanumeric characters. The title appears in the plot title bar; the subtitle is displayed in the top area of the plot. By default, plots are titled *Strip Chart* and have no subtitles.

X Label and Y Label: The X Label and Y Label parameters specify labels for the X and Y axes. Axis labels can contain up to 80 alphanumeric characters.

Trace 1, Trace 2, Trace 3, and Trace 4: Let you specify labels for up to four input signals. The Trace 1 box corresponds to the top input connector tab, the Trace 2 box corresponds to the next lower tab, and so on. Signal labels can contain up to 80 alphanumeric characters.

Axis property sheet

The Axis property sheet lets set upper and lower bounds for the Y axis, choose a time scale, and specify axis divisions.



Y Upper Bound: 0
Y Lower Bound: 0
Displayed Time 2.5
Scroll Back Interval: 7.5
Time Scaling: None

Axis Divisions
☐ Fixed Tick Count
X Divisions: 0
Y Divisions: 0

Y Upper Bound and Y Lower Bound: Specify the upper and lower bounds for the Y axes. These bounds are in effect when you activate the Fixed Bounds parameter in the Options property sheet.

Time Scaling: Specifies X axis scaling in microseconds, milliseconds, seconds, minutes, hours, and days. When you select a different time axis scale, VisSim re-calculates the values in the X Upper Bound and X Lower Bound boxes. When you close the dialog box, the X axis is scaled to the time you chose.

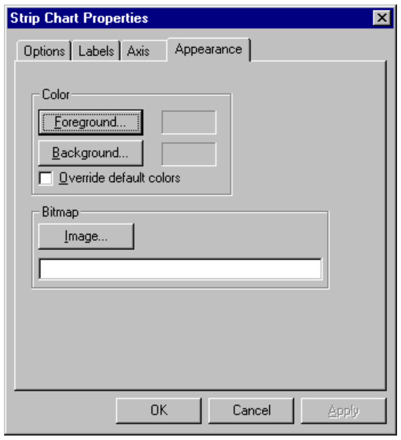
Axis Divisions: You can override the strip chart's grid tick division by activating Fixed Tick Count and entering values into the X Divisions and Y Divisions boxes. The numbers you enter indicate the number of grid ticks on each axis.

Displayed Time: Indicates the amount of units to be displayed in the strip chart window at any given time. The default value is ¼ of the total simulation time.

Scroll Back Interval: Indicates how much data (in X units) is saved for scrolling through. To conserve memory, keep this value low. To retain more data points, but use more memory, raise this value. The default value is the total simulation time.

Appearance property sheet

With the Appearance property sheet, you can add color and background patterns to your strip charts.



Color: Click on Foreground to color the axis labels and scaling text; click on Background to color the plotting area. Activate the Override Default Colors to override the color specified in the View > Colors command.

Bitmap: You can specify a bitmap image background for the plotting area. Type the file name directly into the Bitmap box or select one using Image command button. The specified bitmap image file overrides any background color selection.

Histograms

The `histogram` block shows how data are distributed over the course of a simulation. At each time step, a data point is placed in a bin that corresponds to a specific range. You can select the number of bins and the maximum and minimum bin value for the histogram. You can also select the maximum displayed bin height or have the `histogram` block dynamically rescale the bins as the data points arrive. The bins are spaced equally between the minimum and maximum bin values.

Sizing a histogram block

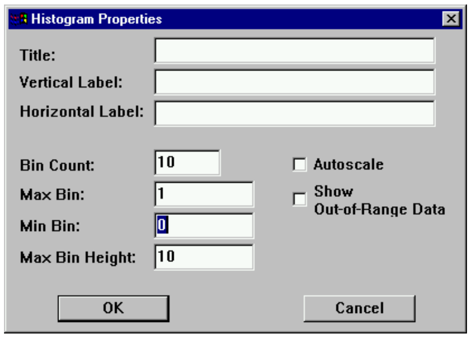
You might want to change the size or shape of the `histogram` block for better viewing. You can expand it to full screen size with the maximize button in the upper right-hand corner of the histogram or you can drag the histogram's borders or corners to adjust its size.

Changing histogram properties

The Histogram Properties dialog box controls how simulation data is presented.

▶ **To access the dialog box**

1. Choose Edit > Block Properties.
2. Click the mouse over the `histogram` block.
3. Select the plotting parameters. (See the descriptions below for information about each parameter.)
4. Click on the OK button, or press ENTER.



Title:	
Vertical Label:	
Horizontal Label:	
Bin Count:	10
Max Bin:	1
Min Bin:	0
Max Bin Height:	10
☐ Autoscale	
☐ Show Out-of-Range Data	

Title: Specifies a title for the histogram. The default title is Histogram.

Vertical Label: Specifies a vertical axis label.

Horizontal Label: Specifies a horizontal axis label.

Bin Count: Indicates the number of bins. If you change the bin count, the bin values are reset. The default is 10.

Max Bin: Indicates the maximum value of the data. The default is 1.

Min Bin: Indicates the minimum value of the data. The default is 0.

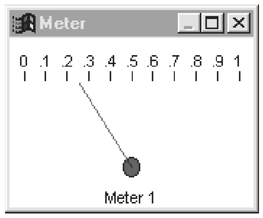
Max Bin Height: Indicates the maximum height of the bin. The default is 10.

Autoscale: Rescales plot when the maximum bin height is exceeded.

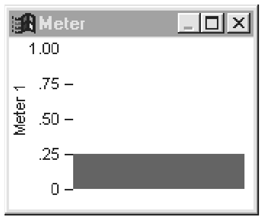
Show Out-Of-Range Data: Displays bins before and beyond the minimum and maximum bins to hold out-of-range data points.

Bar and needle graphs

The `meter` block displays signals in either a gauge- or bar-style display. Initially, the `meter` block appears as a gauge-style display with one input connector tab.

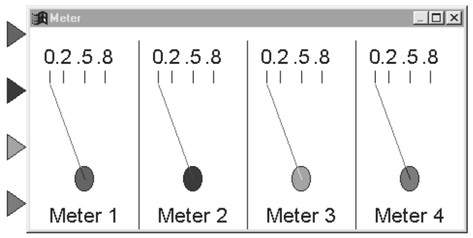


Gauge display



Bar display

You can display up to four signals in a `meter` block.



VisSim displays each signal in a separate meter window. The color of the input connector tab (red, blue, green, or yellow) corresponds to the bar (in a bar display) or bulb (in a gauge display) of the same color. You have the option of changing these colors.

The default number of input connector tabs is one. To change the number of input connector tabs, use the Edit > Add Connector and Edit > Remove Connector commands.

Sizing a meter block

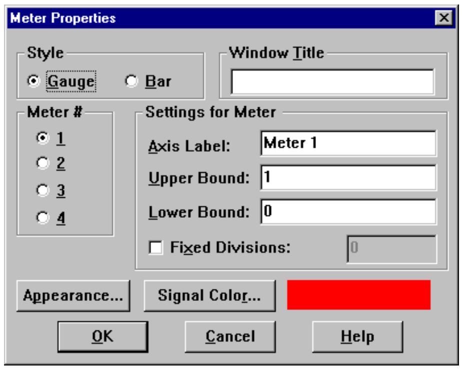
You might want to change the size or shape of the `meter` block for better viewing. You can expand it to full screen size with the Maximize button in the upper right-hand corner of the `meter` or you can drag the `meter's` borders or corners to adjust its size.

Changing meter properties

The Meter Properties dialog box controls how simulation data is presented.

▶ **To access the dialog box**

1. Choose Edit > Block Properties.
2. Click the mouse over the `meter` block.
3. Select the plotting parameters. (See the descriptions below for information about each parameter.)
4. Click on the OK button, or press ENTER.



Style: (● Gauge) Bar
Window Title:
Meter #: (● 1) 2 3 4
Axis Label: Meter 1
Upper Bound: 1
Lower Bound: 0
☐ Fixed Divisions: 0

Style: Switches between a bar and gauge display.

Meter #: 1, 2, 3, and 4 indicate the signal whose settings are to be examined or changed. The text in the Axis Label, Upper Bound, and Lower Bound boxes correspond to the selected signal.

Window Title: Indicates a title for the `meter` block. The title appears in the title bar that runs across the top of the `meter` block. The default title is *Meter.*

Axis Label: Indicates a name for the axis on which the signal is displayed. In a gauge display, the axis label is displayed horizontally across the top of the display; in a bar display, the axis label is displayed vertically along the left-hand side of the display.

Upper Bound and Lower Bound: Control the upper and lower bound of the meter display. The defaults are 1 and 0, respectively.

Fixed Division: Indicates the number of grid ticks.

Appearance: Opens the Appearance dialog box. Click on Foreground to color the axis label and scale text; click on Background to color the plotting area. The color you specify overrides the color specified in the View > Colors command.

You can alternatively specify a bitmap image background for the plotting area. Type the file name directly into the Bitmap box or select one using the Select Bitmap command button. The specified bitmap image file overrides any background color selection.

Signal Color: Opens the Color dialog box in which to specify a color for the input connector tab, the bulb and needle (in a gauge-style display), and the bar (in a bar-style display).

Creating animation

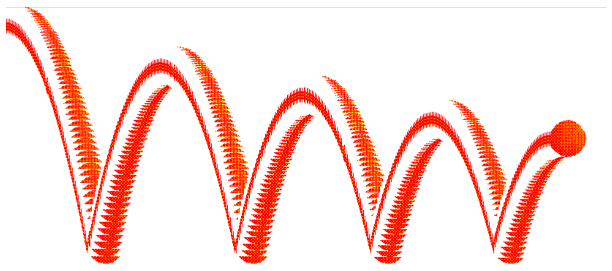
Animation is a series of images that, during a simulation, creates the illusion of movement. VisSim provides two blocks to create animations: the `animate` block, for animating an image, and the `lineDraw` block, for animating a line.

Animation basics

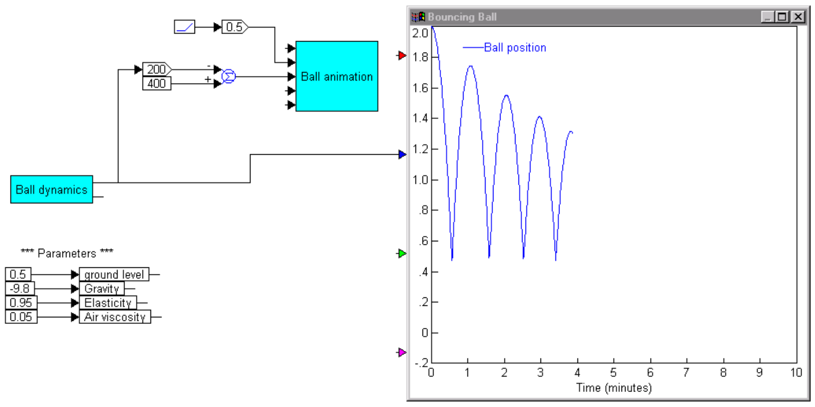
Animation occurs only when you initiate a simulation with display mode active. In this mode, all wires are hidden, all blocks are frozen in place, and with the exception of the interactive elements on `button` and `slider` blocks, block parameter values cannot be changed.

Animation occurs only when display mode is active. You activate display mode with the View > Display Mode command. A check mark in front of the command indicates that it is active.

Animation works by feeding signals into an animation block. The signals drive the coordinates of the animation block, which result in motion. For example, consider the bouncing ball animation, shown below:



The ball is represented as a single `animate` block. Movement is introduced by changing the ball’s *x*,*y* screen position coordinates. As the simulation progresses, the signals entering the `animate` block continually update its position. The diagram below drives the simulation of the bouncing ball.



To create the illusion of depth, you can vary the ball's *w*,*h* size coordinates.

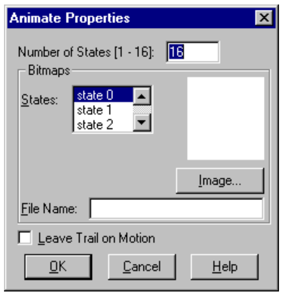
Using the animate block

The `animate` block lets you animate an image during simulation. Animation occurs through movement and changes in the size or appearance of the image.

The Animate Properties dialog box controls how animation data is presented.

▶ **To access the dialog box**

1. Choose Edit > Block Properties.
2. Click the mouse over the `animate` block.
3. Select the animation parameters. (See the descriptions below for information about each parameter.)
4. Click on the OK button, or press ENTER.


Animate Properties

Number of States [1 - 16]: 16

Bitmaps

States: state 0, state 1, state 2

File Name:

☐ Leave Trail on Motion

Applying pictures

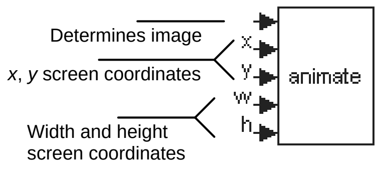
The pictures you apply to an `animate` block must be in bitmap file format (.BMP). VisSim supports Windows-formatted bitmaps with up to 256 colors.

▶ **To apply pictures to an animate block**

1. In the Number of Images box, enter then number of pictures to be applied to the block. You can have up to 16 different pictures
2. In the States box, select state 0.
3. Do one of the following:
   - Click on the Associate Bitmap command button. VisSim displays the File Open dialog box in which you can select a .BMP file to be associated with state 0. The .BMP file name appears in the File Name box.
   - In the File Name box, enter the complete pathname of the .BMP file to be associated with state 0.
4. To apply a second bitmap image to the `animate` block, select state 1 from the States box and repeat step 3. Continue to repeat steps 3 and 4, incrementing the state number, for each bitmap image you want to apply to the `animate` block.
5. Click on the OK button, or press ENTER.

Creating animation

Signals fed into the `animate` block drive the animation during simulation. The `animate` block accepts five input signals.


Determines image
x, *y* screen coordinates
Width and height screen coordinates
x
y
w
h
animate

Signals fed into the top input: The top input connector tab determines which image is applied to the `animate` block. An input signal value of 0 causes the bitmap image file corresponding to state 0 to be displayed; an input signal value of 1 causes the bitmap image file corresponding to state 1 to be displayed; and so on. Signal values entering the top input adhere to these rules:

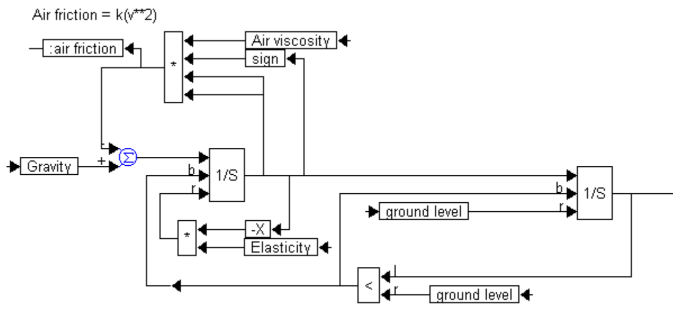
- When a signal value is a non-integer, the `animate` block truncates it to integer form.

- When a signal value is greater than the number of set states, the `animate` block uses the highest set state.
- When the signal value is negative, the `animate` block uses state 0.

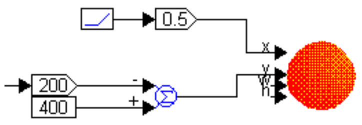
Signals fed into the x, y, w, and h inputs: The "x" and "y" connectors provide the *x,y* screen position coordinates for the image. The input connector tabs labeled "w" and "h" provide the width and height of the image. By varying these values, you can create movement and depth.

The values fed into the x, y, w, and h inputs represent display pixels. The *x, y* position (0,0) is the upper left corner of the VisSim window. Positive values extend to the right and down. For your image to appear on most video screens, keep its position within the bounds of a VGA screen (640x480).

You must perform all coordinate conversion manually. For example, the equations that determine the position of a bouncing ball are shown below:



However, before the output of the `resetIntegrator` block can be fed into the `animate` block, the position of the ball must be mapped to screen coordinates, as shown below:



The objective is to animate the bouncing of the ball as a function of time. This means that time is the independent variable, or the x axis. A `ramp` block is used to generate time, which ranges between 0 and 1000. A gain of 0.5 is used to scale time so that the amount of space used on the screen in the horizontal direction is limited in the range 0 to 500 pixels.

The ball position, *y*, varies from 0.5 to 2 in the simulation. This is scaled 0 to 300 pixels by using the scaling shown. Note that the pixel location (0,0) corresponds to the top left corner of your screen, and the largest pixel location is the bottom right corner of your screen.

Creating a trail: To leave a trail of the picture as the simulation progresses, activate the Leave Trail on Motion parameter.

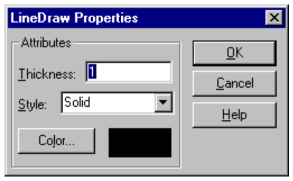
Using the lineDraw block

The `lineDraw` block lets you animate a line during simulation. You define the line by specifying two sets of *x*,*y* screen coordinate endpoints. You can also set the color, thickness, and style of the line.

The LineDraw Properties dialog box controls how a line is animated.

▶ **To access the dialog box**

1. Choose Edit > Block Properties.
2. Click the mouse over the `lineDraw` block. The lineDraw Properties dialog box appears.



LineDraw Properties

Attributes

Thickness: 1

Style: Solid

3. Do the following:

To specify	Do this
Color	Click on the Color command button and choose a color from the color palette.
Line style	Click on the DOWN ARROW for the Style box and select a style from the drop down list.
Line thickness	In the Thickness box, enter a value. Values are specified in points.

4. Click on the OK button, or press ENTER.

Creating line motion

Like the `animate` block, the `lineDraw` block uses the signal fed into its inputs to create motion. The top two inputs provide one set of *x*,*y* screen coordinate endpoints for the line; the bottom two inputs provide the other set. By varying these values, you can create motion.

The signal values fed into the inputs represent display pixels. Position (0,0) is the upper left corner of the VisSim window. Positive values extend to the right and down. For your line to appear on most video screens, keep its position within the bounds of a VGA screen (640x480).

Other ways to create animation

Animation can also be applied to a simulation using the `light`, `button`, and `bezel` blocks.

Use this block	To	For more information
`light`	Alternate among three images	See page 199
`button`	Alternate among 16 images	See page 168
`bezel`	Create operator control panels	See page 165

Chapter 7

Solving Implicit Equations

This chapter covers the following information:

- Setting up an implicit equation with `unknown` and `constraint` blocks
- Solving an implicit equation
- Using the Implicit Solver property sheet
- Sample implicit equation

Setting up an implicit equation

When a system contains an implicit equation, that is, an equation defined in terms of itself, you use `unknown` and `constraint` blocks to solve it. There may be one or more or no solutions for the system.

The key steps to setting and solving implicit equations follows:

1. Define the variable that needs to be determined as an unknown using the `unknown` block. The order is very important — an unknown must be defined first and then given a variable name.
2. Isolate zero on the right-hand side of the equation by moving all terms to the left-hand side.
3. Construct the left-hand side of the equation, and equate the right-hand side by using the `constraint` block to denote zero.

Algebraic loops

In the case of connections backward to earlier blocks already evaluated (often called feedback), VisSim checks to see that such feedback loops contain at least one `integrator`, `transferFunction`, `unitDelay`, or `timeDelay` block. If there is no such block in the feedback, the result is numerically ill-defined and is referred to as an algebraic loop. VisSim detects such algebraic loops and produces a warning message.

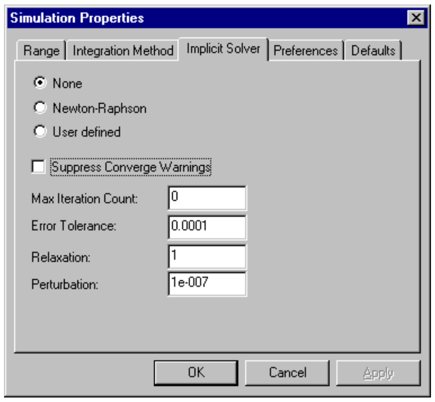
Solving an implicit equation

When solving an implicit equation, you can use the Newton Raphson solver or a custom solver. You can also set the error tolerance, maximum iteration count, perturbation, and relaxation parameters.

▶ **To solve an implicit equation**

1. Choose Simulate > Simulation Properties.
2. Click on the Implicit Solver tab.

   The Implicit Solver sheet in the Simulation Properties dialog box appears.



[✓ None] Newton-Raphson User defined

Suppress Converge Warnings

Max Iteration Count: 0

Error Tolerance: 0.0001

Relaxation: 1

Perturbation: 1e-007

3. Choose the options you want, then click the OK button, or press ENTER. (For more information on the options, see the descriptions below.)
4. Start the simulation.

Using the Implicit Solver property sheet

The Implicit Solver property sheet provides options for selecting the solver, suppressing convergence warnings, specifying the maximum iteration count, and more. The property sheet options are:

None: When you're not solving an implicit equation, activate None.

Newton-Raphson: Newton-Raphson is a singular value decomposition (SVD) based solver that performs static optimization at each time step. VisSim derives an *n*-dimensional slope by numerically perturbing the `unknown` outputs and observing the effects on the `constraints`. VisSim uses the slope matrix to compute values for the `unknown` blocks that drive the `constraints` to a minimum. Newton-Raphson is particularly useful for solving static equations in the presence of concurrent dynamics.

User Defined: When User Defined is activated, VisSim uses the DLL file named VSOLVER.DLL in your current directory to solve the equation. For information on creating a custom solver, see page 267.

Suppress Convergence Warnings: If you're solving for a set of roots that are equidistant from zero, you must initialize the `unknown` block to a value other than zero to force the equations to converge. To suppress the convergence warnings when solving an implicit system with nonlinear dynamics, activate Suppress Convergence Warnings.

Max Iteration Count: This option lets you specify the number of iterations the solver performs while attempting to meet the error tolerance criterion. The default value is 10.

Error Tolerance: This option lets you specify the maximum allowable difference in total error between two successive iterations. A total error is computed as the sum of individual errors squared, where the error signal is the input signal to a `constraint` block. Newton-Raphson ceases iterating when the difference in total error between two successive iterations becomes less than the tolerance.

Use Error Tolerance in conjunction with Max Iteration Count to control the time spent converging. The larger the tolerance, the quicker and less accurate the solution. The default value is 0.0001.

Relaxation: This option attenuates the iteration update value to attain convergence for equations that prove difficult to converge. As a side effect, it slows the convergence process because it forces the iteration to take smaller steps. The typical range is from slightly greater than 0 to 2. Select values less than 1 for systems that appear to be unstable. The default value is 1.

Perturbation: This option indicates the value by which the `unknown` blocks are numerically perturbed to evaluate the Jacobian (matrix of first partials). Each element of the Jacobian is a ratio of constraint change with respect to block perturbation value applied to the `unknown` blocks. The perturbation value should be at least one order of magnitude less than the `unknown` initial value, but greater than 1e-12 of the initial value for the `unknowns`. The default value is 1e-007.

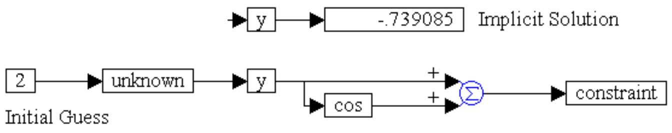
Implicit equation examples

Simple nonlinear implicit equation

Consider the equation:

$$y + \cos(y) = 0$$

This equation can be realized as:



In this configuration, the output of the `summingJunction`, namely *y* + cos(*y*) must equal zero. Further, from left to right, the entire diagram reads

> Starting with an initial guess of 2, find a value for the `unknown variable` called *y* such that *y* + cos(*y*) = 0.

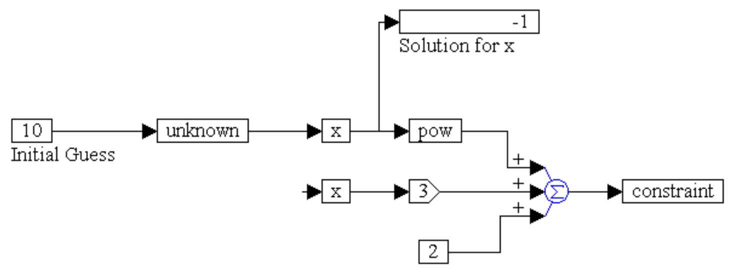
The result indicates that *y* = -0.739085 is one possible solution. Other solutions may exist and can be identified by using different initial guess values for the `unknown` block.

Advanced nonlinear implicit equation

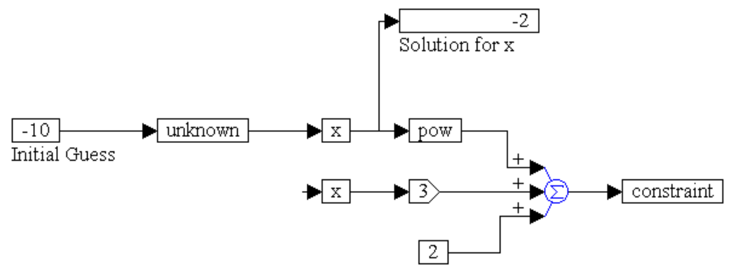
Consider the equation:

$$x^2 + 3x + 2 = 0$$

The roots of this equation can be obtained analytically as *x* = -2 and *x* = -1 for comparison. This equation can be solved implicitly in VisSim as shown on then next page.



In this configuration, the output of the `summingJunction`, namely $x^2 + 3x + 2$, must equal zero. The simulation yields one of the roots to be $x = -1$, as shown, starting from a guess of +10. When a guess value of -10 is used, the diagram becomes:



In this case, the second root, $x = -2$ is obtained.

This model shows that the solution obtained depends on the initial guess supplied to the `unknown` block. When the initial guess value is larger than -1, the solution converges to -1. When the initial guess value is smaller than -2, the solution converges to -2.

Chapter 8

Performing Global Optimization

This chapter covers the following information:

- Setting up global optimization with `cost` and `parameterUnknown` blocks
- Performing global optimization
- Using the Global Optimization dialog box
- Sample global optimization problem

Global optimization basics

Global optimization involves the automatic adjustment of system parameters to maximize or minimize a specified quantity, while satisfying one or more global constraints.

During global optimization, VisSim iteratively updates the parameter vector such that the cost function generally decreases until it finds a minimum. The resulting parameter values become the optimum values because they minimize the cost function.

Cost functions with many local minimum values

Most cost functions will have many local minimum values and although VisSim tries to avoid local minima, the one VisSim finds may not be the overall minimum. To be sure that it is the global minimum, you may want to perform several runs of the optimizer, using different initial `parameterUnknown` values.

Cost functions with no minimum values

It is possible that a cost function has no minimum, or has a flat surface away from the minima. In this case, the global optimizer will get confused and wander aimlessly (*how can you run downhill when there is no hill?*). If the optimizer appears to run for a long time with little convergence, you should suspect flat spots in your cost function. In such cases, you may have to reformulate the cost function such that it has at least one minimum. A common mistake is to put a `limit` block just before the cost input. In this case the optimizer will experiment with larger and larger unknown values to no avail. If a `limit` block is used in the cost function, it must be placed before an integration of total error.

Performing global optimization

Global optimization is almost always a nonlinear problem and rarely is there a single best method for minimizing the cost function. VisSim provides the three optimization methods: Powell, Polak-Ribiere, and Fletcher Reeves. You can alternatively write a custom optimizer, as described on page 270.

Regardless of the method you select, VisSim produces a sequence of parameter updates on a per-run basis that decreases the value of the cost function. The basic parameter update equation is:

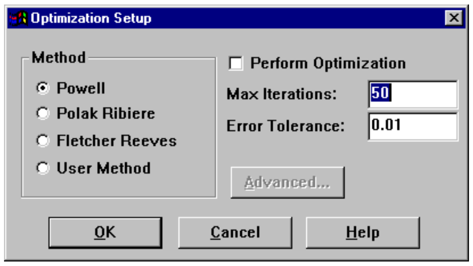
$$P_{k+1} = P_k + \Delta P_k \equiv \text{iteration index or VisSim run number}$$

The difference between each method is the way ΔP_k is generated. For more information on these methods, see *Numerical Recipes, The Art of Scientific Computing* (Cambridge University Press).

▶ **To perform global optimization**

1. Choose Simulate > Optimization Properties.

   The Optimization Properties dialog box appears.



Optimization Setup

Method
- [✓ Powell]
- Polak Ribiere
- Fletcher Reeves
- User Method

☐ Perform Optimization

Max Iterations: 50

Error Tolerance: 0.01

2. Activate the Perform Optimization option.

3. Choose the options you want, then click on the OK button, or press ENTER. (For more information on the options, see the descriptions below.)
4. Click on the OK button, or press ENTER.
5. Start the simulation.

Using the Optimization Properties dialog box

The Optimization Properties dialog box provides the following options:

Method: You can choice between three supplied optimizers or use a custom optimizer.

- **Powell:** A direction-set algorithm that typically runs faster because it does not explicitly calculate the gradient.
- **Polak Ribiere:** A conjugate gradient algorithm that is a bit more sophisticated than Fletcher Reeves for arriving at the supposed minimum of the quadratic form.
- **Fletcher Reeves:** Requires fewer iterations to convergence. This conjugant gradient algorithm is slower than Powell's method.
- **User Method:** When User Method is activated, VisSim uses the DLL file named VOPT.DLL in your current directory to solve the equation. For information on creating a custom global optimizer, see page 270.

Perform Optimization: This option must be activated to perform global optimization.

Max Iteration Count: Indicates the maximum number of iterations.

Error Tolerance: Indicates the maximum error between the results of two successive iterations. The default value is 10.

Global optimization examples

Optimized paper bag problem

Suppose you want to manufacture paper grocery bags at the lowest possible cost, where each bag has a volume of at least one cubic foot (1728 cubic inches). To minimize the cost of material, you must determine the optimal bag dimensions that minimize the amount of paper used for each bag, while ensuring that the volume of the bag ($v = whd$) is greater than or equal to 1728 cubic inches. To simplify the problem, additional material for folding and gluing the bag is ignored.

The cost function can be expressed as:

$s = 2(wh + dh) + dw$

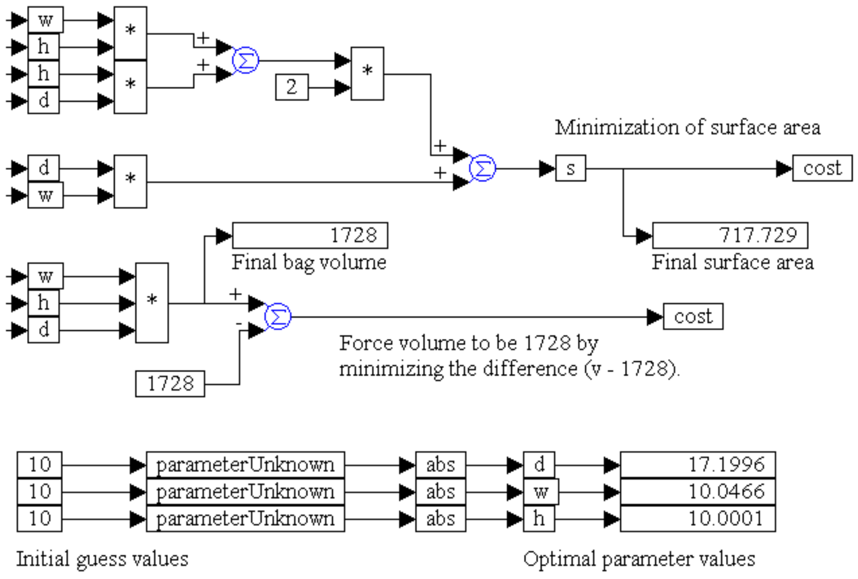
where s is the surface area of the bag, w is its width, h is its height, and d is its depth.

The cost function is also subject to a given volume constraint:

Volume $v = whd \geq 1728$ cubic inches

Though not explicitly specified, it is clear that each of the physical dimensions w, h, and d must be greater than zero.

To solve this problem in VisSim, construct the following block diagram:



Three `const` blocks, all set to 10, are used to produce the initial guess values. Their outputs are connected to three `parameterUnknown` blocks. To enforce the requirement that the physical dimensions of the bag are positive, the outputs of the `parameterUnknown` blocks are connected to three `abs` blocks, and the outputs of the `abs` blocks are connected to three `variable` blocks named *d*, *w*, and *h*. The outputs of the three `variables` are connected to `display` blocks to monitor the changes and the final values of the bag dimensions.

The outputs of *w*, *h*, and *d* are connected to a three-input `*` block, and the output of the `*` block is connected to a `summingJunction` and to a `display` block. A `const` block set to 1728 is connected to the other negated input of the `summingJunction`. A `cost` block, connected to the output of the `summingJunction` block, enforces the minimization of the difference *v* - 1728. This is equivalent to forcing the volume *v* to be very close to the desired value of 1728 cubic inches.

In order to minimize the surface area, the equation *s* = 2(*wh* + *dh*) + *dw* is coded using `variable`, `*`, `summingJunction`, and `const` blocks. The output of the final `summingJunction` block is connected to a `variable` *s*, which is connected to another `cost` block and to a `display` block.

Optimization is performed using the Powell method. The number of iterations is set to 50 and the error tolerance is 0.001. The optimization results indicate that a volume of 1728 cubic inches can be attained by using an ideal surface area of 717.7 square inches, with the final bag dimensions of *w* = 10.05, *h* = 10.00, and *d* = 17.20 inches. In comparison, the intuitive solution of *w* = *h* = *d* = 12 inches has a surface area of 720 square inches.

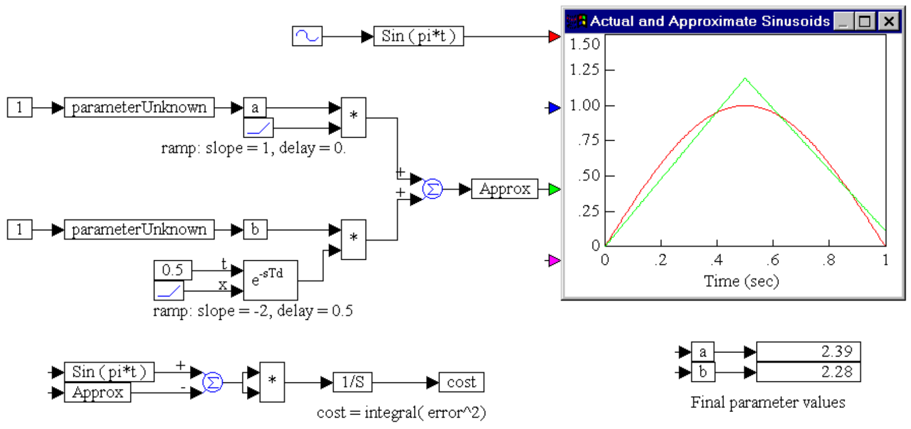
Two segment approximation of sin(π*t*)

Consider the problem of approximating a sinusoid sin(π*t*) in the range (0,1) by two straight line segments. In the range (0,0.5), the line segment has a positive slope, and in the range (0.5, 1.0), the second line segment has a negative slope. The approximation can be written as:

$$\sin(\pi t) \cong a\,r(t) - 2b\,r(t - 0.5)$$

where *a* and *b* are unknown weight factors, *r*(*t*) is a unit ramp that starts at *t* = 0, with a slope of +1, and -2*r*(*t* - 0.5) is a ramp that starts at *t* = 0.5 with a slope of -2.

The optimization problem in this case involves the determination of optimal values for the unknown weight factors *a* and *b*. This equation can be realized as:



In this configuration, two `const` blocks, both set to 1, provide the initial guess values to two `parameterUnknown` blocks. The outputs of the `parameterUnknown` blocks are connected to `variable` blocks *a* and *b*, thereby defining *a* and *b* to be unknown parameters. The output of *a* is connected to a `*` block. A `ramp` block, with an slope of 1 and no delay, is connected to the other input of the `*` block.

To generate a ramp of slope -2 that is delayed by 0.5 sec, a `const` block set to 0.5 is wired to the time delay input of a `timeDelay` block and a `ramp` block with an slope of -2 is wired to the main signal input of the `timeDelay`. The output of the `timeDelay` block is connected to one input of a `*` block and the other input is connected to the output of *b*.

The outputs of the two `*` blocks are connected to a `summingJunction`, and the output of the `summingJunction` is connected to a `variable` *Approx*. The output of a `sin` block set to an amplitude of 1 and a frequency of π radians/sec, is connected to a `variable` named *Sin(pi*t)*. The output of *Sin(pi*t)* and *Approx* are connected to a `plot` block to observe the results.

The `cost` function is constructed using a `summingJunction` to calculate the difference *Sin(pi*t) - Approx*. The output of the `summingJunction` is connected to both inputs of a `*` block to compute the squared error value. The output of the `*` block is connected to an `integrator` block to compute the integrated squared error, and the output of the `integrator` block is connected to a `cost` block, thereby defining the integrated squared error to be the cost or objective function that needs to be minimized by the optimization process. Two `display` blocks are connected to the outputs of *a* and *b* to observe the final values computed by the optimizer for the two weight factors.

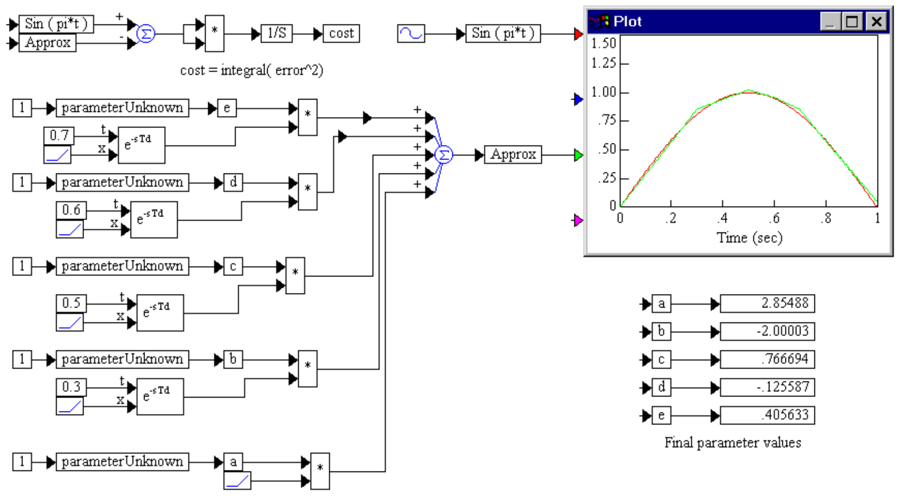
Using the Fletcher-Reeves optimization method, with maximum iterations set to 300 and error tolerance of 0.0001, the values of *a* and *b* are obtained as 2.39 and 2.28, respectively.

Five segment approximation of sin(πt)

By modifying the above problem, consider the usage of five line segments instead of two. The approximation can be written as:

$$\sin(\pi t) \cong a\,r(t) + b\,r(t-0.3) - 2c\,r(t-0.5) + 3d\,r(t-0.6) - 4e\,r(t-0.7)$$

This can be realized as:



The delayed versions of ramp signals are constructed using a `ramp` block with the correct slope connected to the main signal input of a `timeDelay` block, and a `const` block with the correct delay value connected to the time delayed input of the `timeDelay` block.

Five `parameterUnknown` blocks, with initial guesses set to 1, define the `variable` blocks *a*, *b*, *c*, *d*, and *e* to be unknown parameters. The outputs of the five `*` blocks are connected to a five-input `summingJunction` block. The output of the `summingJunction` is connected to a `variable` named *Approx*. The output of a `sin` block with an amplitude of 1, and frequency of π radians/sec is connected to a `variable` named *Sin(pi*t)*. The outputs of *Approx* and *Sin(pi*t)* are connected to a `plot` block.

The cost function is evaluated by computing the integral squared error. In this case, the optimization process yields the parameter values to be *a* = 2.85488, *b* = -2.00003, *c* = 0.766694, *d* = -0.125587, and *e* = 0.405633. From the `plot` block, it is also clear that the five-segment approximation is quite close to the original sinusoid.

Troubleshooting

How do I avoid system instability?

You should limit the cost calculation because, during optimization, some parameters may be supplied with values that drive the system into instability. The resulting large cost value can cause the optimization method to fail to converge due to the limited range of floating point numbers.

When limits are used, they must occur before the integration of the square of the error so that onset of saturation is numerically reflected in the cost function. In this way, onset of saturation is reflected in the cost value and gives the optimizer a slope to follow down.

What do I set the initial tolerance to when I know little about optimal parameter values?

Use an initial tolerance value of 10 in the Optimization Properties dialog box when you know very little about the optimal parameter values; otherwise, the algorithm will take a very long time to search a short distance in parameter space.

Once optimal values are found, the `parameterUnknowns` can be reinitialized with the new optimal values and the optimization can be rerun with a lower tolerance.

Though the algorithm tries to avoid local minima, to verify that the values found are optimal, run the optimizer with different initial values supplied to the `parameterUnknowns`.

Chapter 9

Designing Digital Filters

This chapter covers the implementation of:

- Time domain filters with tapped delay
- Time domain filters with transfer functions
- Frequency domain filters
- Interactive IIR and FIR filter design with the `transferFunction` block

Digital filter basics

A digital filter is a discrete time system that delivers an output, which is a modified version of its input.

Filters are the basic building blocks for most signal processing applications. They are typically used to extract or eliminate one or more constituent frequencies of an incoming signal.

Filters used for signal conditioning are usually designed from frequency response specifications, and are called frequency-selective filters. Frequency-selective filters operate by attenuating some frequency components of the input signal while allowing other components to pass through unchanged. For example, a low-pass filter attenuates all frequencies in the input signal that are above a specified frequency.

Filter operations

Filter operations can be represented mathematically by one or more difference equations. A general difference equation can be written as:

$$y(k) = \sum_{i=0}^{M} a_i x(k-i) - \sum_{j=1}^{N} b_j y(k-j)$$

This equation represents the relationship between the k^{th} sample of the output to the *N* previous values of the output, the *M* previous values of the input, and the current value of the input. If all the coefficients b_j are zero, the resulting filter is called a non-recursive or Finite Impulse Response (FIR) filter. Recursive filters are also known as Infinite Impulse Response (IIR) filters.

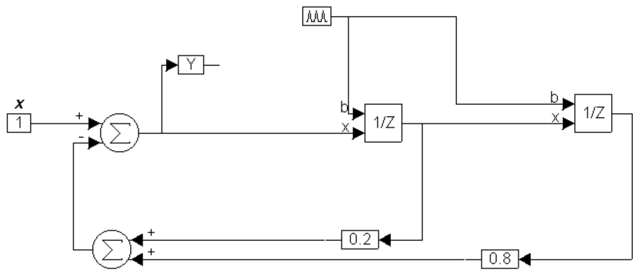
In FIR filters, the output is simply the weighted sum of the current and previous inputs. In contrast, in IIR filters, the output is the weighted sum of the current and previous inputs, and the previous outputs.

Time domain filters with tapped delay

Consider a filter described by the following recursive difference equation:

$y(k) = x(k) - 0.2y(k\text{-}1) - 0.8y(k\text{-}2)$

You can easily specify and implement this filter in time domain using `unitDelay` blocks. The filter input is $x(k)$ and the filter output is $y(k)$. The intermediate states are $y(k\text{-}1)$ and $y(k\text{-}2)$. The filter can be implemented as:



The Time Between Pulses parameter for the `pulseTrain` block must be greater than or equal to the simulation time step. An arbitrary value of 1 is assigned to input *x*.

Time domain filters with transfer functions

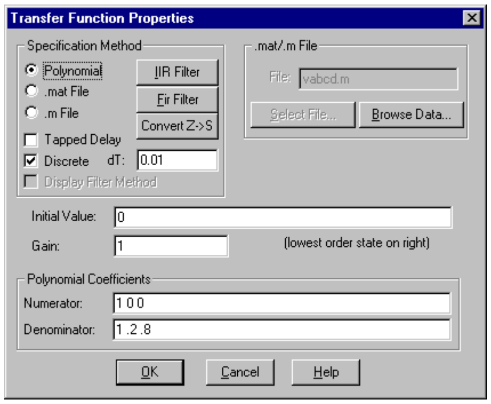
Filters can also be implemented in the time domain using the `transferFunction` block. For example, consider again the difference equation:

$y(k) = x(k) - 0.2y(k\text{-}1) - 0.8y(k\text{-}2)$

You can represent it in the form of a transfer function as:

$$\frac{Y(z)}{R(z)} = \frac{z^2}{z^2 + 0.2z + 0.8}$$

You can then implement the transfer function in block diagram form using the `transferFunction` block:


Transfer Function Properties

Specification Method: Polynomial [✓] / .mat File / .m File
IIR Filter | Fir Filter | Convert Z->S
Tapped Delay [ ]
Discrete [✓] dT: 0.01
Display Filter Method [ ]

.mat/.m File — File: vabcd.m
Select File... | Browse Data...

Initial Value: 0
Gain: 1
(lowest order state on right)

Polynomial Coefficients
Numerator: 1 0 0
Denominator: 1 .2 .8

OK | Cancel | Help

In the Transfer Function Properties dialog box, activate Discrete and set the value of dT to be greater than or equal to the simulation time step.

Frequency domain filter implementation

The dual nature of time and frequency domains means a filter in the time domain can be equivalently implemented in the frequency domain. Depending on the application, however, one domain is usually more convenient to work in than the other.

A recursive IIR filter can be implemented in the frequency domain by taking the product of the frequency domain equivalents of the input sequence and the filter.

$$y(k) = IDFT(Y(\omega)) = IDFT\left(x(\omega) \cdot \frac{H_a(\omega)}{H_b(\omega)} \right)$$

Here, $X(\omega)$ and $Y(\omega)$ are the Discrete Fourier Transforms (DFT) of the input and the output sequences respectively, and *IDFT* represents the Inverse Discrete Fourier Transform operation. $H_a(\omega)$ and $H_b(\omega)$ are the DFTs of the filter coefficients a_i and b_j, respectively, as given by the following difference equation:

$$y(k) = \sum_{i=0}^{M} a_i x(k-i) - \sum_{j=1}^{N} b_j y(k-j)$$

The DFT's $H_a(\omega)$ and $H_b(\omega)$ must be of the same length as $X(\omega)$ and $Y(\omega)$. To accomplish this, the filter coefficients must be zero-padded appropriately. Consequently, the frequency domain implementation is computationally inefficient and will not be discussed further.

Comparison of FIR and IIR filters

The non-recursive FIR filter has a finite memory due to the finite number of delays that can be realized in a practical implementation. FIR filters usually have superior phase characteristics. To obtain sharp cut-off characteristics, FIR filters need to be of high order.

On the other hand, a recursive IIR filter has infinite memory due to its dependence on all prior outputs. Moreover, it generally requires a significantly lower number of elements to obtain a specific cut-off characteristic. The phase characteristics of IIR filter, however, are inferior to those of FIR filters.

Interactive filter design with the transferFunction block

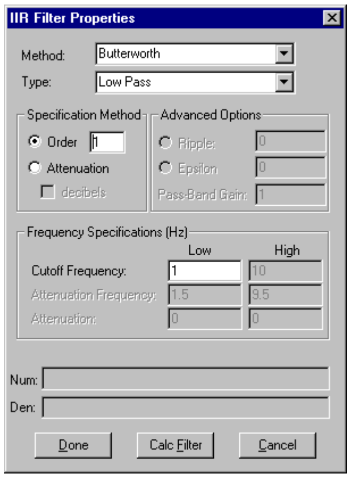
The first step in the digital filter design process is to specify the characteristics that you desire. The more fundamental specification would be the difference equation that is to be satisfied. Such specifications may arise directly from requirements in a signal processing problem.

However, much more common are the specifications that arise when you want to process a continuous time signal digitally, and you expect the digital filter to approximate the performance of an analog filter.

Using the `transferFunction` block you can design either IIR filters using analog prototypes or FIR filters.

IIR filter design

IIR filter design is the design of digital filters using Bessel, Chebyshev, Butterworth, or Inverse Chebyshev analog prototypes. To set up an IIR filter, click on the IIR Filter command button in the Transfer Function Properties dialog box.


IIR Filter Properties

Method: Butterworth

Type: Low Pass

Specification Method

[✓ Order] 1

Attenuation

decibels

Advanced Options

Ripple: 0

Epsilon: 0

Pass-Band Gain: 1

Frequency Specifications (Hz)

	Low	High
Cutoff Frequency:	1	10
Attenuation Frequency:	1.5	9.5
Attenuation:	0	0

Num:

Den:

Using the IIR Filter Properties dialog box

The IIR Filter Properties dialog box lets you constrain the design through either filter order or goodness-of-fit. The analog prototypes can be subsequently converted into a digital filter using bilinear transformation.

Method: You can choose from four analog filter methods, as described below.

- **Bessel:** Bessel filters are designed using Bessel polynomials. The Bessel filters are characterized by the property that the group delay is maximally flat at the origin of the *s*-plane. The step response of the Bessel filters exhibits very low overshoot and both the magnitude and impulse response exhibit gaussian decay as the filter order is increased.

- **Butterworth:** Butterworth filters are characterized by the property that the magnitude characteristic is maximally flat at the origin of the s-plane. This means that all the existing derivatives of the magnitude response are zero at the origin. Butterworth Low Pass filters are all-pole designs and have an attenuation of 3 dB at the critical frequency. The filter order completely specifies the filter and can either be explicitly provided or determined from the attenuation frequency and the attenuation level desired.

- **Chebyshev and Inverse Chebyshev:** Chebyshev filters are characterized by the property that the peak magnitude of the approximation error is minimized over a prescribed band of frequencies. The magnitude is equi-ripple over the band of frequencies. For example, the magnitude oscillates between the maxima and minima of equal amplitude.

  For the Chebyshev filters, the band of the frequencies over which the error is minimized is the Pass Band. For Inverse Chebyshev filters, the error is minimized over the Stop Band. The optimality property of the Chebyshev filters guarantees that no other all-pole filter offers equal or better performance in both the Pass and Stop bands. Inverse Chebyshev filters exhibit monotonic behavior in the Pass Band (maximally flat around the zero frequency) and equi-ripple behavior in the Stop Band. The Low Pass filter has poles in the left half of the s-plane and zeros on the imaginary axis.

Type: Indicates the band pass filter type.

Specification method: The following table describes how the specification method relates to the analog filter prototypes.

Filter	Notes
Bessel	You only need to specify the order.
Butterworth	If you specify the order, VisSim determines the attenuation. If you specify the attenuation, VisSim determines the order.
Chebyshev	If you specify the order, then the order and epsilon define the filter. The attenuation is fixed once a particular order and epsilon are chosen. If you specify attenuation, VisSim determines the order based on the attenuation and epsilon. VisSim determines the order such that the attenuation and epsilon specifications are met.
Inverse Chebyshev	Whether you specify the order or epsilon, the attenuation needs to be specified. If you specify the order, VisSim computes the epsilon based upon the attenuation and the attenuation level desired. In general, as the attenuation desired for a fixed-order filter increases, the corresponding epsilon also increases. This property could be exploited to yield very narrow band filters by specifying an extremely high attenuation, along with a narrow band. If you specify epsilon, VisSim determines the order based upon the attenuation desired. VisSim determines the order such that the attenuation and epsilon specifications are met.

The order of the filter that is generated is twice the order of the filter specified. For example, if you enter 2 in the Order box for a Band Pass filter, the filter generated will have an order of 4.

Cut-off Frequency: The low and high cut-off frequencies in the Frequency Specification box define the band edges. For Low Pass and High Pass filter types, there is only one cut-off frequency. For Band Pass and Band Stop filters, the low and high frequencies are both cut-off frequencies.

Attenuation and Attenuation Frequency: The attenuation characteristics of the filter are defined by:

- **Low and high attenuation frequencies:** The attenuation frequencies are set by the Attenuation Frequency (Low) and Attenuation Frequency (High) parameters. The values you enter indicate the frequency at which the specified attenuation level is reached.

- **Low and high attenuation levels:** The attenuation levels are set by the Attenuation (Low) and Attenuation (High) parameters. The values you enter indicate the amount by which you desire to suppress the level. An attenuation level of 100 equals a magnitude of 1/100.

For example, a Band Pass filter with band edges specified at 100 and 1000, an attenuation level of 10, and attenuation frequencies of 20 (low) and 100 (high) means that the filter gain is 0.1 at 80 and 0.05 at 1100. The epsilon (ε) is a measure of the attenuation level reached by the filter's magnitude characteristics at the critical frequency. Attenuation level at the critical frequency is given by:

Advanced Settings: The Epsilon and Ripple parameters provide two alternate ways of specifying the behavior of a Chebyshev filter.

There is a fluctuation (or ripple) in the amount (or attenuation gain) of the Band Pass and Band Stop. The filter order affects the size of the ripple, and the filter can be tuned to minimize that ripple.

Epsilon refers to the error between the ideal filter and the actual filter, regardless of the ripple. Minimizing the epsilon provides a best fit filter.

$$\frac{1}{\sqrt{1+\varepsilon^2}}$$

The ripple is the attenuation level at the critical frequency. Defining the epsilon completely defines the ripple.

Setting the frequency units

Frequency units can be specified in either radians per second or hertz. You set the frequency unit in the dialog box for the Simulate > Simulation Properties command.

Generating an IIR filter

When you generate an IIR filter, VisSim calculates the polynomial coefficients for the transfer function with the desired frequency characteristics.

▶ **To generate an IIR filter**

1. Click on the Calc Filter command button to calculate the filter coefficients. The coefficients will be displayed in the Num (numerator) and Den (denominator) boxes.
2. Click on the Done button to close the IIR Filter Setup dialog box and transfer the filter numerator and denominator coefficients to the Transfer Function Setup dialog box.

FIR filter design

VisSim uses the Remez Multiple Exchange algorithm to design FIR filters. FIR filters in discrete time are realized as all-zero filters and are characterized by a finite impulse response in the time domain. Because they are all-zero filters, they are particularly well-suited to efficient computation by tapped delay.

FIR filter design is typically executed in the frequency domain for convenience. The filter has the desired magnitude specifications and a linear phase characteristic.

Differentiators and Hilbert transformers

You can also design differentiators and Hilbert transformers using the Remez Multiple Exchange algorithm.

Differentiators are characterized by an approximate linear magnitude response over the desired frequency range.

Hilbert transformers are characterized by a flat magnitude response and a phase of 90^o over the specified frequency range. Frequency characteristics of an ideal Hilbert transformer are:

$$H(e^{j\omega}) = -j \qquad 0 \le \omega \le \frac{\pi}{T}$$

$$= +j \qquad \frac{\pi}{T} \le \omega \le \frac{2\pi}{T}$$

where ω is the angular frequency and *T* is the sampling time period.

Discrete and continuous FIR filter design

The discrete time filter design problem is treated as a weighted Chebyshev approximation problem and is solved using the Remez Multiple Exchange algorithm to compute the filter coefficients. The algorithm builds a discrete time representation of the filter.

In VisSim, the Discrete parameter in the Transfer Function Setup dialog box controls whether the generated FIR filters are discrete or continuous. When you design a discrete FIR filter, you must also specify a time step in the dT box.

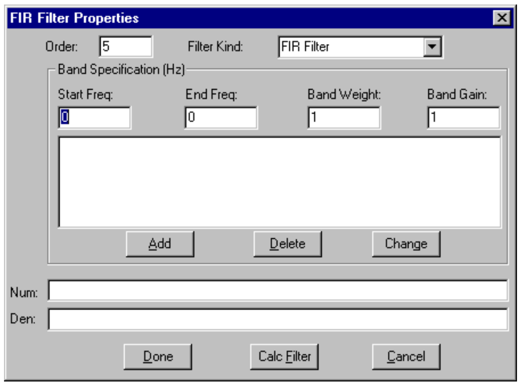
To implement a continuous FIR filter, de-activate the Discrete parameter. In this case, the filter is initially designed as a discrete time filter. Bilinear transformation is subsequently used to produce a continuous time equivalent. For more information on the Remez algorithm, see *Theory and Application of Digital Signal Processing* (Prentice Hall).

Tapped delay implementation

Tapped delay is a method of transfer function implementation that has linear computational and storage requirements with respect to model order. Because most FIR filters have a tendency to be high order, it makes sense to design FIR filters with tapped delay implementation. To do so, activate Tapped Delay in the Transfer Function Properties dialog box.

Using the FIR Filter Properties dialog box

When you click on the FIR Filter command button in the Transfer Function Setup dialog box, the FIR Filter Setup dialog box is opened.


FIR Filter Properties

Order: 5

Filter Kind: FIR Filter

Band Specification (Hz)

Start Freq:	End Freq:	Band Weight:	Band Gain:
0	0	1	1

Order: The value specified in the Order box defines the filter order. Typically, higher orders yield better approximations.

Filter Kind: Indicates the type of filter to be generated. Your choices are FIR, differentiator, and Hilbert transformation. Descriptions of these filters are on page 104.

Band specification: Band specifications describe the frequency bands magnitude response characteristics of the filter. The following rules must be observed when entering band specifications:

- Frequencies are specified in hertz for discrete and continuous filters.
- For discrete filters, the frequency specified must be lower than the Nyquist frequency.
- For continuous filters, infinite frequency is indicated using the reserve word “inf.”

Start Freq and End Freq: Defines the lower and upper cut-off frequencies for each band.

Band Weight: Dictates the relative amounts of error allowed for each band. Higher weight values of a particular frequency band reflect higher sensitivity to error, where error is perceived as the difference between the actual and desired filter response. At least one band must have a weight of 1. For each of the other bands, you can use a higher or lower weight depending on the relative error that can be tolerated.

An equal weight of 1 on all bands indicates that the maximum absolute error on all bands is the same. A weight of 10 on one band and a weight of 1 on other bands implies that the former band has a maximum approximation error that is ten times less than that of the other bands.

Band Gain: Defines the desired frequency response magnitude for each band.

▶ **To add a band**

- Enter the band specification and click on the Add button.

The band information is added to the list box. Each row in the list box corresponds to a single band. For FIR filters, if the number of bands increases, the filter order must be increased correspondingly, to maintain the same approximation error. For differentiator and Hilbert transformers, the number of bands is limited to one. The gain on the differentiator implies the gain achieved at the end frequency. The weight in either case is optimally adjusted to give the best error characteristics.

▶ **To delete a band**

- Select the band to be deleted from the list box and click on the Delete button.

▶ **To change a band's specifications**

1. Select the band to be changed from the list box.

   The band's data appears in the edit boxes.

2. Make the desired changes.
3. Click on the Change button. The band data is modified in the list box to reflect the changes.

Generating an FIR filter

Calc Filter command button. This generates the appropriate filter coefficients. Before computing the filter coefficients, the algorithm computes the maximum approximation error. This error is usually referred to as delta (δ) and is defined as the weighted difference between the actual and the desired magnitude response. A band with a weight of one will have delta as its absolute approximation error, while a band with a weight of 10 will have its absolute error 0.1 times δ. The value of δ is displayed in the message box.

▶ **To generate an FIR filter**

1. Click on the Calc Filter button. The coefficients are displayed in the Num (numerator) and Den (denominator) boxes. If the delta displayed is too large, increase the order of the filter and re-calculate the filter.
2. Click on the Done button to close the FIR Filter Properties dialog box and transfer the filter numerator and denominator coefficients to the Transfer Function Properties dialog box.

Chapter 10

Working with Other Applications

This chapter covers the following information:

- Using the `import` block to import data from other applications into VisSim
- Using the `export` block to export data out of VisSim
- Using the `DDE`, `DDEreceive`, and `DDEsend` blocks to create links to other applications

Importing basics

VisSim uses the `import` block to import information from many different file types generated by other applications. These include data files (.DAT), MatLab and MatLab-like files (.MAT and .M), and 8-bit or 16-bit sound files (.WAV).

The `import` block reads data points from the specified input file into the system model and translates them into scalar, vector, or matrix output signals. The `import` block can receive up to 50 scalar inputs and an unlimited number of vector or matrix input. The data can be either fixed interval or asynchronous.

The `import` block is particularly useful for comparing experimental data with simulated results and for inserting trial control data from an external source.

Setting up the input file

The input file can contain a header line to describe the separation of data points. The following table describes the header line format:

For this type of interval	Use this format
Fixed interval	#I=start-time, end-time, increment
Asynchronous interval	#T=number (time-column)

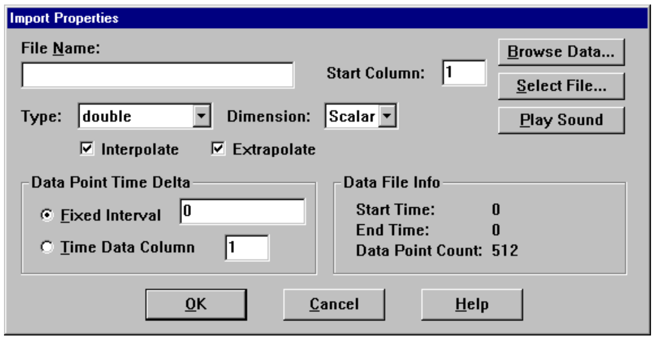
Importing data

Importing data involves dragging an `import` block into the work area and setting up the block to reference the input file.

▶ **To import data**

1. From the Signal Producer category in the Blocks menu, drag an `import` block into the work area.
2. Choose Edit > Block Properties.
3. Click the mouse over the `import` block.

   The Import Properties dialog box appears.


Import Properties

File Name:

Start Column: 1

Type: double Dimension: Scalar

Interpolate [✓] Extrapolate [✓]

Data Point Time Delta

Fixed Interval [●] 0

Time Data Column [○] 1

Data File Info

Start Time: 0

End Time: 0

Data Point Count: 512

4. Select the import parameters. (See the descriptions below for more information about each parameter.)
5. Click on the OK button, or press ENTER.

Using the Import Properties dialog box

The Import Properties dialog box provides the following options:

File Name: Indicates the file to be used as input to VisSim. You can specify .DAT, .M, .MAT, or .WAV files. When you specify a .WAV file, you can play the sounds you imported by clicking on the Play Sound button.

If you do not know the name or location of the input file, click on the Select File button to locate and choose a file.

To browse or edit the input file, click on the Browse Data button after you select a file.

Start Column: In a multi-column file, you can choose the column that corresponds to the top-most connector tab. The default value 1 corresponds to the first column.

Type: Indicates the type of data to be imported.

Dimension: Controls the dimensionality of the output signals. The choices are scalar, vector, and matrix.

Interpolate: Interpolates between two data points, instead of using the last known data value. Thus, if the data point is 5 at t_1, and 15 at t_2, then at $t_{1.5}$, the data point is 10 with interpolation, and 5 with no interpolation.

Extrapolate: Infers the next unknown data point based on the difference between the last two known data points.

Data Point Time Delta: Indicates the time interval between data points in the input file. If the input file was generated by VisSim using the `export` block, VisSim automatically reads the time interval information from the file header and sets the parameter accordingly. You have the following choices:

- **Fixed Interval:** Indicates that data points occur in fixed intervals. Enter the interval in the corresponding box. This is the default setting.
- **Time Data Column**: Indicates that data points occur in irregular time intervals. Enter the column containing the time data points in the corresponding box. Valid column numbers are 1 through 16.

Data File Info: Provides read-only information about the imported data. The Start Time and End Time fields indicate when VisSim starts and stops recording data. The Data Point Count field indicates the maximum number of data points to be read into VisSim. If the input file was generated in VisSim using the `export` block, VisSim automatically reads the data point count from the file header and sets the field accordingly. The maximum number of data points that can be read into VisSim is 128,000 (Windows 3.1) or 250 million (Windows 95 and Windows NT).

Exporting basics

The `export` block writes signals to an output file in .DAT, .M, .MAT, or .WAV file format. The `export` block can send up to 50 scalar outputs and an unlimited number of vector or matrix output. The output file can subsequently be used as input to VisSim or to a variety of other programs, such as MatLab and Microsoft Excel. The following information is written to the file:

- Data points that represent signal values. Data points are stored as ASCII text.
- Time interval information that applies to the data points.

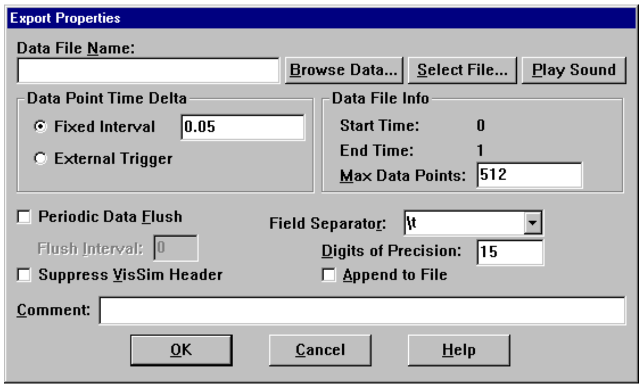
Exporting data

Exporting data involves dragging an `export` block into the work area and setting up the block to reference the output file.

▶ **To export data**

1. From the Signal Consumers category in the Blocks menu, drag an `export` block into the work area.
2. Choose Edit > Block Properties.
3. Click the mouse over the `export` block.

   The Export Properties dialog box appears.



Export Properties

Data File Name:

Data Point Time Delta: [✓ Fixed Interval] 0.05 / External Trigger

Data File Info: Start Time: 0; End Time: 1; Max Data Points: 512

☐ Periodic Data Flush; Flush Interval: 0

☐ Suppress VisSim Header

Field Separator: \t

Digits of Precision: 15

☐ Append to File

Comment:

4. Select the export parameters. (See the descriptions below for more information about each parameter.)
5. Click on the OK button, or press ENTER.

Using the Export Properties dialog box

The Export Properties dialog box provides the following options:

Data File Name: Indicates the name of the export file into which data points are to be written. You can type the file name directly into this box or select one using the Select File button. If you do not specify a data file, VisSim writes the data points to a file using the same name as your current block diagram. VisSim applies the .DAT extension to the file and stores it in your current directory.

You can export data in .DAT, .M, .MAP, .MAT, and .WAV file formats. The following special considerations apply to map files and wave files:

- If you want to create an output file to be used as input to the `map` block, you must specify the .MAP extension.
- You can create 8-bit and 16-bit sound files. You specify the bit count in the Digits of Precision box. Provided you have the appropriate hardware

configuration and software drivers installed, you can preview the sound by clicking on the Play Sound button.

If you click on the Browse Data button, the file specified in the Data File Name box is opened for you to examine or edit.

Data Point Time Delta: Controls how VisSim writes the time interval information to the data file. You have the following choices:

- **Fixed Interval:** Indicates that data points occur in fixed intervals. The default interval used will be taken from the simulation step size. You can specify a different interval, however, it should be a multiple of the simulation step size, because the `export` block does not interpolate. Data is only exported at integral multiples of the simulation step size. This automatic adjustment is invisible when it occurs, which means it is not reflected in either the `export` block's dialog box or the data file header. You can see the adjustment only when you open the data file.

  If you import the output file into a simulation, you should edit the file header to reflect the interval at which the data was actually exported. The `import` block will interpolate as needed, retaining the timing of the original simulation run. Use the Browse Data button to open the data file for editing. The format of the data header file should be as follows:

  #I = *start-time, end-time, increment*

  The default is Fixed Interval.

- **External Trigger:** Indicates that data will be recorded based on the state of the external trigger input. When External Trigger is activated, VisSim adds a round input connector tab to the `export` block. A zero value on the trigger inhibits data recording. A value of 1 causes a data point to be recorded.

  Valid column numbers are 1 through 16, inclusive.

Periodic Data Flush and Flush Interval: When activated, Periodic Data Flush writes the data in the export buffers to the specified data file at intervals established with Flush Interval.

Suppress VisSim Header: Suppresses writing the data header to the export file. Suppressing the data header may be necessary if the export file is to be imported into a software product other than VisSim.

The header information indicates whether the data is fixed or variable interval; the valid time range over which the data is collected; the actual fixed interval; and the time column for variable interval data. The following formats are used:

Fixed Interval	#I = *start-time, end-time, increment*
Variable Interval	#T= *number (time-column)*

Field Separator: Specifies the column separation character in the export file, which allows for compatibility with other applications. Recognized column separators are tabs, new lines, spaces, commas, semi-colons, and colons.

Digits of Precision: Specifies (for .DAT, .M, .MAT files) the maximum number of significant digits printed regardless of the decimal point. The default is 15.

For .WAV files, use Digits Of Precision to indicate whether the sound file is 8-bit or 16-bit. Enter 8 for 8-bit sound files or 16 for 16-bit sound files.

Append to File: Appends the exported data to the end of a specified file, instead of re-writing the file at the start of each new simulation run. This parameter is useful for multi-run applications, such as data acquisition, Monte Carlo simulations, and neural network training

Comment: Specifies a comment that is placed at the beginning of the exported data file. A comment is limited to 180 characters.

Data File Info: Provides read-only information about the export file. The Start Time and End Time fields indicate when VisSim starts and stops writing data points to the export file. These settings are obtained from the current settings of Range Start and Range End in the Simulation Properties dialog box.

The Max Data Points field indicates the maximum number of data points to be written to the export file. The default is 512 data points. The maximum number of data points that can be written to file is 128,000 (Windows 3.1) or 250 million (Windows 95 and Windows NT).

DDE basics

By creating dynamic data exchange (DDE) links, you can share information in one file with several other files, and you need only maintain the original file; the other files are updated automatically. For example, if you store data in a Microsoft Excel spreadsheet, you can use that data in a VisSim block diagram. When you update the spreadsheet, VisSim automatically updates the data in the block diagram when you run a simulation.

You create DDE links by copying a selection from one application (referred to as the source or server) and pasting it into another one (referred to as the destination or client) using the Paste Link or Paste Special command. Before you can create a link, the source file must be saved to disk.

VisSim offers three blocks for creating DDE links:

- **The `DDEsend` block,** which links source information in a VisSim block diagram to another application, such as a Microsoft Excel or Visual Basic file.
- **The `DDEreceive` block,** which links source information in an application file to a VisSim block diagram.

- **The** `DDE` **block,** which establishes a two-way link: it acts as both the source (sender) and destination (receiver). For example, a `DDE` block can send data to a Visual Basic program to work on, and then receive the updated data back from Visual Basic.

You can create DDE links only between VisSim and other Windows applications that support DDE linking. Some applications do support DDE links, but do not support creating the links by copying and pasting selections. When this is the case, you can still create a link by entering the link address directly to both the source and destination files.

Creating an app-to-VisSim link with DDEreceive

Follow this procedure when the source information for the link is contained in an application other than VisSim.

▶ **To create a DDE link from an application into VisSim**

1. In the application, select the information you want linked to your block diagram, and from the Edit menu, choose the Copy command.

   The selected information is copied to the Clipboard.

2. Switch to VisSim and open the block diagram in which you want to create a DDE link.

3. Do one of the following:

   a) Choose Edit > Paste Link.

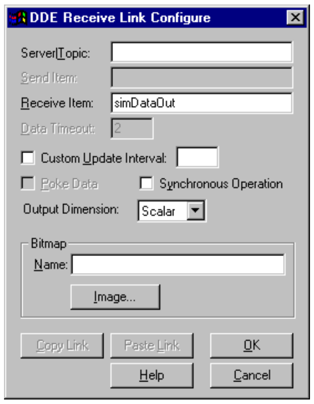
   b) Position the pointer where you want the `DDEreceive` block to appear and click the mouse.

   -Or-

   a) From the Blocks menu under DDE, drag a `DDEreceive` block into the work area.

   b) Choose Edit > Block Properties and click the mouse over the `DDEreceive` block.

4. The DDE Receive Link Configure dialog box appears.


DDE Receive Link Configure

Server|Topic:

Send Item:

Receive Item: simDataOut

Data Timeout: 2

☐ Custom Update Interval:

☐ Poke Data ☐ Synchronous Operation

Output Dimension: Scalar

Bitmap

Name:

5. Click on the Paste Link button and choose the options you want. (For information on the options, see the descriptions below.)
6. Click on the OK button, or press ENTER.
7. Choose Simulate > Go to update the link.

Using the DDE Receive Link Configure dialog box

The DDE Receive Link Configure dialog box provides the following options:

Server|Topic: Indicates the name of the source application (server) and the type of information (topic). For example, Excel|FOO.XLS indicates an Excel spreadsheet named FOO.

If the source application supports Copy Link, VisSim automatically fills in this parameter when you click on the Paste Link button.

If the source application does not support Copy Link, you must enter the source application name and topic name directly to this box. Use the same names that the source file uses as its server and topic names. Separate the names with a pipe (|) character.

Send Item: This option does not apply to the `DDEreceive` block.

Receive Item: Indicates a name that references cells, cell ranges, values, or a field of data in the source file. For example, R1C1 references the information in the cell occupying row 1, column 1 of an Excel spreadsheet.

If the source application supports Copy Link, VisSim automatically fills in this parameter when you click on the Paste Link button.

If the source application does not support Copy Link, you must enter the same name that the source file uses as its item name.

Data Timeout: This option does not apply to the `DDEreceive` block.

Custom Update Interval: Indicates how often the `DDEreceive` block requests information from the linked application. If you enter the value 1, `DDEreceive` requests information once per sec; if you enter 10, `DDEreceive` requests information once every 10 sec; and so on. If you do not enter a value, `DDEreceive` updates at each time step of the simulation by default.

Poke Data: This option does not apply to the `DDEreceive` block.

Synchronous Operation: Suspends the simulation until the `DDEreceive` block receives a message with updated data.

The `DDEreceive` block has a buffer that contains the current value of the block. If the block is not synchronous, at every time step, `DDEreceive` supplies whatever value is in its buffer. When Synchronous Operation is turned on and the `DDEreceive` block has not received updated data since the last time step, `DDEreceive` waits until it receives a new message with updated data.

Output Dimension: Controls the dimensionality of the data exiting the `DDEreceive` block. The choices are scalar, vector (*n* x *m*), or matrix (*m* x *n*).

Bitmap: Applies a bitmap image to the `DDEreceive` block. You can type the file name directly into the Name box or select one by pressing on the Select Bitmap button.

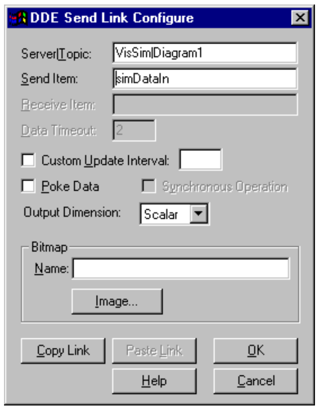
Creating a VisSim-to-app link with DDEsend

Follow this procedure when the source information for the link is contained in a block diagram.

▶ **To create a DDE link from a VisSim block to another application**

1. In VisSim, wire a `DDEsend` block to the output of the block that contains the information you want linked to another application.
2. Choose Edit > Block Properties and click the mouse over the `DDEsend` block.

   The DDE Send Link Configure dialog box appears.



Server|Topic: VisSim|Diagram1
Send Item: simDataIn
Receive Item:
Data Timeout: 2
Output Dimension: Scalar
Name:

3. In the Send Item box, enter a name. The default name is simDataIn.

   Note: When the block diagram contains multiple links to other applications (that is, the diagram contains more than one `DDEsend` block), the name you enter in the Send Item box must be unique to that block diagram. If it's not unique, VisSim will not pass the correct information to the application.

4. Choose the options you want. (For information on the options, see the descriptions below.)
5. Click on the Copy Link button.
6. Click on the OK button, or press ENTER.
7. Switch to the destination application and open the file in which you want to create a link.
8. Position the insertion point where you want to insert the information.
9. Choose Edit > Paste Link.

   Note: Some applications have a Paste Special command instead of a Paste Link command. Refer to the application's documentation for information on linking.

10. Switch back to the block diagram, and choose Simulate > Go to update the link.

Using the DDE Send Link Configure dialog box

The DDE Send Link Configure dialog box provides the following options:

Server|Topic: Indicates the name of the source application (server) and the type of source information (topic). This parameter defaults to VisSim|*name-of-block-diagram*. The server name must always be VisSim.

Send Item: Indicates a name for the source information. The destination file uses this name in its item field. To maintain multiple DDE links from a single block diagram, the name you enter must be unique.

The information in this box defaults to simDataIn.

Receive Item: This option does not apply to the `DDEsend` block.

Data Timeout: This option does not apply to the `DDEsend` block.

Custom Update Interval: Overrides the time step interval for sending data to the destination application. If you enter the value 1, `DDEsend` sends data once per sec; if you enter 10, `DDEsend` sends data once every 10 sec; and so on. If you do not enter a value, `DDEsend` sends data at each time step of the simulation. You can use Custom Update Interval only when Poke Data is activated.

Poke Data: Sends data to the destination application at every time step, regardless of whether it is ready to receive the data. When Poke Data is not activated, data is sent only when the destination application requests it.

You can override the time step interval for sending data with the Custom Update Interval.

Output Dimensions: Controls the dimensionality of the data entering the `DDEsend` block. The choices are scalar, vector (*n* x *m*), or matrix (*m* x *n*).

Bitmap: Applies a bitmap image to the `DDEsend` block. You can type the file name directly into the Name box or select one by pressing on the Select Bitmap button.

Creating a two-way link with DDE

The `DDE` block is a combination of the `DDEreceive` block and `DDEsend` block: the `DDE` block can both send and receive information. As a server, the `DDE` block passes source information to another application to work on. As a client, the `DDE` block receives updated information back from the application.

▶ **To create a two-way DDE link**

Use this procedure when you want VisSim to fill in the name of the server and topic pair of the destination application.

1. Create the link to pass information from the application to VisSim:
    a) In the application, select the information you want linked to your VisSim block diagram, and choose Edit > Copy.

       The information is copied to the Clipboard.
    b) Switch to VisSim and open the block diagram in which you want to link the copied information.
    c) From the Blocks menu under DDE, drag a `DDE` block into the work area.

   d) Choose Edit > Block Properties and click the mouse over the DDE block.

      The DDE Link Configure dialog box appears.

   e) Choose the Paste Link button and the additional options you want. (For information on the options, see the descriptions below.)

   f) Click on the OK button, or press ENTER.

2. Create the link to pass information from VisSim back to the application:

   a) In VisSim, wire the block (containing the information you want linked to the other application) to the DDE block.

   b) Choose Edit > Block Properties and click the mouse over the DDE block.

      The DDE Link Configure dialog box appears.

   c) In the Send Item box, enter a unique name.

   d) Choose the Copy Link button and the additional options you want. (For information on the options, see the descriptions below.)

   e) Click on the OK button, or press ENTER.

   f) Switch back to the application file.

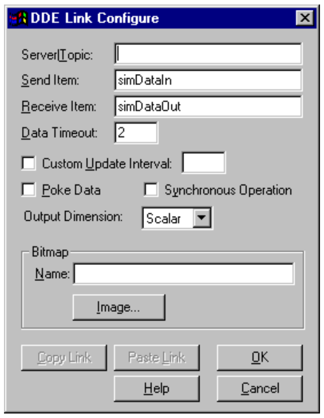
   g) Move the insertion point to where you want to insert the information.

   h) Choose Edit > Paste Link.

      Note: Some applications have a Paste Special command instead of a Paste Link command. Refer to the application’s documentation for information on linking.

Using the DDE Link Configure dialog box

The DDE Link Configure dialog box provides options for establishing a DDE link, specifying the source information, indicating the time-out interval, and more. These options are described below.



Server|Topic: Indicates the name of the application (server) and the type of information (topic) to which you're establishing a link. For example, VBDDE|NNET sends data to and receives data from the application called VBDDE on the NNET topic. Use the pipe (|) character to separate the server from the topic.

Send Item: Indicates a name for the source information. The destination file uses this name for its item field. To maintain multiple DDE links from a single block diagram, the name you enter must be unique.

The information in this box defaults to simDataIn.

Receive Item: Indicates a name that references cells, cell ranges, values, or a field of data in the source file. For example, R1C1 references the information in the cell occupying row 1, column 1 of an Excel spreadsheet.

If the source application supports Copy Link, VisSim automatically fills in this parameter when you click on the Paste Link button.

If the source application does not support Copy Link, you must enter the same name that the source file uses as its item name.

Custom Update Interval: Overrides the time step interval for sending and receiving information. If you enter 1, `DDE` requests and sends information once per sec; if you enter 10, `DDE` requests and sends information once every 10 sec; and so on. If you do not enter a value, `DDE` updates at each time step of the simulation.

Data Timeout: Indicates the time, in sec, that VisSim will wait to receive simulation time step data from the client. The default is two sec.

Poke Data: This option does not apply to the `DDE` block.

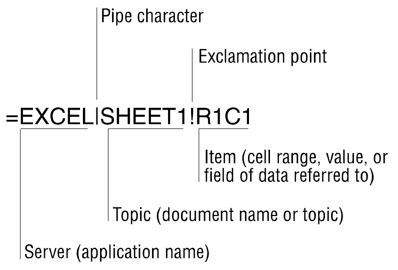
Synchronous Operation: This option does not apply to the `DDE` block.

Output Dimension: Controls the dimensionality of the data entering and exiting the block. The choices are scalar, vector (*n* x *m*), or matrix (*m* x *n*).

Bitmap: Applies a bitmap image to the `DDE` block. You can type the file name directly into the Name box or select one by pressing on the Select Bitmap button.

Creating DDE links with applications that do not support Copy Link and Paste Link

A DDE link consists of a three-part link address contained in both the source (server) and destination (client) files. An example of such an address is shown below:


Pipe character
Exclamation point
=EXCEL|SHEET1!R1C1
Item (cell range, value, or field of data referred to)
Topic (document name or topic)
Server (application name)

Most applications, including VisSim, automatically create the link address using the Copy Link and Paste Link commands. If, however, the application with which you're linking supports DDE but not the Copy Link and Paste Link commands, you can still create a DDE link by typing the link address directly into the source and destination files. Just make sure that the server, topic, and item names are the same in source and destination files.

Refer to the descriptions of the `DDE`, `DDEreceive`, and `DDEsend` blocks earlier in this chapter for information on how to enter these fields directly into the blocks. Refer to the documentation for the other application for entering link addresses.

Chapter 11

Working with Large Diagrams

This chapter covers the following information:

- Creating model hierarchy
- Embedding blocks
- Adding block diagrams
- Using variables to pass signals
- Using path aliases to reference files
- Tracking diagram progress
- Protecting your work

Creating model hierarchy

Compound blocks allow you to encapsulate one or more blocks in a single block. This gives you more flexibility in constructing and editing your block diagram models, especially if they are complex. The top level blocks display major component connectivity, leaving the underlying levels to describe the logic of each component.

Compound blocks also encourage a modular approach to large model construction by allowing you to design and test functionally independent subcomponents concurrently. Then using the `embed` block or the File > Add command (as described on pages 127 and 128), you can incorporate each subcomponent back into the large system diagram.

You can have as many levels as you want in a compound block. (The number is limited only by your system resources.) If your compound block contains sensitive information, you can prevent other users from viewing the compound block by locking it closed. You can alternatively apply read-only attributes to the compound block. This lets other users view the contents of the compound block but denies them the ability to edit it. Applying protection to compound blocks is described on page 136.

You can also make compound blocks disappear from view in display mode.

To make compound blocks easier to distinguish, you can color them blue, or for more visual power, attach bitmap images to them. If you choose to identify a compound block by name, you can change the name with the Edit > Block Properties command, as described on page 17.

Creating a compound block

When you create a compound block, VisSim attaches connector tabs to the compound block for each of the following situations:

- All unsatisfied connector tabs on the internal blocks (except global variables)
- All satisfied connector tabs to external blocks

▶ **To create a compound block**

1. Select the blocks to be encapsulated.
2. Choose Edit > Create Compound Block**.**
3. Under compound name, enter a name. Avoid using the dot (.) character in the name; VisSim uses it to separate compound block names in the title bar. The default name is *Compound.*
4. Click on the OK button, or press ENTER.

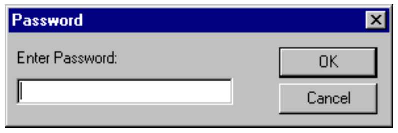
Drilling into a compound block

The process of moving through and displaying the levels of a compound block is referred to as *drilling.* As you drill into a compound block, VisSim adds the name of the compound block to the title bar to help you keep track of where you are.

▶ **To drill down**

1. Point to a compound block.
2. Click the right mouse button.

3. If the compound block is password-protected, enter the password in the Password dialog box, then click on the OK button or press ENTER.



The compound block remains unlocked until you close the diagram.

▶ **To pop up**

1. Point to empty screen space.
2. Click the right mouse button.

Hiding compound blocks

You can selectively hide compound blocks while working in display mode.

▶ **To hide compound blocks**

1. Choose Edit > Block Properties.
2. Point to the compound block you want hidden in display mode and click the mouse.
3. Activate the Hide In Display Mode parameter.
4. Click on the OK button, or press ENTER.
5. Activate View > Display Mode.

Configuring pictures on compound blocks

The pictures that can be configured on compound blocks are graphical images in .BMP file format. You can create them yourself or choose from the VisSim bitmap library, which resides in VISSIM\BITMAP\DIAGRAM. See Appendix E, “Working with Bitmaps,” for pictures of these bitmaps.

▶ **To configure a picture on a compound block**

1. Choose Edit > Block Properties.
2. Point to the compound block on which the picture is to be configured and click the mouse.
3. Click on the Select Image button and choose the bitmap image to be configured on the block.
4. Click on the OK button, or press ENTER.

5. Click on the OK button, or press ENTER when the Compound Properties dialog box appears.

Labeling connector tabs on compound blocks

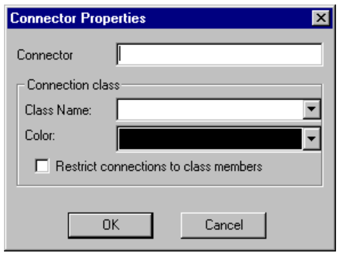
When you want to distinguish between the inputs and outputs on compound blocks, you can assign labels to their connector tabs. When the View > Connector Labels command is activated, connector labels are displayed on the block. In addition, when you drill into it, those labels appear next to the input and output connectors on the left and right side of the screen.

If you do not specify a connector label, the label defaults to the class name specified in the Connector Properties dialog box, if one is specified.

▶ **To assign connector labels**

1. Point to the connector tab on the compound block you want to label. The pointer turns into an upward pointing arrow.
2. Double-click the mouse.

   The Connector Properties dialog box appears.



3. In the Connector box, enter a name.
4. Click on the OK button, or press ENTER.
5. If you want the label to appear on the block, activate the Connector Labels command in the View menu.

Dissolving a compound block

Use the Edit > Dissolve Compound Block command to de-encapsulate the blocks one level below the current level. When you execute Dissolve Compound Block, the blocks immediately below the current level move up to the current level. The blocks remain highlighted until the next command is executed to make it easier to recreate the compound block, in case you change your mind.

When you dissolve a compound block, VisSim maintains all internal wiring connections.

▶ **To dissolve a compound block**

1. Choose Edit > Dissolve Compound Block.
2. Point to the compound block and click the right mouse button.
3. Click the mouse on empty screen space to exit this command.

Other things you can do with compound blocks

VisSim lets you do other things with compound blocks.

For information about	See
Coloring compound blocks	Page 267
Protect compound blocks	Page 136

Embedding blocks

With embedding, you can include information created in one VisSim block diagram, referred to as the *source diagram*, in one or more other block diagrams, referred to as the *destination diagrams*. Each time the source diagram changes, the changes are propagated in the destination diagrams.

When you embed a block diagram, a read-only version of the diagram is inserted into the destination diagram along with a link to the source diagram. You can drill into the embedded diagram just as you would a compound block; however, you cannot edit it. Edits can only be made to the source diagram.

Setting up a diagram to be embedded

Before you can embed a diagram, check that its top level is a single compound block. If it's not, use the Edit > Create Compound Block compound to create one, as described on page 124.

If you want to restrict access to the `embed` block, apply the protection to the compound block, as described on page 138.

Embedding a block diagram

Embedding a block diagram involves dragging an `embed` block into the work area and setting up the link to the source file.

▶ **To embed a block diagram**

1. Open the destination diagram and move to the block diagram level where you want to insert an embedded block diagram.
2. Drag an `embed` block into the work area.

3. Choose Edit > Block Properties.
4. Click the mouse over the `embed` block.
5. In the File Name box, enter the name of the block diagram file to be embedded. If you do not see the file you want, click on the Select File button to search for it.
6. Click on the OK button, or press ENTER.

Editing an embedded block diagram

You cannot edit an embedded block diagram itself. Instead, you open and edit the source file to which the embedded diagram is linked. When you edit a source file, all embedded diagrams linked to that source file are immediately updated to reflect the changes.

Reconnecting an embedded block diagram

You may lose a link if you move or rename the source file. If this occurs, you must redirect the link to the appropriate location or file name.

▶ **To reconnect a link**

1. Choose Edit > Block Properties.
2. Point to the `embed` block and click the mouse.
3. In the File Name box, enter the correct path or new file name. If you are unsure of the path or file name, click on the Select File button to search for the file you want.
4. Click on the OK button, or press ENTER.

Adding block diagrams

You can add another block diagram to the currently opened diagram using the File > Add command. After you've added the block diagram, some blocks and wires may overlap as a result of this operation; use the mouse and Edit menu commands to reposition them appropriately.

▶ **To add a block diagram to the current diagram**

1. Open the block diagram into which you want to add another block diagram.
2. Choose File > Add.
3. In the File Name box, type or select the name of the block diagram you want to add. If you do not see the block diagram you want to add, select a new drive or directory.

4. Click on the OK button, or press ENTER.

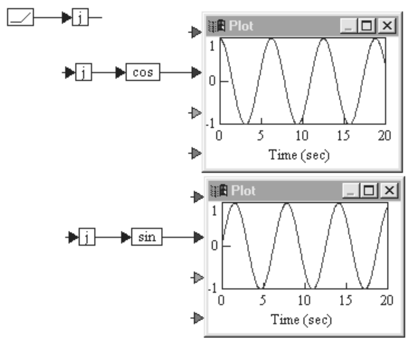
   An empty rectangular box appears that represents the block diagram. The pointer is anchored to the box.

5. Move the box to where you want the block diagram added.
6. Click the mouse.

Using variables to pass signals

The `variable` block lets you name a signal and transmit it throughout your diagram without the use of wires.

Variables of the same name share signals. For example, in the diagram below, the variable *j* is used in three different locations:



The variable *j* in the upper part of the diagram is the declared variable. Only declared variables are allowed input signals. In addition, there can be only one declared variable of a given name. The other two *j* variables are referenced variables. Wires cannot be fed directly into referenced variables; they receive their input from the declared variable.

All variables can have any number of output signals.

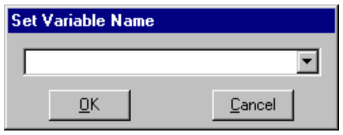
Creating variables

The `variable` block is located under the Blocks menu in the Annotation category.

▶ **To create a variable block**

1. Choose Edit > Block Properties.
2. Click the mouse over the `variable` block.

The Set Variable Name dialog box appears.



3. Do one of the following:

To	Do this
Create a new variable	Enter a new name. To limit the scope of the variable, preface the name with a colon (:) character to make it local; two colon (::) characters to make it definition-scoped, or no colons to make it global. (For information on scoping variables, see the descriptions below.)
Reference an existing variable	Click on the DOWN ARROW and choose a name from the list. (For information on the scope of the variables, see the descriptions below.)

4. Click on the OK button, or press ENTER.

Naming a variable

It is never a good idea to name a `variable` block *-X*, or a number, like 1 or 2 or 3. Naming a variable *-X* leads to confusion with the `-X` block. Naming a variable a number leads to confusion with the `const` block.

Scoping variables

In VisSim, you can define which portions of the diagram can reference a variable by designating its scope. There are three types of scope: *diagram scope, definition and below scope*, and *level scope*.

- Diagram scope indicates that the variable can be referenced at any hierarchical level of the block diagram. Variables with diagram scope are referred to as *global variables*.
- Definition and below scope indicates that the variable can be referenced at the current hierarchical level, as well as all levels beneath it. To identify definition-scope variables, preface their names with two colon (::) characters.
- Level scope indicates that the variable can be referenced only at the current level of the block diagram. Variables with hierarchical level scope are referred to as *local variables*.

Level scope

A variable with level scope cannot be referenced outside of its current hierarchical level. Although level scoping is the most restrictive type of scope, it actually has several key advantages. By limiting the region over which variables can be referenced, you can construct sections of a diagram without worrying about whether your variable names conflict with other names used in other parts of the diagram. In addition, users reading your diagram will know immediately that the variables' use is limited to a small region.

Variables with level scoping are prefaced with the colon (:) character.

Definition and below scope

Giving a variable definition and below scope allows the variable to be referenced not only at the hierarchical level of the diagram at which it was defined, but also at all the levels beneath it. For example, if compound block A contains a definition scope variable, all the sublevels in compound block A are able to use the variable.

By using definition-scoped `variables`, you can copy or add subsystems to an existing diagram without breaking or misdirecting references.

To give a variable definition and below scope, preface its name with two colon (::) characters.

Diagram scope

Variables with diagram scope are called global variables. Because global variables reference any part of a block diagram, you should exercise caution in your use of them. Global variables can make a block diagram hard to maintain because they increase the diagram's complexity.

In addition, global variables increase the chances of a conflict in names between modules. For example, engineers working on different parts of a large project may choose the same name for different global variables. The problem won't surface until each module is added to the master diagram.

As a rule of thumb, global variables should be used only when transmitting system-wide constants or signals that would be laborious or visually messy to represent as wires.

Making copies of global variables: When you make a copy of a compound block containing a global variable with wired input, VisSim renames the copied occurrence of the global variable in the following manner:

original-variable-block-name@unique-number

Finding variable definitions

To find where a variable is defined, use the Edit > Find command and activate the Match Variable Definitions Only option.

Built-in variables

The following variable block names are built into VisSim:

Block name	**Description**
$firstPass	Generates an initial unit pulse on the first step of a simulation.
$lastPass	Generates a final pulse on the last step of a simulation.
$runCount	Holds the simulation iteration count for multiple simulation runs, such as Monte Carlo simulations and parameter sweeps.
$timeStart	Returns the start time of the simulation.
$timeStep	Returns the step size of the simulation.
$timeStop	Returns the stop time of the simulation.

Using path aliases to reference files

As it name implies, a path alias is another name for all or part of the full specification of a file. You use path aliases to quickly insert frequently referenced files — for example, map files, import files, and bitmap image files. Rather than entering the complete file specification for each file, you can use path aliases to reference any part of the specification.

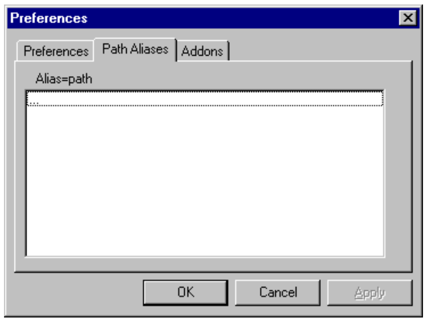
You can also use path aliases whenever automatically updating information would make maintaining your diagrams easier. For example, suppose an `animate` block referenced numerous bitmap images in C:\MYTEST\BMPS. If you moved the location of the bitmap images to C:\DIAGRAMS\BITMAPS, changing each file specification of each referenced bitmap would be a frustratingly long exercise. With path aliasing, you'd only have to update the path alias once for VisSim to correctly locate each bitmap image.

Creating path aliases

You create path aliases using the Preferences command in the Edit menu and clicking on the Path Aliases tab. The Alias=Path window lists all existing path aliases. New path aliases are entered in this window.

▶ **To create a path alias**

1. Choose Edit > Preferences.
2. Click on the Path Aliases tab.



3. In the Alias=Path window, double-click the mouse over the ellipsis. The cursor becomes an I beam.
4. Enter the path alias in the following format:

   path-alias=path

   When entering path aliases, follow the MS/DOS rules for drive and directory specifications. For example, to create an alias BmpDir that references the \BMPS directory on your C: drive, enter:

   BmpDir=C:\BMPS\PUMPS.BMP
5. Click on the OK button, or press ENTER.

Inserting path aliases in blocks

You can use path aliases in any block that references a file. When you specify a path alias, prefix it with a dollar ($) sign. For example, to use the path alias BmpDir, enter $BmpDir in the file specification box.

▶ **To insert a path alias**

1. In the Properties dialog box for the block, position the insertion point where you want to insert the path alias. (Typically, this is the File Name, Name, Bitmap Image, or Image box.)
2. Enter the path alias, prefaced with a $ and followed by a backslash; then the file name.
3. Close the dialog box.

Maintaining an edit history

The Diagram Information command in the File menu helps you keep track of important information about a block diagram as it is being developed. You can list the author's name and attach comments or an edit history to the block diagram. You can also identify the block diagram by a longer, more descriptive name. The name appears in File Open and File Add dialog boxes when you select its DOS file name.

The Diagram Information command also maintains statistics about the block diagram, including its DOS file name, its byte and block size, its last modification date, and the version of VisSim used to create it. Note that the Byte Size and Last Modified fields are not updated until you save the block diagram.

▶ **To add or view diagram information**

1. Open the block diagram whose information is to be added to or viewed.
2. Choose File > Diagram Information.
3. You can add or change information in the Title, Author, and Comment boxes. The statistical information can be viewed, but not edited.
4. When you finish adding or viewing diagram information, click on the OK button, or press ENTER.

You can add or revise diagram information for the current block diagram at any time.

Protecting your work

VisSim offers several levels of protection for your work:

- You can assign your block diagram a password to keep other users from opening the diagram. You can also request or require that they open the diagram in read-only mode.
- In large project development, where multiple users are working on the same diagram, you can assign password protection to particular parts of the diagram to prevent other users from viewing the information. You can also request or require that they view the information in read-only mode.

If you decide to use a password to restrict access, make sure to write it down exactly as you entered it — passwords are case sensitive — and store it in a safe place. Without the password, even you can't access the information.

Protecting block diagrams

To assign a password to a block diagram and set options that control how much access other users have to the diagram, choose the Diagram Information command in the File menu.

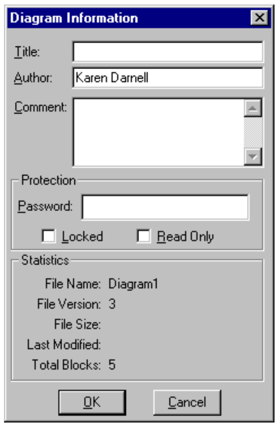


Diagram Information

Title:
Author: Karen Darnell
Comment:

Protection
Password:
☐ Locked ☐ Read Only

Statistics
File Name: Diagram1
File Version: 3
File Size:
Last Modified:
Total Blocks: 5

OK Cancel

Password Protected: To prevent other users from opening the block diagram, type a password in the Password box and activate the Locked option. Only users who know the password can open the diagram and make changes to it.

Read-Only Password Protection: To allow other users to open the diagram, but prohibit them from making changes, type a password in the Password box and activate the Read Only option. Only users who know the password can open the diagram. Once opened, the diagram can only be viewed, however, it cannot be changed.

Read-Only Requested Protection: To recommend, but not require, that other users only view a diagram without making changes to it, activate the Read Only check box. Although the diagram is opened in read-only mode, any user can de-activate the read-only protection and edit the diagram.

▶ **To restrict access to a block diagram**

1. Open the block diagram to which restricted access is to be applied.
2. Choose File > Diagram Information.
3. Do one of the following:
    - To lock the diagram closed, enter a password in the Password box and activate the Locked parameter.

- To make the block diagram read-only, enter a password in the Password box and activate the Read Only parameter.

  A password can contain up to 10 characters and can include any combination of letters and numbers. VisSim echoes an asterisk (*) for each character you type. Passwords are case sensitive.

4. Click on the OK button, or press ENTER.
5. VisSim asks you to re-enter the password for verification.

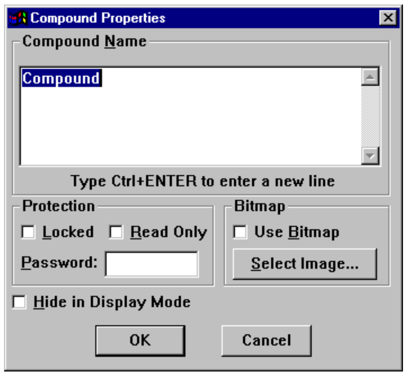
▶ **To change or delete a password**

1. Choose File > Diagram Information command.
2. In the Password box, select the row of asterisks that represent the existing password and do one of the following:
   - To change the password, type in a new password.
   - To delete the password, press the DEL key.
3. Click on the OK button, or press ENTER.

   If you entered a new password, VisSim asks you to re-enter the new password for verification.

Protecting compound blocks

Restricting access to a compound block is similar to restricting access to a block diagram. You have the choice of password protection, read-only password protection, and read-only requested protection. You enter the level of protection in the Compound Properties dialog box.



▶ **To restrict access to a compound block**

1. Choose Edit > Block Properties.
2. Point to the compound block you want protected and click the mouse.
3. Do one of the following:
   - To lock the compound block, activate the Locked parameter and enter a password in the Password box.
   - To make the compound block read-only, activate the Read Only parameter and enter a password in the Password box.

   A password can contain up to 10 characters and can include any combination of letters and numbers. VisSim echoes an asterisk (*) for each character you type. Passwords are case sensitive.
4. Click on the OK button, or press ENTER.
5. VisSim asks you to re-enter the password for verification.

▶ **To change or delete a password**

1. Choose Edit > Block Properties.
2. Point to the compound block whose password you want to change and click the mouse.
3. In the Password box, select the row of asterisks that represent the existing password and do one of the following:
   - To change the password, type in a new password.
   - To delete the password, press the DEL key.
4. Click on the OK button, or press ENTER.

   If you entered a new password, VisSim asks you to re-enter the new password for verification.

▶ **To edit a read-only compound block with password protection**

1. Choose Edit > Block Properties.
2. Point to the compound block that is read-only with password protection and click the mouse.
3. In the Password box, select the row of asterisks that represent the existing password and type in the correct password.
4. Click on the Read Only attribute to de-select it.
5. Click on the OK button, or press ENTER.

Protecting embed blocks

Password locking is a mechanism that prevents other users from drilling into and viewing the contents of an embedded diagram. Only users who know the password can unlock the `embed` block and view its contents.

Password locking is inherited the compound block in the source file. In other words, you do not apply protection to the `embed` block itself, but rather to the compound block in the source file.

▶ **To drill into a protected embed block**

1. Choose Edit > Block Properties.
2. Click the mouse over the `embed` block.
3. Enter the password.
4. Click on the OK button, or press ENTER.

▶ **To protect an embedded block diagram**

1. Open the source file that contains the compound block to be protected.
2. Choose Edit > Block Properties.
3. Click the mouse over the compound block.
4. Activate the Locked parameter and enter a password in the Password box.

   A password can contain up to 10 characters and can include any combination of letters and numbers. VisSim echoes an asterisk (*) for each character you type. Passwords are case sensitive.

   If you do not enter a password, a user can subsequently unlock the `embed` block.
5. Click on the OK button, or press ENTER.
6. VisSim asks you to re-enter the password for verification.

▶ **To change or delete a password**

1. Open the source file that contains the compound block to whose password is be changed or deleted.
2. Choose Edit > Block Properties.
3. Point to the compound block and click the mouse.
4. In the Password box, select the row of asterisks that represent the existing password and do one of the following:
   - To change the password, type in a new password.
   - To delete the password, press the DEL key.

5. Click on the OK button, or press ENTER.

   If you entered a new password, VisSim asks you to re-enter the new password for verification.

Chapter 12

Block Reference

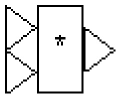
The Blocks menu lists the standard blocks provided with VisSim. When you click on the Blocks menu, most of the items that appear have a filled triangle (▶) next to them. These items are block categories. Click on a block category and a cascading menu appears listing the additional blocks.

To make it easier to find blocks in this chapter, they are presented in alphabetical order, regardless of their block category. For most blocks, a mathematical function is included. The following table translates the symbols that may appear in the function:

Symbol	**What it represents**
A	Amplitude
e	Naperian constant
dt	Derivative with respect to time
lb	Lower bound
mod	Modulus
s	Laplacian operator
t	Time
ub	Upper bound
ω	Frequency
x	Input signal
y	Output signal

Input signals are represented as $x_1, x_2, \ldots x_n$, where x_1 represents the topmost signal entering the block. When n is omitted, x_1 is assumed. Output signals are represented as $y_1, y_2, \ldots y_n$, where y_1 represents the topmost signal exiting the block. When n is omitted, y_1 is assumed.

* (multiply)



$y = x_1 * x_2 * \ldots * x_n$

Block Category: Arithmetic

The * block produces the product of the input signals. Inputs can be scalars or vectors.

VisSim assigns ones to all unconnected inputs.

Multiplying vectors and matrices

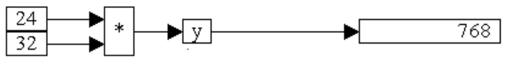
To perform a single value summation of an element-by-element multiply of two vectors, use the `dotProduct` block, as described on page 181.

To multiply two matrices, use the `multiply` block, as described on page 220.

Examples

1. Multiplication of two scalar inputs

Consider the equation $y = 24 * 32$, which can be realized as:



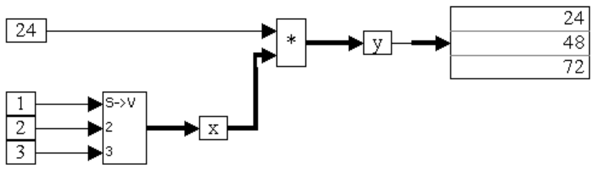
Two `const` blocks provide the values 24 and 32. When connected to a * block, the product is 768.

2. Multiplication of a scalar and a vector

Consider the equation

$y = 24\,\mathbf{x}$

where $\mathbf{x} = [1\ 2\ 3]$. This equation can be realized as shown on the next page.



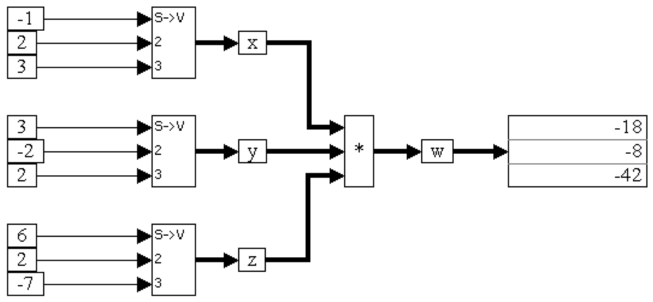
A `scalarToVec` block creates a three-element vector from the constant values 1, 2, and 3. When the simulation runs, the * block multiplies all the elements of the incoming vector line with the constant value 24.

3. Multiplication of vectors

Consider the equation:

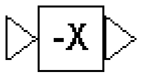
w = **x y z**

where **x** = [-1 2 3], **y** = [3 -2 2], and **z** = [6 2 -7]. This equation can be realized as:



When the simulation runs, the * block performs an element-by-element multiplication operation on the incoming vectors. For example, **w**(1) = **x**(1) * **y**(1) * **z**(1), **w**(2) = **x**(2) * **y**(2) * **z**(2), and so on.

-X (negate)



$y = -x$

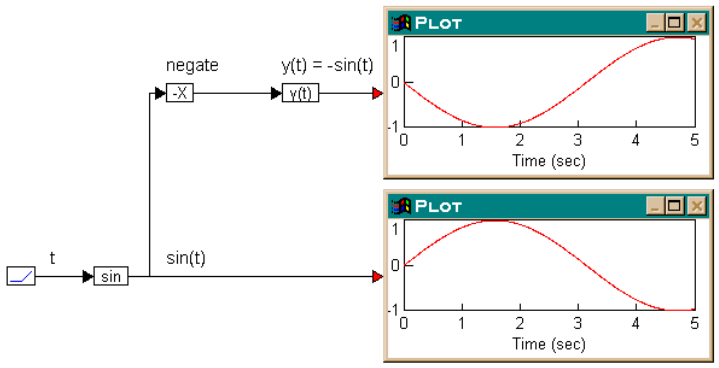
Block Category: Arithmetic

The `-x` block negates the input signal. Input can be scalar, vector, or matrix.

Examples

1. Negation of a scalar

Consider the equation $y(t)$ = - sin(t), which can be realized as:



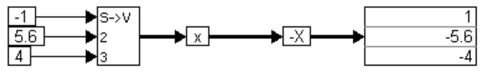
A `ramp` block is used to access simulation time t, a sin block generates sin(t), and a `-X` block converts sin(t) to -sin(t). Both sin(t) and $y(t)$ are plotted for comparison.

2. Negation of a vector

Consider the equation:

z = -**x**

where **x** = [-1 5.6 4]. This equation can be realized as:



A `scalarToVector` block creates a three-element vector from the constant values -1, 5.6, and 4. When the simulation runs, the `-X` block performs an element-by-element negate operation on the incoming vector.

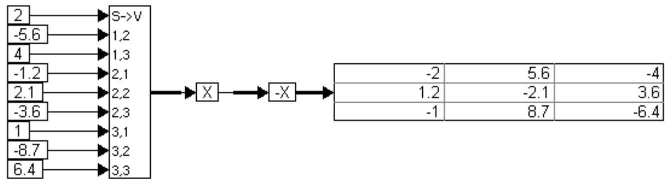
3. Negation of a matrix

Consider the equation:

Z = -**X**

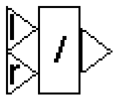
$$\text{where } \mathbf{X} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$$

This equation can be realized as:



When the simulation runs, the `-X` block performs an element-by-element negate operation on the incoming matrix.

/ (divide)



$$y = \frac{x_1}{x_2}$$

Block Category: Arithmetic

The `/` block produces the quotient of the input signals. The inputs can be scalars or vectors. On the connector tabs, “l” represents the numerator x_1 and “r” represents the denominator x_2. If x_1 is unconnected, VisSim feeds it a zero. If x_2 is equal to 0 or unconnected, VisSim displays a “Divide by 0” message and highlights the offending block in red.

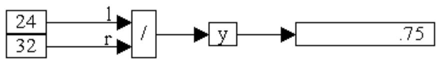
Performing matrix inversions

To perform matrix inversions, use the `invert` block, as described on page 198.

Examples

1. Division of two scalar inputs

Consider the equation $y = 24/32$, which can be realized as:

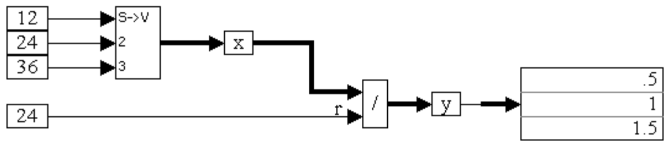


2. Division of a vector by a scalar

Consider the equation:

y = **x**/24

where **x** = [12 24 36]. This equation can be realized as:



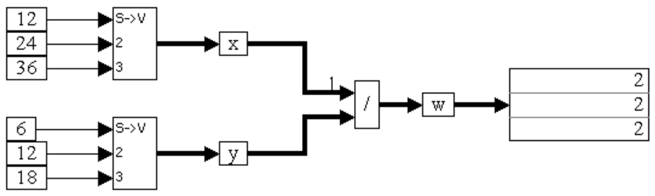
When the simulation runs, the / block divides each element of vector **x** with the constant value of 24.

3. Division of vectors

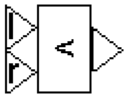
Consider the equation:

w = **x**/**y**

where **x** = [12 24 36] and **y** = [6 12 18]. This equation can be realized as:



When the simulation runs, the / block performs an element-by-element division operation on the incoming vectors. For example, **w**(1) = **x**(1)/**y**(1), **w**(2) = **x**(2)/**y**(2), and so on.



< (less than)

$$y = \begin{cases} 1 & \text{if } x_1 < x_2 \\ 0 & \text{if } x_1 \geq x_2 \end{cases}$$

Block Category: Boolean

The < block produces an output signal of 1 if and only if input signal x_1 is less than input signal x_2. Otherwise, the output is 0. On the connector tabs, "l" represents x_1 and "r" represents x_2.

If you click the right mouse button over the < block, the Boolean block menu appears allowing you to assign a different function to the block.

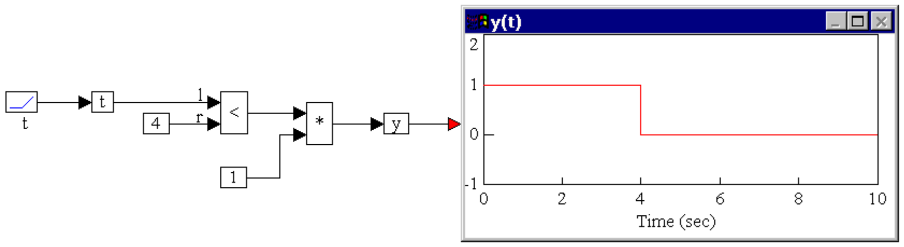
Examples

1. Simple if-then-else construct

Consider a variable y such that:

If $t < 4$ then $y = 1$; else $y = 0$

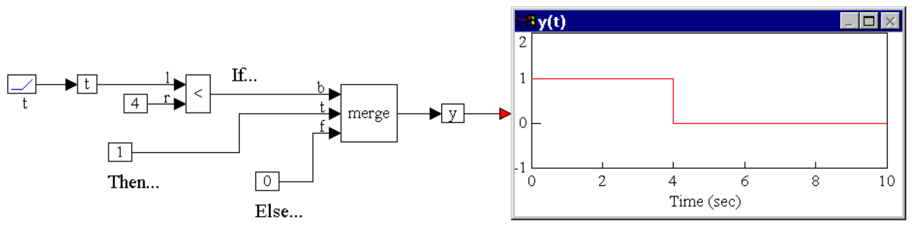
Assume that t is simulation time. This system can be realized as:



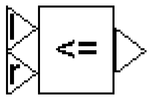
By multiplying a constant value 1 with the output of the < block, y is guaranteed to assume a value of 0 until the inequality is true. When the inequality is true, y assumes a value equal to the output of the * block.

2. Modified if-then-else construct

The previous example can also be realized as:



The key difference in implementation is the use of a `merge` block rather than a * block. The `merge` block explicitly depicts the if-then-else structure; the * block is a shortcut and can lead to confusion.



<= (less than or equal to)

$$y = \begin{cases} 1 & \text{if } x_1 \le x_2 \\ 0 & \text{if } x_1 > x_2 \end{cases}$$

Block Category: Boolean

The <= block produces an output signal of 1 if and only if input signal x_1 is less than or equal to input signal x_2. Otherwise, the output is 0. On the connector tabs, “l” represents x_1 and “r” represents x_2. The <= block accepts two scalar inputs.

If you click the right mouse button over the <= block, the Boolean block menu appears allowing you to assign a different function to the block.

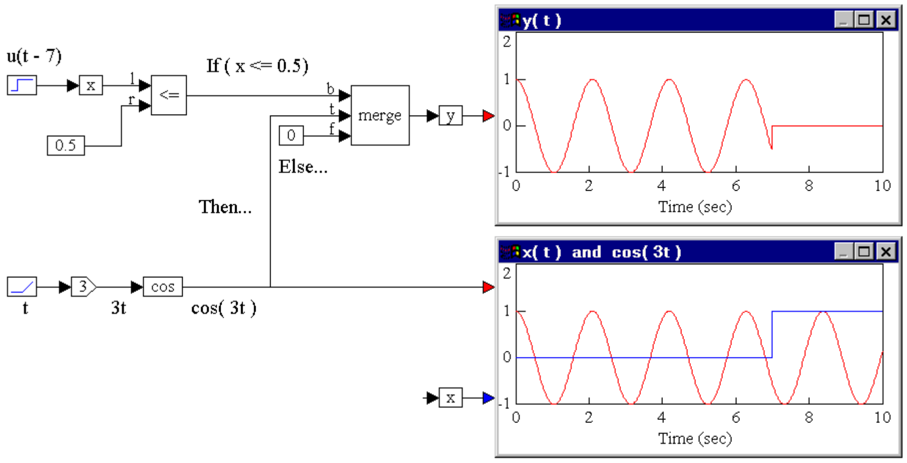
Examples

1. Simple if-then-else construct

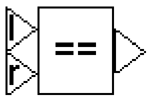
Consider a variable *y* such that:

If $x \le 0.5$ then $y = \cos(3t)$; else $y = 0$

where *t* is simulation time. Let *x* be a unit step delayed by 7 sec, represented as $u(t - 7)$. This system can be realized as shown on the next page.



Until the onset of the step input at $t = 7$ sec, the Boolean inequality $x \leq 0.5$ evaluates to true, and y takes on a value of cos(3t). At $t = 7$ sec, the Boolean inequality evaluates to false and remains false for the duration of the simulation. Consequently, from this point onwards, y takes on the value of 0. The lower `plot` block monitors the outputs of the `cos` and `variable` x blocks.



== (equal to)

$$y = \begin{cases} 1 & \text{if } x_1 = x_2 \\ 0 & \text{if } x_1 \neq x_2 \end{cases}$$

Block Category: Boolean

The == block is useful for evaluating the Boolean == equality. This block accepts two scalar inputs labeled “l” (for x_1)and “r” (for x_2). The output of the == block is 1 if and only if input “l” is identically equal to input “r;” otherwise, the output is zero.

Boolean equality comparisons of floating point variables and non-integer constants

As with programming in any language, it is generally not a good idea to perform Boolean equality comparisons involving floating point variables and non-integer constants. These types of comparisons should be converted to Boolean inequality comparisons. (For example, {If position is equal to π, then…}) should be converted to {If position is greater than or equal to < π rounded off>, then…}.) The reason for this is because a floating point variable, such as position, is rarely exactly equal to a non-zero non-integer value, particularly if it is obtained by solving one or more equations.

If you click the right mouse button over the == block, the Boolean block menu appears allowing you to assign a different function to the block.

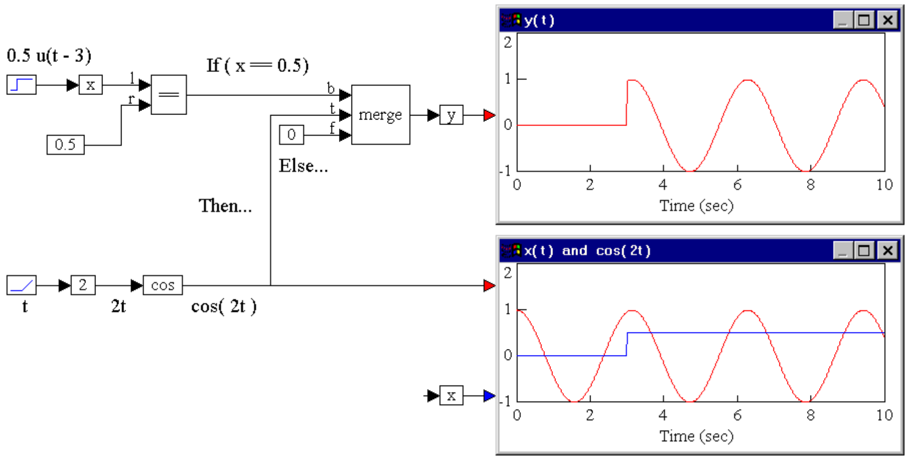
Examples

1. Comparing constants

Consider a variable *y* such that:

If *x* = 0.5 then *y* = *cos*(*2t*); else *y* = 0

where *t* is simulation time. Let *x* be a step function of amplitude 0.5, delayed by 3 sec. This is usually represented as 0.5 *u*(*t* - 3). This system can be realized as:



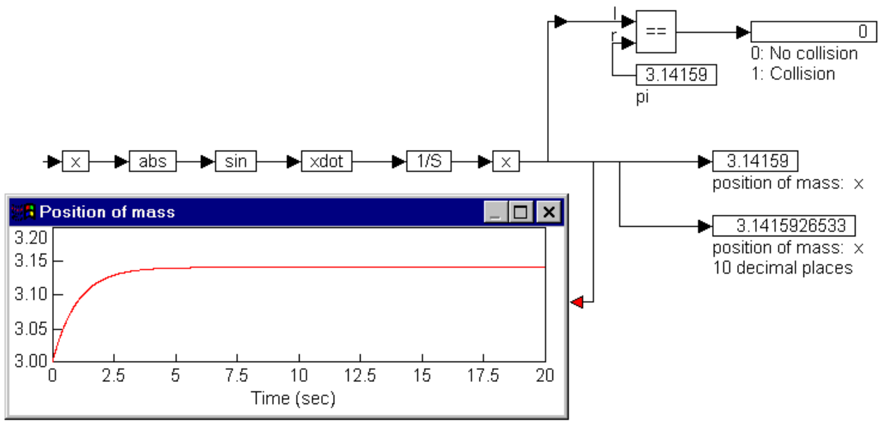
Until the onset of the step input at *t* = 3 sec, the Boolean equality x == 0.5 evaluates to false, and *y* takes on a value of 0. At *t* = 3 sec, the Boolean equality evaluates to true, and remains true for the duration of the simulation. Consequently, from this point onwards, *y* takes on the value of cos(2*t*). The lower `plot` block is used to monitor the outputs of the `cos` block and the `variable` *x*.

2. Comparing a floating point variable with a non-integer constant

In a collision detection problem, if position *x* of a mass in motion is equal to π, then a collision is assumed to have occurred with an immovable wall that is located at $x = \pi$. Furthermore, the position of the mass is assumed to be given by the solution of the following first order differential equation:

$$\dot{x} = \sin(|x|)$$

The initial condition is assumed to be $x(0) = 3.0$. It is tempting to realize this system as:



From the result shown in the `plot` block, at around $t = 7$ sec, the mass arrives at the boundary located at π. However, the collision detection logic, using an == block that compares *x* with a constant value of π, never detects the collision. This is because the final mass position, as obtained from the output of the `integrator`, is 3.141592653, which is not equal to 3.14159.

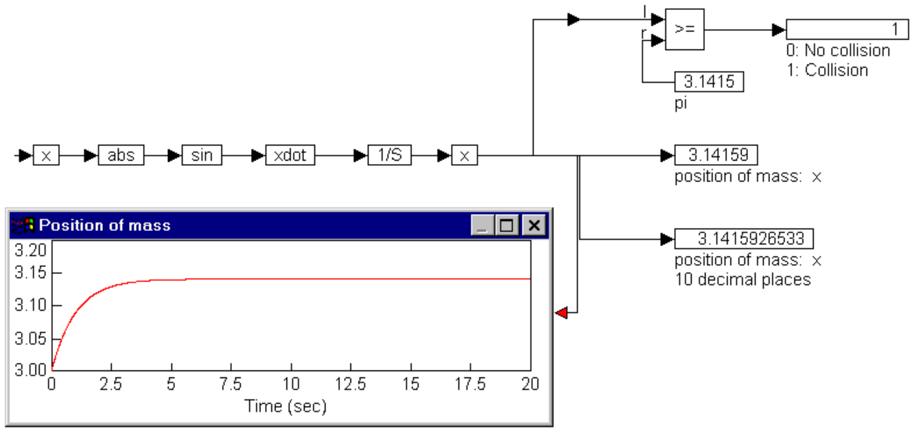
It is clear from the `plot` block, that for all practical purposes, the mass collided with the wall around $t = 7$ sec. To capture this reality in the simulation, convert the Boolean equality comparison:

If $x = 3.14159$ then…

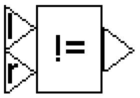
to a Boolean inequality comparison:

If $x \geq 3.1415$ then…

After reducing the `const` block to four decimal places with no round-off, the system can be realized as:



Except for replacing the == block with the >= block, this diagram is similar to the previous one. In this case, the collision detection logic detects a collision around $t = 8$ sec. Obviously, the time at which the collision is detected depends on the number of decimal places retained for the π approximation.



!= (not equal to)

$$y = \begin{cases} 1 & \text{if } x_1 \neq x_2 \\ 0 & \text{if } x_1 = x_2 \end{cases}$$

Block Category: Boolean

The != block produces an output signal of 1 if and only if the two scalar input signals are not equal. On the connector tabs, “l” represents x_1 and “r” represents x_2.

If you click the right mouse button over the != block, the Boolean block menu appears allowing you to assign a different function to the block.

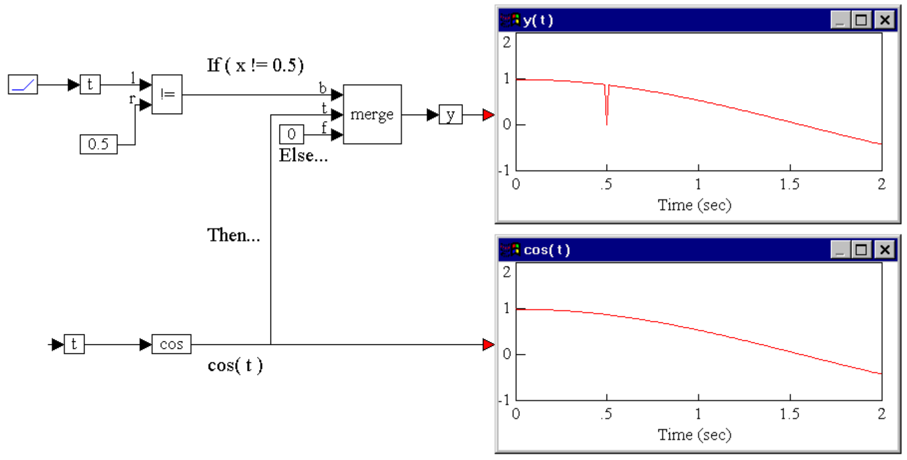
Examples

1. Comparing constants

Consider a variable *y* such that:

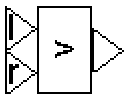
If *t* ≠ 0.5 then *y* = cos(*t*); else *y* = 0

where *t* is simulation time. This system can be realized as shown on below.



Until the value of *t* reaches 0.5, the Boolean inequality *t* ≠ 0.5 evaluates to true, and *y* takes on a value of cos(*t*). At *t* = 0.5 sec, the Boolean inequality evaluates to false, and at the very next time step, returns to true, and remains true for the duration of the simulation. Consequently, at the moment *t* = 0.5 sec, *y* takes on the value of 0, and at every other point, *y* is equal to cos(*t*).

> (greater than)



$$y = \begin{cases} 1 & \text{if } x_1 > x_2 \\ 0 & \text{if } x_1 \le x_2 \end{cases}$$

Block Category: Boolean

The > block is useful in evaluating the Boolean > inequality. It accepts two scalar inputs, labeled "l" and "r." The output of the > block is 1 if and only if input "l" > input "r;" otherwise the output is zero.

If you click the right mouse button over the > block, the Boolean block menu appears allowing you to assign a different function to the block.

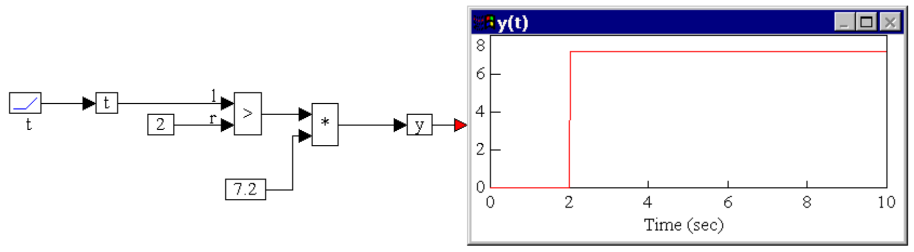
Examples

1. Simple if-then-else construct

Consider a variable *y* such that:

If *t* > 2 then *y* = 7.2; else *y* = 0

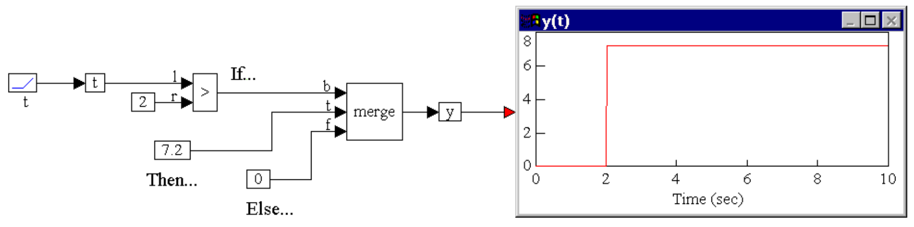
Assume that *t* is simulation time. This system can be realized as:



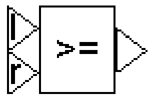
By multiplying a constant value of 7.2 with the output of the > block, *y* is guaranteed to assume a value of 0 until the inequality is true. When the inequality is true, *y* assumes a value equal to the output of the * block.

2. Modified if-then-else construct

Using the above equation, it can also be realized as:



The key difference in implementation is the use of a `merge` block rather than a * block. The `merge` block explicitly depicts the if-then-else structure, whereas the * block is a shortcut and can lead to confusion.



>= (greater than or equal to)

$$y = \begin{cases} 1 & \text{if } x_1 \geq x_2 \\ 0 & \text{if } x_1 < x_2 \end{cases}$$

Block Category: Boolean

The >= block produces an output signal of 1 if and only if input signal x_1 is greater than or equal to input signal x_2. Otherwise, the output is 0. On the connector tabs, "l" represents x_1 and "r" represents x_2. The >= block accepts two scalar inputs.

If you click the right mouse button over the >= block, the Boolean block menu appears allowing you to assign a different function to the block.

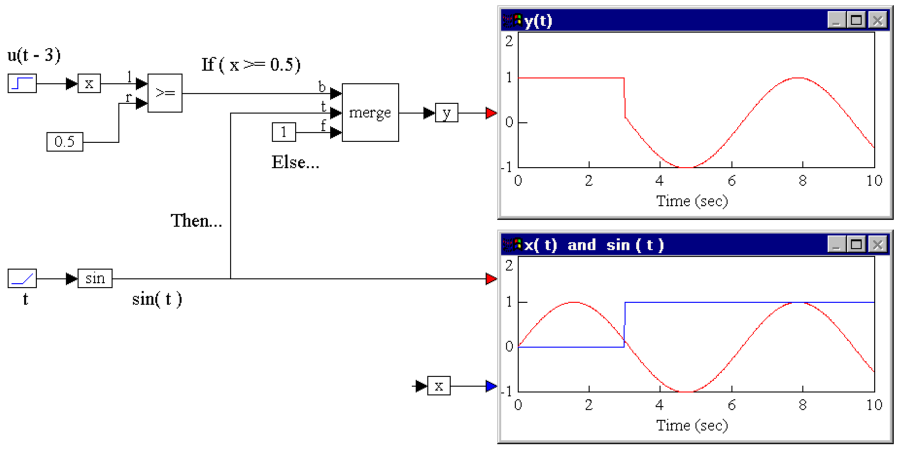
Examples

1. Simple if-then-else construct

Consider a variable y such that:

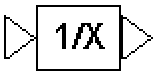
If $x \geq 0.5$ then $y = \sin(t)$; else y = 1

where t is simulation time. Let x be a unit step delayed by 3 sec. This is usually represented as $u(t - 3)$. This system can be realized as:



Until the onset of the step input at $t = 3$ sec, the Boolean inequality $x \geq 0.5$ evaluates to false and y takes on a value of 1. At $t = 3$ sec, the Boolean inequality evaluates to true and remains true for the duration of the simulation duration. Consequently, from this point onwards, *y* takes on the value of sin(*t*).

1/X (inverse)



$$y = \frac{1}{x}$$

Block Category: Arithmetic

The `1/X` block produces the inverse of the input signal. The input can be scalar, vector, or matrix.

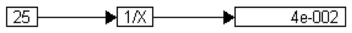
Computing the matrix inverse of a matrix

Use the `invert` block to compute the matrix inverse of a matrix. If a vector or matrix is fed into an `1/X` block, the result will be an element-by-element inversion of the vector or matrix (that is, [one divided by the element] operation). This is not equivalent to a normal vector pseudo-inverse operation or a normal matrix inverse operation.

Examples

1. Computation of 1/X of a scalar

Consider the equation *y*= 1/25, which can be realized as:



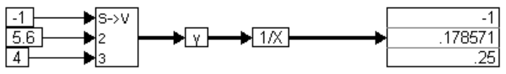
The incoming constant value of 25 results in 1/25 = 0.04.

2. Computation of 1/X of a vector

Consider the equation:

z = 1/**y**

where **y** = [-1 5.6 4], and where an element-by-element inversion is implied. This equation can be realized as:



An element-by-element inverse operation is performed on the three elements in the `scalarToVec` block.

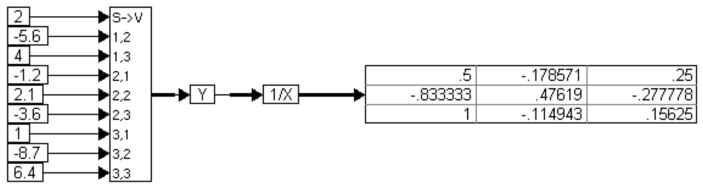
3. Computation of 1/X of a matrix

Consider the equation:

Z = 1/ **Y**

$$\text{where } \mathbf{Y} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$$

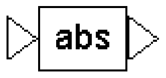
This can be realized as:



.5	-.178571	.25
-.833333	.47619	-.277778
1	-.114943	.15625

When the simulation runs, the `1/X` block performs an element-by-element inverse operation on the incoming matrix.

abs



$y = |x|$

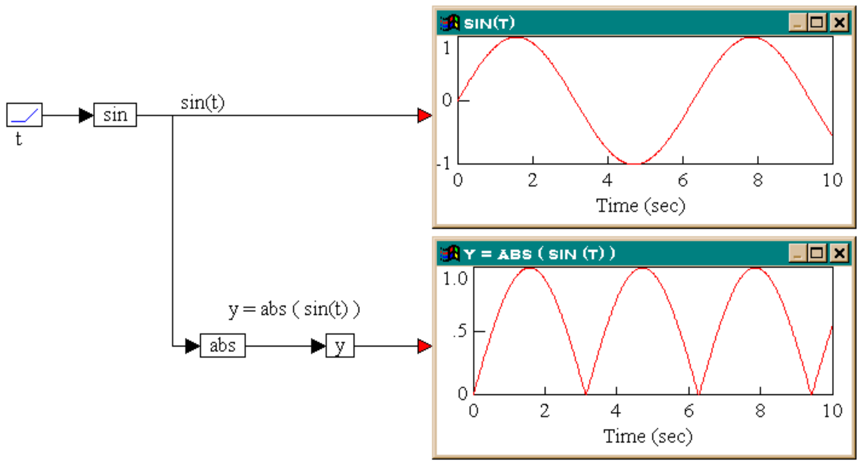
Block Category: Arithmetic

The `abs` block produces the absolute value of the input signal. The inputs can be scalars, vectors, or matrices.

Examples

1. Absolute value of a scalar

Consider the equation *y* = abs (sin (*t*)), which can be realized as shown on the next page.



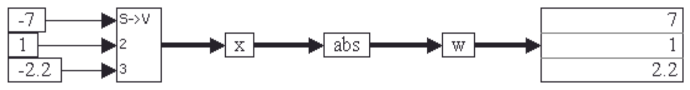
The results in the two `plot` blocks show that the `abs` block computes the absolute value of the input signal.

2. Absolute value of a vector

Consider the equation:

w = abs (**x**)

where **x** = [-7 1 -2.2]. This equation can be realized as:



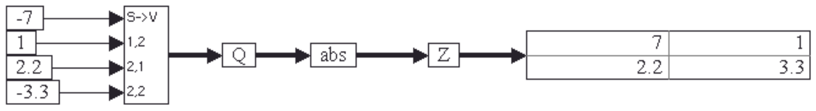
When the simulation runs, the `abs` block computes and outputs an element-by-element absolute value of the vector **x**.

3. Absolute value of a matrix

Consider the equation:

Z = abs(**Q**)

where $\mathbf{Q} = \begin{bmatrix} -7 & 1 \\ 2.2 & -3.3 \end{bmatrix}$. This equation can be realized as shown on the next page.



Four `const` blocks provide the vector element values of **Q** through a `scalarToVector` block. When the simulation runs, the `abs` block computes the element-by-element absolute value of the incoming matrix.

acos

$y = \text{arc}\cos x$

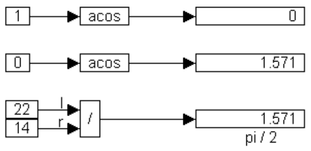
Block Category: Transcendental

The `acos` block produces the inverse cosine of the input signal. The output is an angle in radians.

Examples

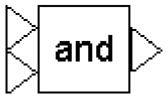
1. Computation of $\cos^{-1}(1) = 0$; $\cos^{-1}(0) = \pi/2$

This equation can be realized as:



Two `acos` blocks are used to compute the inverse cosines. For comparison, the constant value of π/2 is generated by connecting two `const` blocks, set to 22 and 14, to the “l” and “r” inputs of a `/` block. From the results obtained, the `acos` blocks yield correct values for the angles.

and



$y = x_1$ bitwise AND x_2

Block Category: Boolean

The and block produces the bitwise AND of two to 256 scalar input signals.

If you click the right mouse button over the and block, the Boolean block menu appears allowing you to assign a different function to the block.

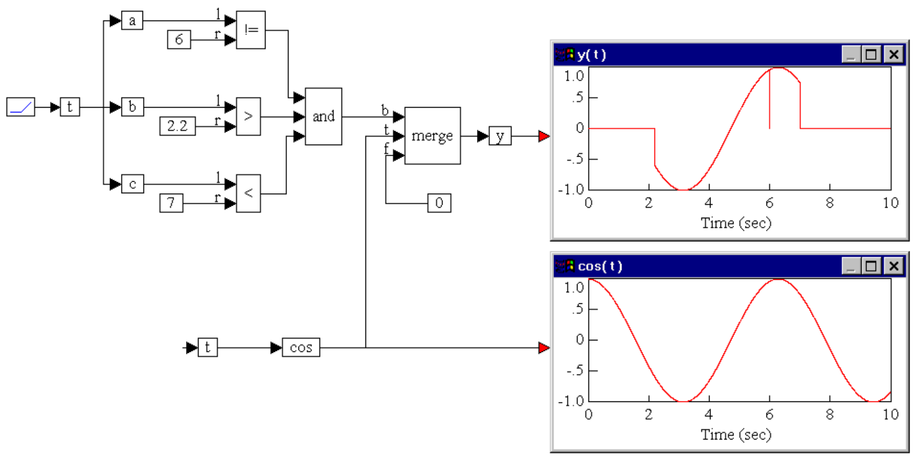
Examples

1. Three variable and

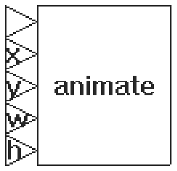
Consider a variable *y* such that:

If $a \neq 6$ and $b > 2.2$ and $c < 7$, then $y = \cos(t)$; else $y = 0$

where *t* is simulation time. Furthermore, let *t* be the input to all three parameters *a*, *b*, and *c*. This system can be realized as:



The output of the and block is true only when all the three inputs are true. This happens in the range $t = (2.2, 7)$, except for the instant $t = 6$. This result is apparent from the top plot block. The variable *y* is equal to $\cos(t)$ in the range $t = (2.2, 7)$. At the instant $t = 6$, variable *a* is momentarily false, and consequently, $y = 0$ at $t = 6$, since the output of the and block evaluates to false at that instant.



animate

Block Category: Animation

The `animate` block lets you animate an image during simulation. For more information, see page 77.

asin

$y = \text{arc}\sin x$

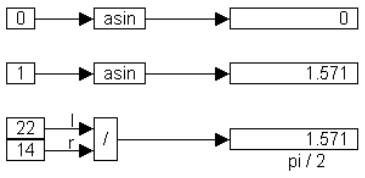
Block Category: Transcendental

The `asin` block produces the inverse sine of the input signal. The output is an angle in radians.

Examples

1. Computation of $\sin^{-1}(1) = \pi/2$; $\sin^{-1}(0) = 0$

This equation can be realized as:



Two `const` blocks, set to 0 and 1, are fed to the `asin` blocks. For comparison, the constant value of π/2 is generated by feeding two `const` blocks, set to 22 and 14, into the “l” and “r” inputs of a `/` block. From the results obtained, the `asin` blocks yield correct values for the angles.

atan2

$y = 4\text{quad}\,\text{arc}\tan\,(x_1, x_2)$

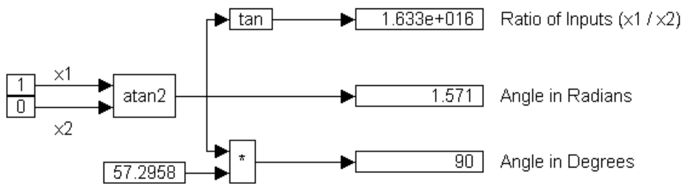
Block Category: Transcendental

The `atan2` block computes the four quadrant inverse tangent of the input signals. The `atan2` block uses the signs of both input signals to determine the sign of the output signal. The output is an angle in radians.

Examples

1. Computation of tan^{-1}(∝) = π/2

This equation can be realized as:

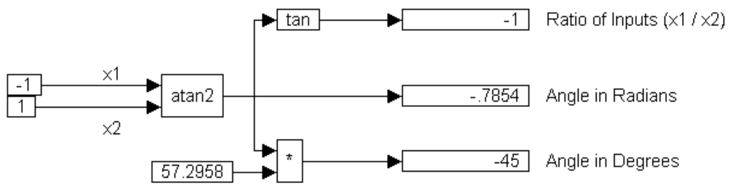


To convert radians to degrees, the angle in radians is multiplied by (180/π) = 57.2958.

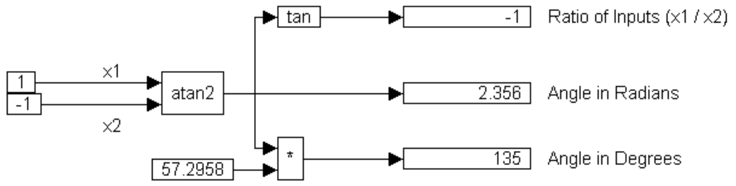
Since the `atan2` block uses the value of x_1, the signs of x_1 and x_2, and the ratio x_1/x_2 in computing the inverse tangent, the result depends on all these parameters. In the current case, since the ratio is infinity, `atan2` computes the inverse correctly to be π/2 radians, or 90^o. Also, in the current case, x_1 can be any positive value, since its ratio with 0 will be infinity, regardless of its value.

2. Computation of tan^{-1}(-1): quadrant dependency

Using the same configuration in the above example, tan^{-1}(-1) can be realized as:



Here, the angle obtained is -.7854 radians, or -45°, because the `atan2` block determines that the angle lies in the fourth quadrant. However, it is immediately apparent that the same ratio of -1 can be obtained by flipping the signs on x_1 and x_2:



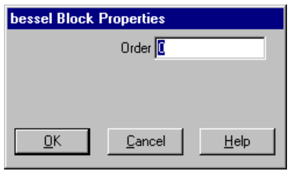
In this case, the `atan2` block uses the relative signs of x_1 and x_2 to determine that the angle lies in the second quadrant, and yields an angle of 180 - 45 = 135°, or 2.356 radians.

bessel

$y = \text{bessel}_n x$

Block Category: Transcendental

The `bessel` block generates the Bessel function of order *n*.


bessel Block Properties

Order 0

Order: Sets the order of the Bessel function. Specify the order as an integer. The default is 0.

Examples

1. Approximation of sin(a sin ϕ)

Bessel functions come up frequently in the analysis and solution of nonlinear differential equations. Consider the following approximation:

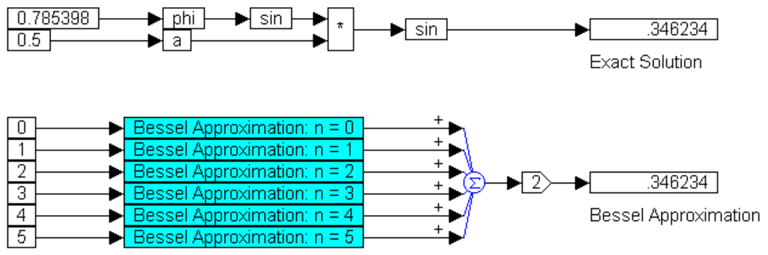
$$\sin(a \sin \phi) = 2\sum_{n=0}^{\infty} J_{2n+1}(a) \sin[(2n+1)\phi]$$

where *a* and ϕ are parameters, and J_m is a Bessel function of order *m*. Such approximations are a part of the standard procedure for obtaining approximate analytical solutions to equations of the type:

$$\ddot{u} + 2\varepsilon\mu \sin \dot{u} + u = K \cos \Omega t$$

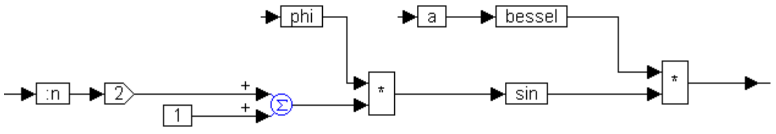
These equations are used in the harmonic analysis of forced oscillations of single degree of freedom systems.

The above approximation can be realized as:


0.785398
0.5
phi
a
sin
*
sin
.346234
Exact Solution
0
1
2
3
4
5
Bessel Approximation: n = 0
Bessel Approximation: n = 1
Bessel Approximation: n = 2
Bessel Approximation: n = 3
Bessel Approximation: n = 4
Bessel Approximation: n = 5
Σ
2
.346234
Bessel Approximation

Two `const` blocks produce π/4 and 0.5 as the values for ϕ and *a*, respectively. The sine of *phi* is multiplied by *a* and the result is fed through another `sin` block to compute the exact solution.

Six `const` blocks, set to 0 through 5, generate different terms of the infinite series approximation. In this case, only the first six terms of the series are retained. Each of these `const` blocks is connected to a compound block, which has the following internal structure:



The `const` block feeds a value to a local `variable` *:n*. The output of *:n* is connected to a `gain` block set to 2. The output of the `gain` block, and the output of a `const` block set to 1, are fed into two inputs of a `summingJunction` block. The output of the `summingJunction` block and the output of `variable` *phi* are connected to a `*` block, to compute the term (*2n+1*)ϕ. The `variable` *a* is connected to a `bessel` block whose internal order is set to the correct value (0, 3, 5, 7, 9, or 11, depending on the value of *:n*).

At the top level, the outputs of the six compound blocks are summed using a six input `summingJunction` block, and the output of the `summingJunction` block is connected to a `gain` block set to 2. The output of the `gain` block is connected to a `display` block.

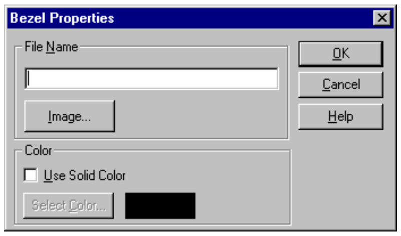
From the results obtained, it is proven that by retaining the first six terms in the approximation, very close agreement can be obtained with the actual value of sin(a sin ϕ).

bezel

Block Category: Annotation

The `bezel` block is an effective way to add background characteristics, such as operator control panels, to your screen. Designed to be used in display mode, the `bezel` block accepts bitmaps or background color specifications. When display mode is turned on, `bezel` blocks act as background and appear beneath other blocks.

When display mode is turned off, you can resize a `bezel` block by dragging on its borders. If a bitmap is associated with the `bezel` block, it initially assumes the size of the bitmap. For solid color backgrounds, the chosen color fills in the bezel area and can also be resized. When you turn on display mode, the sizing border goes away.



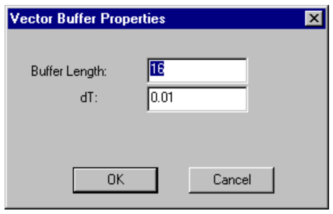
File Name: Indicates the name of the .BMP file used as the background bezel. You can type the file name directly into this box or select one using the Image button.

Color: Lets you use a solid color as the background for the bezel. To select a color, activate Use Solid Color and click on the Select Color button to choose a color. When Use Solid Color is not activated, the `bezel` block defaults to the name of the .BMP file specified in the File Name box.

buffer

Block Category: Matrix Operations

The `buffer` block places a sequence of values in a buffer based on the buffer length, the time between successive samples, and the duration of the simulation. The buffer block accepts a single scalar input and produces a single vector output. It is useful for performing basic digital signal processing operations.



Buffer Length: 16
dT: 0.01

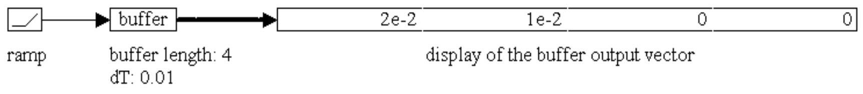
Buffer Length: Determines the number of samples; that is, the size of the buffer.

dT: Determines the time between successive samples; that is, the rate at which samples of the incoming signal are collected and placed in the buffer.

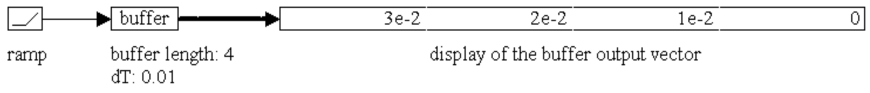
Examples

1. Basic buffer operation

Consider the following `buffer` block, with a buffer length of 4 and time between successive samples of 0.01.



For simplicity, let the simulation step size be equal to 0.01. If the input to the `buffer` is an arbitrary non-zero signal, such as a `ramp` signal, then after two simulation time steps, the output of `buffer` is a vector of length 4, with the first two elements containing non-zero values and the remaining two still at zero. At the very next time step, the simulation appears as:

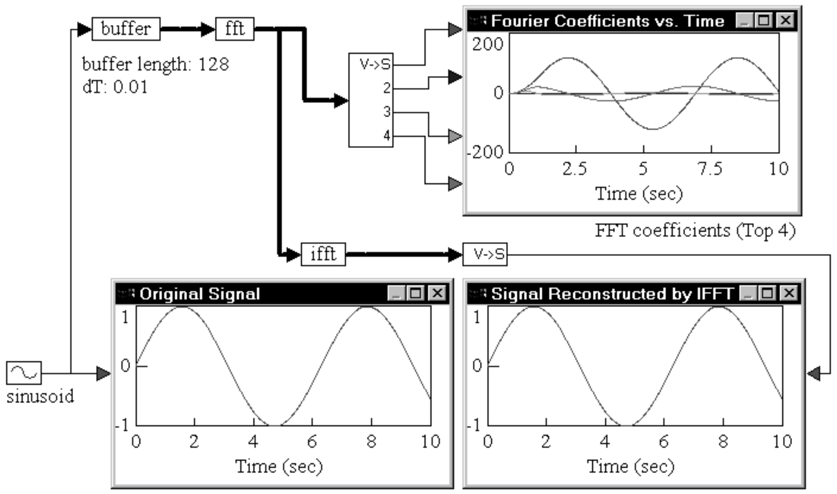


The previous values are pushed down the vector by one cell, and the top cell is occupied by the latest sample. Once the simulation goes beyond four time steps, the output will be a full vector.

Obviously, if the input signal itself is zero for some points, those values will be reflected accurately in the output.

2. Computation of FFT and inverse FFT

Consider a simple example, where a sinusoidal signal is converted to frequency domain via FFT, and then reconstructed using the IFFT.



A `sinusoid` block generates a sinusoid signal with a frequency of 1 rad/sec. The signal is passed through a `buffer` block of length 128 samples and a time between successive samples of 0.01. The output of the `buffer` block is connected to an `fft` block, which computes a 128-sample FFT of the original sinusoid at a sampling rate of 0.01.

The output of the `fft` block is Fourier coefficients. The individual coefficients are accessed using a `vecToScalar` block. The first four coefficients are plotted to show their variation with time.

Signal reconstruction is performed by feeding the output of the `fft` block to an `ifft` block to compute the IFFT. The output of the `ifft` block is a vector of length 128 samples. The contents of this vector are just 128 sinusoid reconstructions, with each sinusoid trailing the preceding sinusoid by an amount equal to the sampling rate.

The first element in the `ifft` output vector does not have any delay because zero time has elapsed between the FFT and IFFT phases. In most real-world situations, however, there is a small, non-zero delay between the input signal and its reconstruction that is introduced by the processor performing the numerical computations of FFT and IFFT algorithms.

button

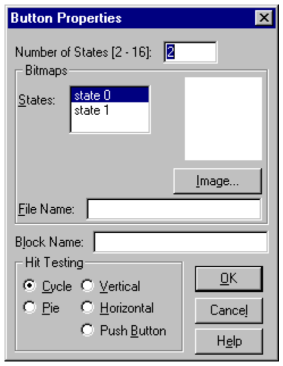
$y = \text{state} - 1$

Block Category: Signal Producer

The `button` block lets you dynamically insert signal values during a simulation.

You can set the number of states that a `button` block has, from two to a maximum of 16. You can also associate a bitmap with any of the states. The `button` block toggles between white and red if it is a 2-state button and there are no bitmaps associated with it.

The `button` block also provides cycle through, pie area, push button horizontal, and vertical hit testing.


Button Properties

Number of States [2 - 16]: 2

Bitmaps

States: state 0, state 1

File Name:

Block Name:

Hit Testing: [✓ Cycle] Vertical Pie Horizontal Push Button

Number Of States [2-16]: Indicates the number of states for the `button` block. The maximum allowable states is 16. The number of listed states in the States box is determined by the value entered in the Number of States box.

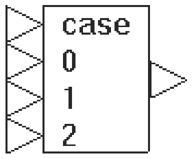
Bitmaps: Lets you associate a bitmap file with the selected state. To make an association:

1. From the States box, select a state.
2. In the File Name box, enter the bitmap file name to be associated with the state. If you do not know the name or location of the file, click on the Image button to select one. A picture of the selected bitmap appears to the right of the States box.

Block Name: Indicates a name for the `button` block. The name appears only when there is no bitmap associated with the block.

Hit Testing: You have the choice of five hit testing methods.

- **Cycle Through:** Causes the state to increase by 1 each time you click the right mouse button over the block. When the maximum state value is reached, the next mouse click changes the state back to 0.
- **Pie:** Divides the block into a number of pie-shaped wedges equal to the number of states. When you click the right mouse button over a particular wedge or when you drag the mouse over the `button` block, the state changes. States advance in a clockwise direction with State 0 at the top.
- **Vertical**: Divides the block into a number of vertical wedges equal to the number of states. When you click the right mouse button over a particular wedge or when you drag the mouse over the `button` block, the state changes. State 0 corresponds to the bottom wedge; state *N* corresponds to the top wedge.
- **Horizontal:** Divides the block into a number of horizontal wedges equal to the number of states. When you click the right mouse button over a particular wedge or when you drag the mouse over the `button` block, the state changes. State 0 corresponds to the leftmost wedge; state *N* corresponds to the rightmost wedge.
- **Push Button:** Activates state 1 while you hold down the mouse button. When you release the mouse button, it activates state 0. Push button hit testing only supports a two-state `button` block.



case

$y = x_{n+2}$

Block Category: Nonlinear

The `case` block lets you specify an unlimited number of execution paths based on the value of a single input, called the case input value. The case input value is the top input to the block and is labeled *case*. The remaining inputs are the possible execution paths. They are labeled 0 through *n*.

The main application of the case block is in the construction of large nested if-else decision structures, where regular if-else constructs using Boolean blocks become too cumbersome.

When you want to output a particular element in a matrix, use the `index` block.

Case input value: The following rules apply to values fed into the connector tab labeled *case*:

- The case input value must be scalar. If a non-integer value is fed into it, it is truncated. For example, 0.999 is truncated to 0.
- If the case input value targets an out-of-range input, the `case` block returns an error. For example, an error results if the case input value is 5 for a four-element `case` block.
- If the case input value targets an unconnected input, the `case` block outputs a 0.

Scalar and matrix output: With the exception of the case input, all other inputs to the `case` block can be scalar, vector, or matrix.

Examples

1. Implementation of five scalar branches

Consider the decision tree:

If J = 0, then Y = A;

else

If J = 1, then Y = B;

else

If J = 2, then Y = C;

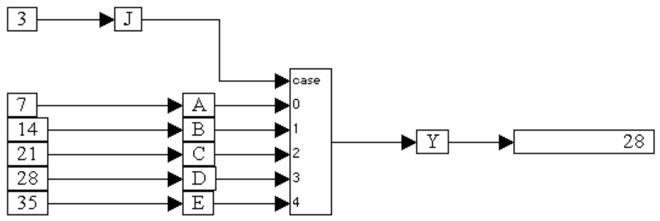
else

If J = 3, then Y = D;

else

If J = 4, then Y = E;

If A, B, C, D, and E are assumed to be constant values equal to 7, 14, 21, 28, and 35 respectively, the decision tree can be realized as:



Six `const` blocks produce values for the `variable` blocks named *J*, *A*, *B*, *C*, *D*, and *E*. The outputs of these `variables` are connected to a `case` block, with *J* connected to the case input, and `variables` *A*, *B*, *C*, *D*, and *E* connected to inputs 0, 1, 2, 3, and 4 respectively. The output of the `case` block is fed through a `variable` named *Y* and into a `display` block.

Since *J* is set to 3, the `variable` *D* is presented to the output as expected, and consequently, *Y* takes on the value of *D*, namely 28.

2. Implementation of three matrix branches

Consider the following part of the decision tree presented above:

If J = 0, then **Y** = **A**;

else

If J = 1, then **Y** = **B**;

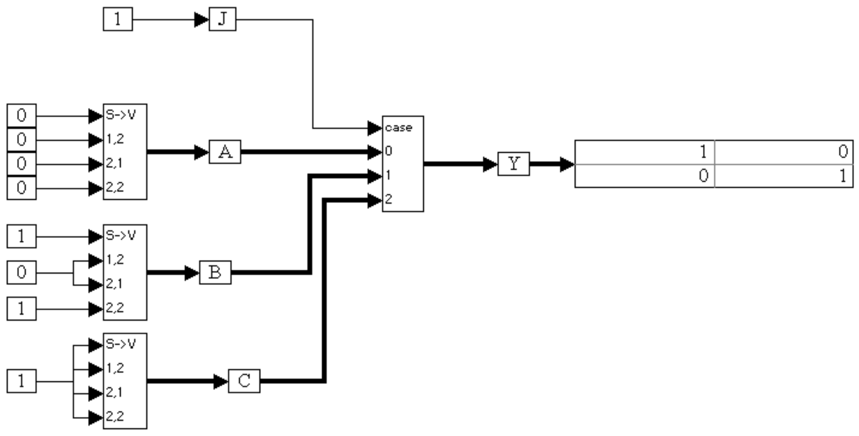
else

If J = 2, then **Y** = **C**;

If you let **A**, **B**, and **C** be:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

the decision tree can be realized as shown below.



Eight `const` blocks generate the elements of the three matrices, represented as three `scalarToVector` blocks. The `variable` *J* is set to 1 and is fed into the case input of a `case` block. The outputs of variables *A*, *B*, and *C* are wired to inputs 0, 1, and 2 of the `case` block. The output of the `case` block is connected to a `variable` **Y**, which is wired to a `display` block.

Since *J* is set to 1, the contents of variable *B* are presented at the output of the case block such that **Y** = **B**.

comment

Block Category: Annotation

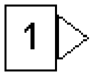
The `comment` block adds a comment to the diagram. When you position the pointer over the block and click the right mouse button, the pointer changes into a vertical I-beam, indicating that you're in text-entry mode. As you insert text, VisSim automatically scrolls the text if it runs out of room in the viewable region of the block. To correct or remove text, use the DEL and BACKSPACE keys. To exit text-entry mode, click the right mouse button on the `comment` block a second time.

You can also copy text from an application file into a `comment` block. For example, to copy text from a WORD document, highlight the text to be copied and press CTRL+V. In the `comment` block, position the I-beam where you want to insert the text and press CTRL+C.

To retain the format of the copied text, activate the Use Rich Text Format under the Preferences tab in the dialog box for the Edit > Preferences command. If Use Rich Text Format is not activated, the text will revert to the text format specified with the View > Fonts command.

When reading a comment, use the scroll bar to move text in and out of the viewable region. To resize a `comment` block, drag on its edges.

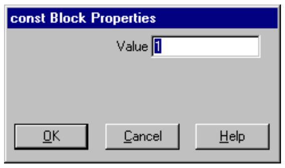
const



$y = \text{constant value}$

Block Category: Signal Producer

The `const` block generates a constant signal.


const Block Properties

Value: 1

Value: Indicates the value of the output signal. The default is 1.

constraint

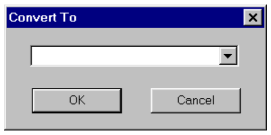
Block Category: Optimization

The `constraint` block is used to solve an implicit equation. For more information, see Chapter 8, “Performing Global Optimization.”

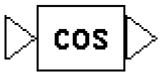
convert

Block Category: Arithmetic

The `convert` block converts the data type of the input signal to one of the following: char, unsigned char, short, unsigned short, int, long, unsigned long, float, or double. To check for overflow errors, activate Warn Numeric Overflow under the Preferences tab in the dialog box for the Simulate > Simulation Properties command.



cos



$y = \cos x$

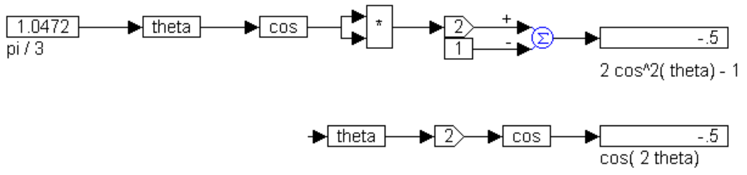
Block Category: Transcendental

The `cos` block produces the cosine of the input signal. The input signal must be represented in radians.

Examples

1. Computation of cos(2θ) = 2cos^2(θ) - 1

With θ chosen to be π/3, the above trigonometric identity can be realized as:



cosh

$$y = \frac{e^{x} + e^{-x}}{2}$$

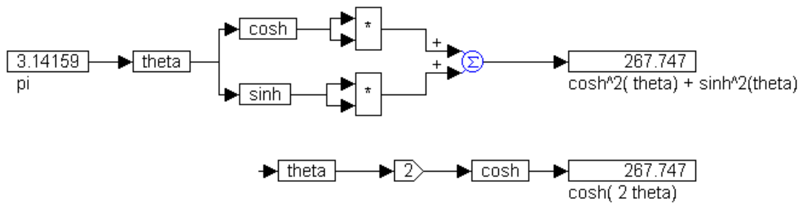
Block Category: Transcendental

The `cosh` block produces the hyperbolic cosine of the input signal. The input signal must be represented in radians.

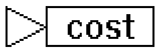
Examples

1. Computation of cosh(2θ) = cosh^2(θ) + sinh^2(θ)

With θ chosen to be π, the above trigonometric identity can be realized as shown below.



cost



Block Category: Optimization

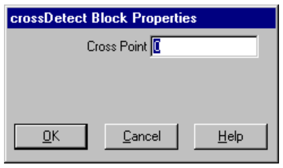
The `cost` block measures the cost function for parameter optimization. For more information, see Chapter 8, “Performing Global Optimization.”

crossDetect

$$y = \begin{cases} -1 & \text{if } x \text{ crosses crosspoint with neg. slope} \\ 1 & \text{if } x \text{ crosses crosspoint with pos. slope} \\ 0 & \text{otherwise} \end{cases}$$

Block Category: Nonlinear

The `crossDetect` block monitors its input value and compares it with a user-specified crosspoint. When the input value crosses the crosspoint, the `crossDetect` block outputs either +1 or -1, depending on whether the crossover occurred with a positive slope or negative slope, respectively. If a crossover is not detected, the `crossDetect` block outputs 0.


crossDetect Block Properties

Cross Point: 0

Cross Point: Represents the value that, when *x* crosses it, causes the output signal to go to 1, -1, or 0. The default is 0.

Examples

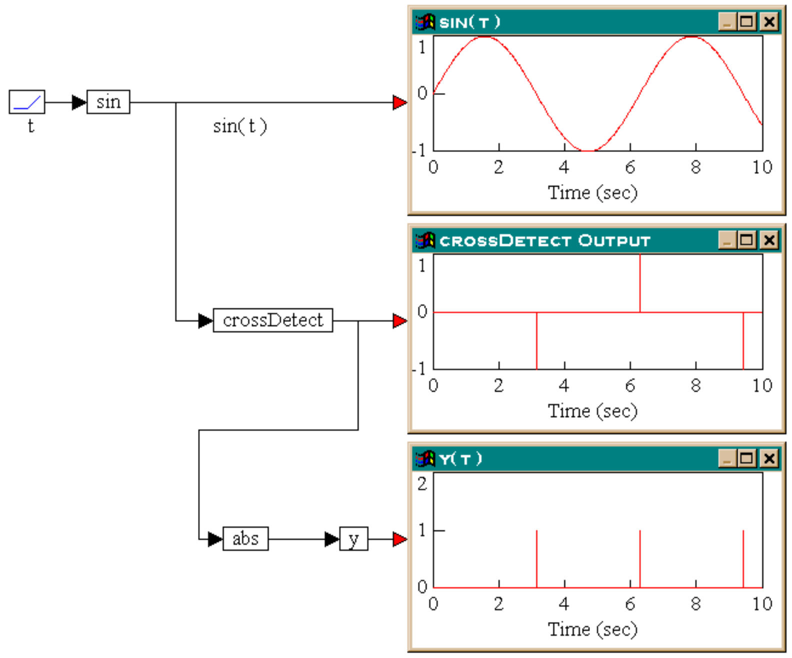
To obtain correct results from the examples described below, increase the point count for the `plot` blocks to at least 1,000.

1. **Detection of zero crossover of a sinusoid**

Consider the equation:

y = 1 if sin(t) = 0, else y = 0

This equation can be realized as:



As can be seen from the `crossDetect` block output, three 0 crossings are detected in the simulation. The first and third 0 crossings occur with negative slope (that is, the value of sin(*t*) is decreasing, as it approaches zero), while the second 0 crossing occurs with positive slope (that is, the value of sin(*t*) is increasing as it approaches zero.) Consequently, the first and third 0 crossing events are -1, and the second 0 crossing event is + 1.

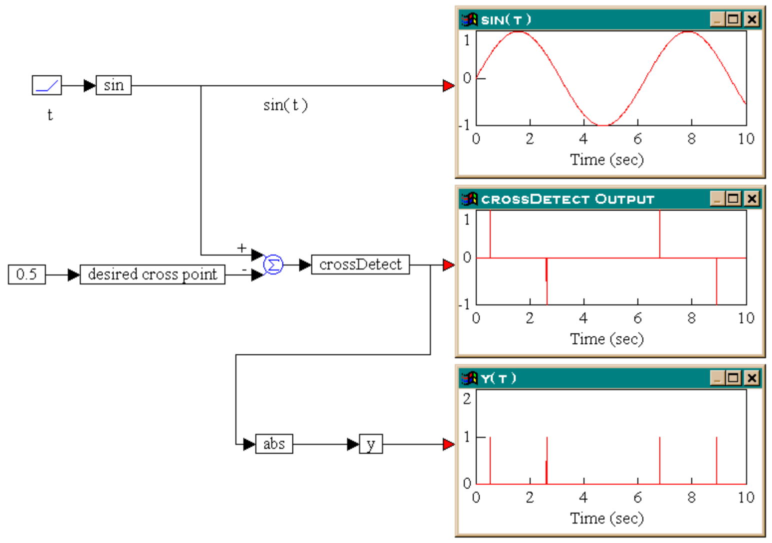
However, since *y* is required to be equal to +1 whenever sin(*t*) = 0, irrespective of the slope, the output of the `crossDetect` block is passed through an `abs` block to extract the absolute value, and this output is defined as the `variable` *y*. The bottom `plot` block shows that *y* = 1 when sin(*t*) = 0; otherwise *y* = 0.

2. Detection of non-zero crossover with externally set crosspoint

Consider the equation:

y = 1 if sin(*t*) = 0.5, else *y* = 0

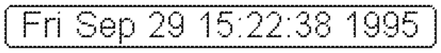
This equation can be realized exactly as above by setting the internal crosspoint on the `crossDetect` block to 0.5. Unfortunately, this may not be acceptable in some cases, particularly when the crosspoint itself is to be computed as a part of the simulation. In such cases, the crosspoint must be set externally, as shown below:



The key difference here is that the output of the `sin` block is connected to a `summingJunction` block, which computes the difference between sin(*t*) and a `variable` called *desired cross point*. This difference is connected to the `crossDetect` block, which has an internal crosspoint of 0.

In effect, a non-zero crossover detection problem is converted to a 0 crossover detection problem. That is, the problem of $y = 1$ when $\sin(t) = 0.5$ is converted to $y = 1$ when $\sin(t) - 0.5 = 0$. The rest of the diagram is identical to the previous one.

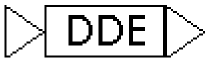
date



Block Category: Annotation

The `date` block displays the current date and time. The date and time are updated when you move a block, print the diagram, or repaint the screen. If you need to reset the time or date, use the system Control Panel. For more information, see the *Microsoft Windows User's Guide.*

DDE



Block Category: DDE

The `DDE` block exchanges information with another Windows application. Use this block when you want to create a link that sends information to and receives information from another application. You can create links between VisSim and other applications that support DDE.

For more information on the `DDE` block, see page 119.

DDEreceive

DDEreceive

Block Category: DDE

The `DDEreceive` block creates a DDE link that passes information from a Windows application (referred to as the source or server) into a block diagram (referred to as the destination or client).

For more information on the `DDEreceive` block, see page 115.

DDEsend

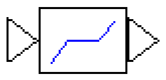
VisSim|Diagram1

Block Category: DDE

The `DDEsend` block creates a DDE link that passes information from a block diagram (referred to as the source or server) to another Windows application (referred to as the destination or client).

For more information on the `DDEsend` block, see page 117.

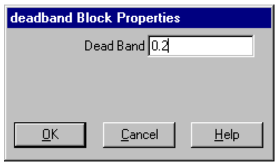
deadband



$$y = \begin{cases} 0 & \text{if}\,|x| \le \dfrac{\text{deadband}}{2} \\ x - \left(\text{sign}(x)\dfrac{\text{deadband}}{2}\right) & \text{otherwise} \end{cases}$$

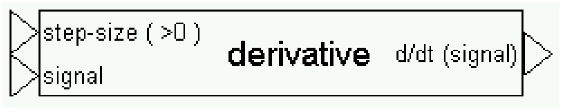
Block Category: Nonlinear

The `deadband` block produces an output signal, which is the input signal reduced by a zone of lost motion about the signal's 0 value. Use this block to simulate play in mechanical systems, such as gears or chains.


deadband Block Properties
Dead Band 0.2

Dead Band: Indicates the width of the zone of lost motion about the input signal's 0 value. The default is 0.2.

derivative

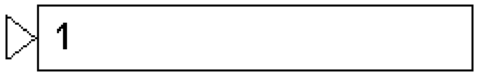


Block Category: Toolbar

The `derivative` block appears on the toolbar (🔍) when you install VisSim 3.0b+. It calculates the change in function value with respect to time.

The `derivative` block has two inputs: step size and signal. The step size indicates the sampling rate of the derivative. It must be greater than zero. When the step size is large with respect to the function, the signal can become unstable. When you integrate the output of the `derivative` block, you will see degradation in the signal.

display

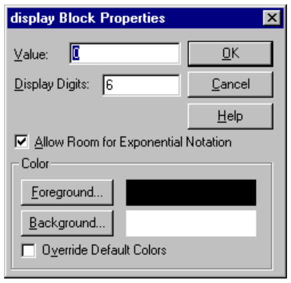


display = x_1

Block Category: Signal Consumer

The `display` block displays the current value of the input signal in any number of significant digits. You can select a color for the displayed value, as well as a background color for the block.

The `display` block automatically expands to display vector and matrix elements in individual cells.



Value: 0

Display Digits: 6

[✓] Allow Room for Exponential Notation

Color

[ ] Override Default Colors

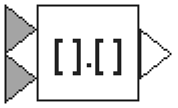
Value: Controls the current value in the display. The default is 1.

Display Digits: Indicates the number of displayed significant digits. The value you enter overrides the setting of the High Precision Display parameter under Preferences in the dialog box for the Edit > Preferences command. The default is 6.

Allow Room For Exponential Notation: Expands the `display` block so there is room for exponential notation. If you wish to have a very small `display` block, perhaps for use in display mode, you should turn off this option.

Color: Applies background and foreground color to the `display` block. Click on the Foreground and Background buttons to select a color. The selected colors are displayed to the right of the buttons. To override the background color selected using the View > Colors command, activate Override Default Color.

dotProduct



$$y = \sum_{K=1}^{N} x1_K \times x2_K$$

Block Category: Matrix Operations

The dotProduct block produces a single value summation of an element-by-element multiply. The dotProduct block accepts two vector inputs and produces a scalar output. If the input vectors have an uneven number of elements, an error occurs.

Multiplying scalars and matrices

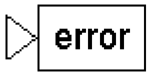
To multiply two or more scalars, use the * block, as described on page 142.

To multiply two matrices, use the multiply block, as described on page 220.

embed

The embed block lets you embed a multi-level block diagram in the current block diagram. For more information, see page 127.

error



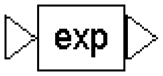
$$\text{error condition} = x_1$$

Block Category: Signal Consumer

The error block flags an error in a simulation. When the input signal becomes non-zero, the error block and all compound blocks which contain it are highlighted in red and the simulation is stopped.

You can reset the error condition by clicking the right mouse button on the error block.

exp



$y = e^x$

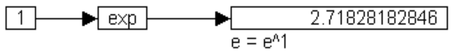
Block Category: Transcendental

The `exp` block performs the inverse operation of the `ln` block and raises the input as a power of *e*. The irrational number *e* is the base of natural logarithms and is approximately equal to 2.7182828.

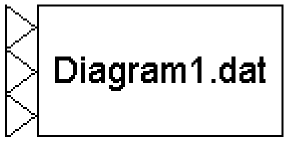
Examples

1. Computation of the value of *e*

The value of *e* can be obtained by providing an input value of 1 to an `exp` block as:



export



data file $\text{column}_n = x_n$

Block Category: Signal Consumer

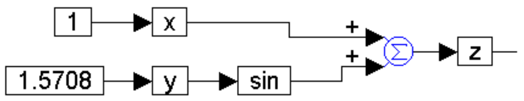
The `export` block writes signals to a file in .DAT, .M, .MAT, or .WAV file format. The file can subsequently be used as input to VisSim or to a variety of other programs, such as MatLab and Microsoft Excel. For more information, see page 111.

expression

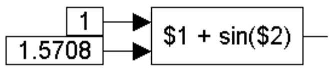
The `expression` block allows you to enter a C expression that VisSim parses and acts upon. With expressions, you can significantly reduce the number of blocks in your diagrams. For example, consider the simple equation:

$x + \sin(y) = z$

Without the `expression` block, the block diagram representation of this equation is:



Instead of using the `variable`, `sin`, and `summingJunction` blocks, you can create a single C expression that performs the same function:



The elements $1 and $2 are VisSim-specific notation that reference the inputs.

What you can do with expression blocks:

- Speed up simulation time

The more blocks in a diagram, the longer it takes to simulate to the diagram. Consequently, as you replace series of blocks with `expression` blocks, the simulation time decreases.

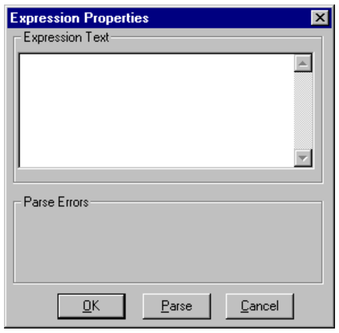
- Reduce development time

Instead of inserting groups of blocks and wiring them together, you can insert a single `expression` block that performs the same function.

- Simplify troubleshooting

You can request that VisSim check the logic of the expression before you simulate the diagram. You simply press a key and VisSim does the rest.

Writing an expression: The Expression Properties dialog box lets you set up your expression.



Expression Text: Indicates a C expression. A C expression consists of one or more operands and zeros or more operators linked together to compute a value. You enter expressions according to the syntax rules for the C language. If you're unfamiliar with the language, refer to *C: A Software Engineering Approach*, (Springer-Verlag, 1990).

The following VisSim-specific rules apply to entering C expressions:

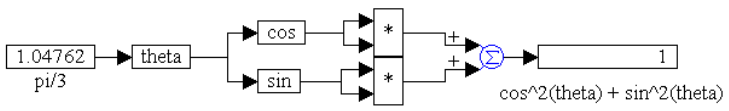
- Inputs are referenced using the notation $*n*, where *n* represents an connector number. For example, $1 is input 1 (the top input connector), $2 is input 2 (the second from the top input connector), and so on.
- Only one output value is allowed.

Parse Errors: Lists the errors that occur when VisSim parses the C expression. This is a read-only box.

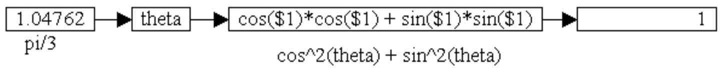
Examples

1. Computation of cos^2(θ) + sin^2(θ) = 1

If θ is chosen to be π/3, the above expression can be realized as:



The same equation can be realized using an `expression` block as:


1.04762
pi/3
theta
cos($1)*cos($1) + sin($1)*sin($1)
1
cos^2(theta) + sin^2(theta)

Here, the expression cos^2(θ) + sin^2(θ) is entered directly into the `expression` block as cos($1) * cos($1) + sin($1) * sin($1), where $1 corresponds to the only input on the `expression` block. When the simulation runs, VisSim substitutes $1 in the expression with the top input connected to it and then evaluates the expression.

From the results obtained, both methods yield the correct answer.

fft

Block Category: Matrix Operation

The `fft` block converts data from time domain to frequency domain.

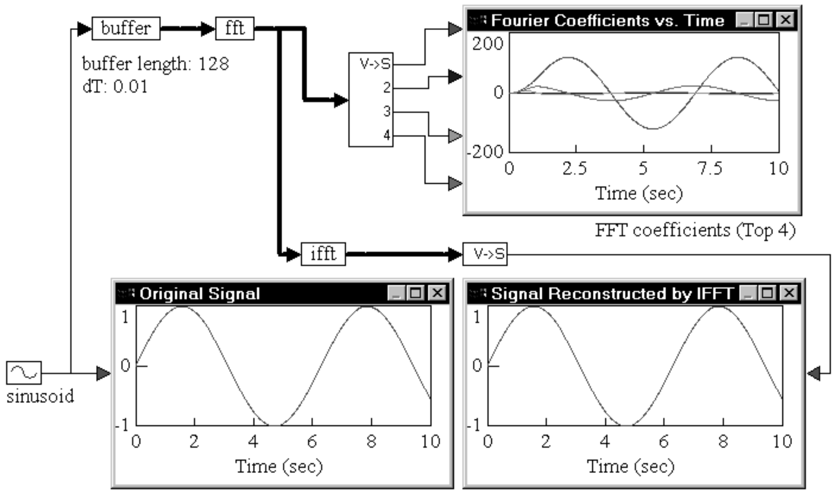
The fft block computes an *n*-sample FFT at every simulation time step, where *n* is the length of the input vector.

If the input to the fft block is not an integral power of 2, automatic zero padding is performed to make the input vector size an integral power of 2. This is a standard procedure in FFT computation. The output of the fft block is Fourier coefficients. Individual coefficients can be accessed using a vecToScalar block.

Examples

1. Computation of FFT and inverse FFT

Consider a simple example, where a sinusoidal signal is converted to frequency domain via FFT, and then reconstructed using inverse FFT.



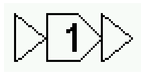
A sinusoid block generates a sinusoid signal with a frequency of 1 rad/sec. The signal is passed through a buffer block of length 128 samples and a dT of 0.01. The output of the buffer block is connected to an fft block, which computes a 128-sample FFT of the original sinusoid at a sampling rate of 0.01.

The output of the fft block is Fourier coefficients. The individual coefficients are accessed using a vecToScalar block. The first four coefficients are plotted to show their variation with time.

Signal reconstruction is performed by feeding the output of the fft block to an ifft block to compute the inverse FFT. The output of the ifft block is a vector of length 128 samples. The contents of this vector are just 128 sinusoid reconstructions, with each sinusoid trailing the preceding sinusoid by an amount equal to the sampling rate.

The first element in the `ifft` output vector does not have any delay because zero time has elapsed between the FFT and inverse FFT phases. In most real-world situations, however, there is a small, non-zero delay between the input signal and its reconstruction that is introduced by the processor performing the numerical computations of FFT and inverse FFT algorithms.

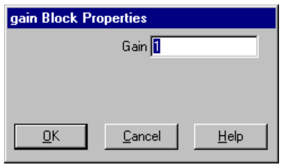
gain



$y = x \cdot \text{gain}$

Block Category: Arithmetic

The `gain` block multiplies the input signal, by the gain amount. The input can be a scalar, vector, or matrix.


gain Block Properties

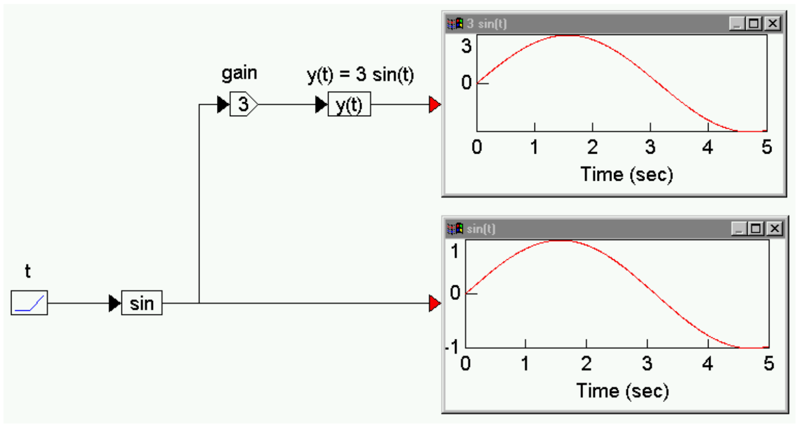
Gain 1

Gain: Indicates the constant multiplier of the input signal. The default is 1.

Examples

1. Gain of a scalar

Consider the equation $y(t) = 3\ \sin(t)$, which can be realized as:



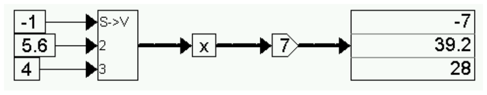
A `ramp` block is used to access simulation time *t*, a `sin` block generates sin(*t*), a `gain` block amplifies sin(*t*) to 3 sin(*t*). Both sin(*t*) and y(*t*) are shown in `plot` blocks for comparison.

2. Gain of a vector

Consider the equation:

$\mathbf{z} = 7\,\mathbf{x}$

where $\mathbf{x}$ = [-1 5.6 4]. This equation can be realized as:



The `gain` block performs an element-by-element gain operation on the incoming vector.

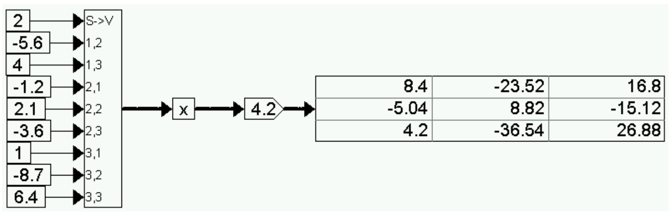
3. Gain of a matrix

Consider the equation:

$\mathbf{Z} = 4.2\,\mathbf{X}$

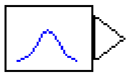
where $\mathbf{X} = \begin{bmatrix} 2 & -5.6 & 4 \\ -1.2 & 2.1 & -3.6 \\ 1 & -8.7 & 6.4 \end{bmatrix}$

This equation can be realized as:



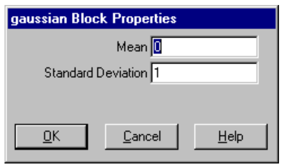
The `gain` block performs an element-by-element gain operation on the incoming matrix.

gaussian



Block Category: Random Generator

The `gaussian` block creates a normally distributed, random noise signal. You specify a random seed value under the Preferences tab in the dialog box for the Simulate > Simulation Properties command.


gaussian Block Properties

Mean: 0

Standard Deviation: 1

Mean: Indicates the center of the distribution. The default value is 0.

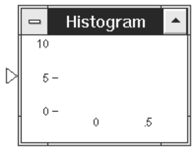
Standard Deviation: Indicates the distance from the mean, which covers one standard deviation. The default value is 1.

globalConstraint

Block Category: Optimization

The `globalConstraint` block provides side constraint information when writing your own global optimizer. For more information, see Chapter 8, “Performing Global Optimization.”

histogram



Block Category: Signal Consumer

The `histogram` block shows how data are distributed over the course of a simulation. For more information, see page 73.

ifft

Block Category: Matrix Operation

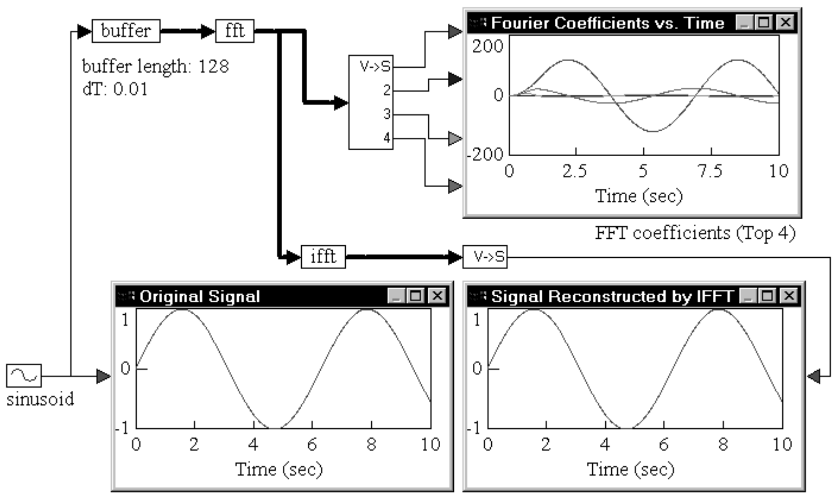
The `ifft` block converts data from frequency domain to time domain. The `ifft` block computes an *n*-sample inverse FFT at every simulation time step, where *n* is the length of the input vector.

If the input to the `ifft` block is not an integral power of 2, automatic zero padding is performed to make the input vector size an integral power of 2. This is a standard procedure in inverse FFT computation. The output of the `ifft` block is Fourier coefficients. Individual coefficients can be accessed using a `vecToScalar` block.

Examples

1. Computation of FFT and inverse FFT

Consider a simple example, where a sinusoidal signal is converted to frequency domain via FFT, and then reconstructed using inverse FFT.



A `sinusoid` block generates a sinusoid signal with a frequency of 1 rad/sec. The signal is passed through a `buffer` block of length 128 samples and a sampling rate of 0.01. The output of the `buffer` block is connected to an `fft` block, which computes a 128-sample FFT of the original sinusoid at a sampling rate of 0.01.

The output of the `fft` block is Fourier coefficients. The individual coefficients are accessed using a `vecToScalar` block. The first four coefficients are plotted to show their variation with time.

Signal reconstruction is performed by feeding the output of the `fft` block to an `ifft` block to compute the inverse FFT. The output of the `ifft` block is a vector of length 128 samples. The contents of this vector are just 128 sinusoid reconstructions, with each sinusoid trailing the preceding sinusoid by an amount equal to the sampling rate.

The first element in the `ifft` output vector does not have any delay because zero time has elapsed between the FFT and inverse FFT phases. In most real-world situations, however, there is a small, non-zero delay between the input signal and its reconstruction that is introduced by the processor performing the numerical computations of FFT and inverse FFT algorithms.

import

Diagram1.dat

y_n = data file column_n

Block Category: Signal Producer

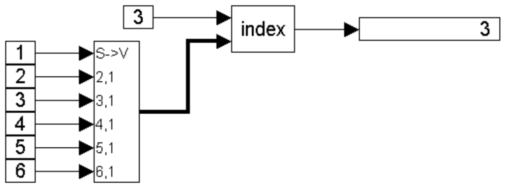
The `import` block imports data points from a .DAT, .M, .MAT, or .WAV file and translates them into output signals. The data can be either fixed interval or asynchronous. For more information, see page 109.

index

Block Category: Annotation

Like the `case` block, the `index` block provides an unlimited number of execution paths based on the value of a single input. With the `index` block, however, all the execution paths are contained in a vector or matrix. The top input to the `index` block points to the matrix or vector element to be output. The bottom input to the `index` block is the matrix or vector from which the output is selected.

For example, a 6 × 1 vector fed into the `index` block yields six possible execution paths:



The `index` block outputs 1, 2, 3, 4, 5, or 6 depending on whether the index value is 1, 2, 3, 4, 5, or 6, respectively. In this example, the index value is 3, causing a 3 to be output.

It is important to know how an index value references matrix elements. Index values map to matrix elements in sequential order, starting with the element in *column*1-*row*1, through *column*1-*rowN*; then *column*2-*row*1 through *column*2-*rowN*; and so on. For example, in the following 2 × 3 matrix, an index value of 3 yields 5:

INDEX VALUE 1: 1	INDEX VALUE 4: 2
INDEX VALUE 2: 3	INDEX VALUE 5: 4
INDEX VALUE 3: 5	INDEX VALUE 6: 6

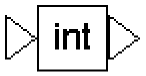
In a 3 × 2 matrix, an index value of 3 yields 2:

1	2	3
4	5	6

Index value: The following rules apply to the index value:

- Index values that are non-integers are truncated. For example, 0.999 is truncated to 0.
- If the index value targets an unconnected matrix or vector element, the `index` block outputs a 0.
- If the index value targets an out-of-range matrix or vector element, the `index` block outputs spurious results. For example, if the index value is 5 for a four-element matrix, the output, might look something like this: 1.06983e-306.

int



y = integer part *x*

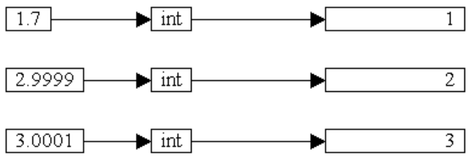
Block Category: Nonlinear

The `int` block accepts a scalar input and outputs only the integer portion of the input. The `int` block does not perform numerical round-off operations. Thus, an input of 2.9999 yields 2. Inputs can be scalar constants or scalar variables.

Examples

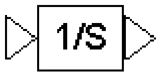
1. Integer portions of scalar inputs

Consider three scalar inputs 1.7, 2.9999, and 3.0001. These inputs are applied to the `int` blocks, as shown below:



The `int` blocks isolate and output the integer portion of the scalar inputs.

integrator (1/S)

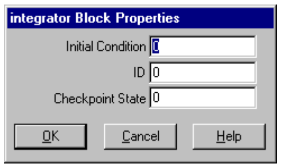


$$y = \int_{t_{\text{start}}}^{t_{\text{end}}} x\, dt$$

Block Category: Integration

The `integrator` block performs numerical integration on the input signal using the integration algorithm (Euler, trapezoidal, Runge Kutta 2d and 4th orders, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff)) established with the Simulate > Simulation Properties command.

The `integrator` block is one of the most fundamental and powerful blocks in VisSim. This block, together with the `limitedIntegrator` and `resetIntegrator` blocks, offer the power to solve an unlimited number of simultaneous linear and nonlinear ordinary differential equations.


integrator Block Properties

Initial Condition: 0

ID: 0

Checkpoint State: 0

Initial Condition: Indicates the initial value of the integrator. The default value is 0.

ID: Represents an identification number for the block. It keeps track of the state number that VisSim assigns to the integrator. The number of states in any block diagram equals the number of integrators. The default value is 0.

Checkpoint State: Contains the value of the integrator state at the checkpoint. If you have not checkpointed your simulation using the Simulate > Simulation Properties command, the value is 0.

Examples

1. Solving a first order ODE

Consider the simple first order linear differential equation:

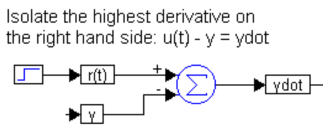
$$\dot{y} + y = r(t)$$

where $r(t)$ is an external input. In this case, assume that the external input to the system is a step function. In VisSim, such equations are best solved by numerical integration.

The first step is to isolate the highest derivative term on one side. To understand the procedure better, it is easier to think of isolating the highest derivative term on the right-hand side as:

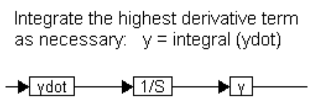
$$r(t) - y = \dot{y}$$

This equation can be constructed as:

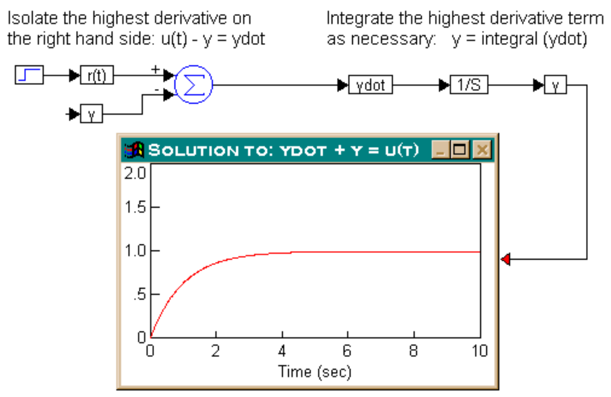

Isolate the highest derivative on
the right hand side: u(t) - y = ydot
r(t)
y
+
-
ydot

Here, three `variable` blocks are used for *r*(*t*), *y*, and *ydot*.

The second step is to integrate the highest derivative term a sufficient number of times to obtain the solution. Since the highest derivative is of first order, *ydot* must be integrated once to obtain *y*. This can be realized as:


Integrate the highest derivative term
as necessary: y = integral (ydot)
ydot
1/S
y

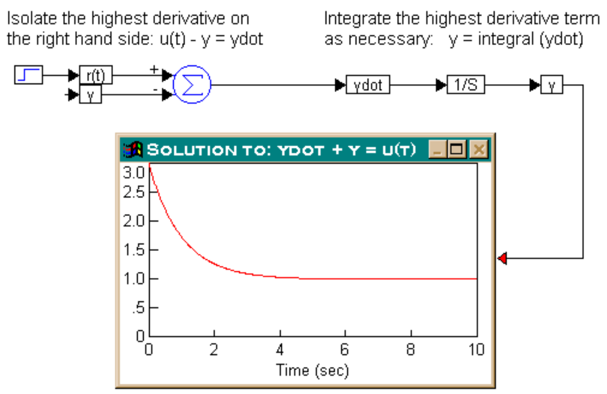
The overall simulation is shown below.


Isolate the highest derivative on
the right hand side: u(t) - y = ydot
Integrate the highest derivative term
as necessary: y = integral (ydot)
r(t)
y
+
-
ydot
1/S
y
SOLUTION TO: YDOT + Y = U(T)
2.0
1.5
1.0
.5
0
0
2
4
6
8
10
Time (sec)

The result shown in the `plot` block indicates the solution of the differential equation subjected to a step external forcing function.

2. Setting the integrator initial condition internally

Consider the same problem above with the assumption that $y(0) = 3$. In this case, in addition to the external input $r(t)$, the system response also depends on $y(0)$. This initial condition can be set directly in the `integrator` block. The result for this case is:


Isolate the highest derivative on the right hand side: u(t) - y = ydot
Integrate the highest derivative term as necessary: y = integral (ydot)
r(t)
y
+
-
ydot
1/S
y
Solution to: ydot + y = u(t)
3.0
2.5
2.0
1.5
1.0
.5
0
0
2
4
6
8
10
Time (sec)

The `plot` block shows that the response $y(t)$ begins at $y(0) = 3$ and settles down to 1 for $t > 4.5$, as expected.

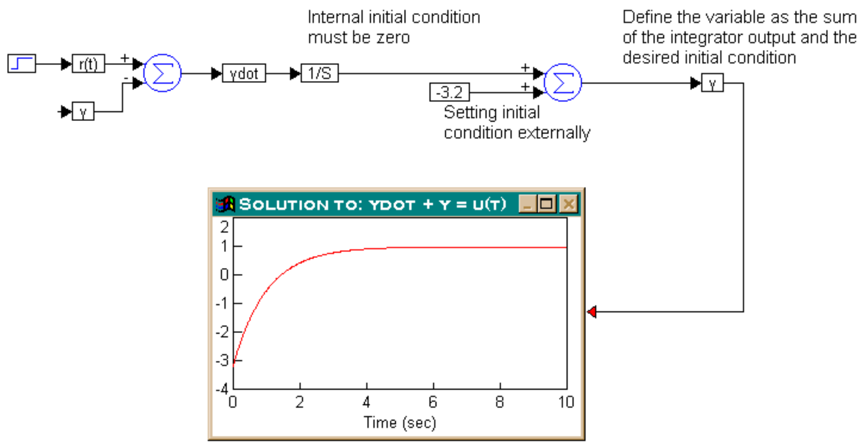
It is important to note that the initial condition on any state (or variable) must be set on the `integrator` block that is generating that state. (This concept becomes clearer in Example 4.)

3. Setting the integrator initial condition externally

Consider once again the following ordinary differential equation:

$$\dot{y} + y = r(t)$$

Let $r(t)$ be a step function and assume that $y(0) = -3.2$. The initial condition can be set externally, as shown on the next page.


Internal initial condition must be zero
Define the variable as the sum of the integrator output and the desired initial condition
r(t)
ydot
1/S
y
-3.2
Setting initial condition externally
SOLUTION TO: YDOT + Y = U(T)
Time (sec)

In this configuration, make sure that the internal initial condition of the `integrator` is set to zero. By default, all `integrators` have zero initial condition.

The results indicate that the solution of the ordinary differential equation, subject to the external input and the initial conditions, is computed correctly.

4. Second order nonlinear ODE with external initial conditions

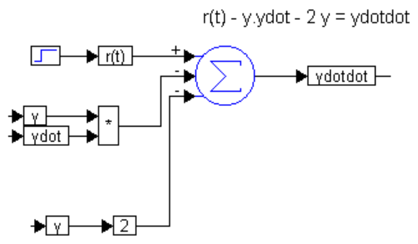
Consider a second order nonlinear system given by:

$$\ddot{y} + y\dot{y} + 2y = r(t)$$

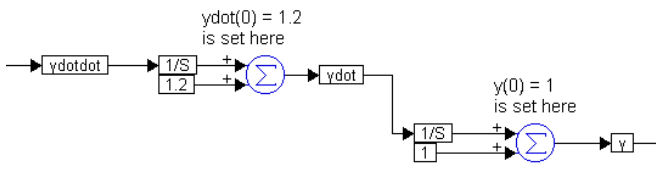
Furthermore, assume that *r*(*t*) is a unit step function and that the initial conditions are given by:

$$y(0) = 1.0 \text{ and } \dot{y}(0) = 1.2$$

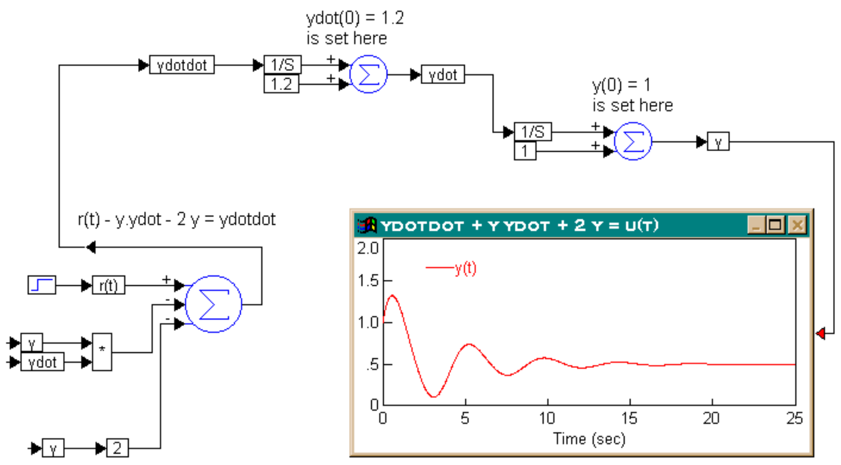
The first step is to isolate the highest derivative term on the right-hand side as $r(t) - y\dot{y} - 2y = \ddot{y}$. This segment can be coded in VisSim as shown below:



The second step is to integrate *ydotdot* twice: once to generate *ydot*, and once more to generate *y*. As can be imagined, it is crucial to maintain consistent variable names throughout. Furthermore, the initial conditions must be added using the same procedure described in Example 3. This segment can be realized as:



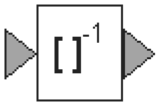
The complete solution for this problem is given by:



The solution of the equation, *y*(*t*) is shown in the `plot` block.

This example illustrates the real power of numerical integration using VisSim. If you want to use the results of a computational segment in a given VisSim diagram as initial conditions for one or more integrators, replace the `const` blocks with appropriate `variable` blocks when setting the external initial conditions.

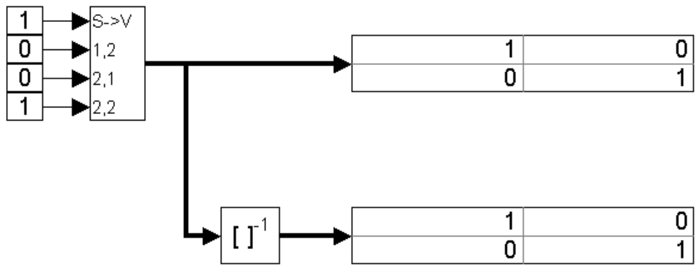
invert



$$[\mathbf{A}]^{-1} = \frac{adj(\mathbf{A})}{det(\mathbf{A})}$$

Block Category: Matrix Operation

The `invert` block inverts a square matrix using singular value decomposition. The `invert` block accepts one vector input and produces one vector output.

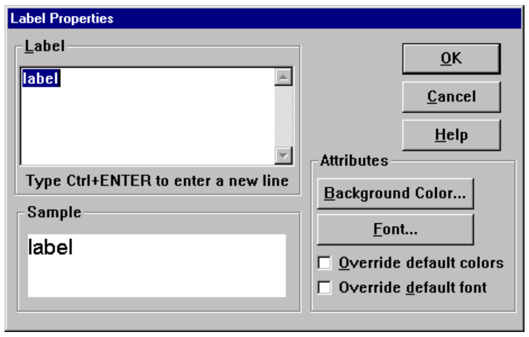


label

Block Category: Annotation

The `label` block lets you insert floating labels in a block diagram. You can choose the text attributes for the label, as well as a colored background.

The `label` block is particularly useful for tagging signals.



Label: Specifies a label. To continue a label to a new line, hold down the CTRL key while you simultaneously press the ENTER key.

Attributes: Assigns a background color and text attributes to the label. Click on the Background Color button to select a background color for the label. Click on the Fonts button to select a font, font style, point size, color, and special effects for the text. A sample of the text is displayed in the Sample box.

To override the selections in the View > Colors and View > Fonts dialog boxes, activate Override Default Colors and Override Default Font, respectively.

light

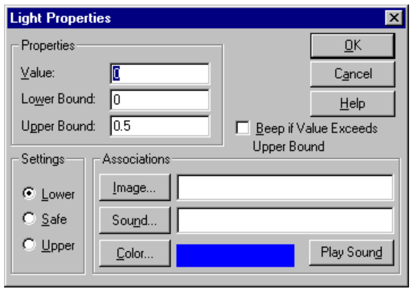
$$y = \begin{cases} \text{red} & \text{if } x_1 > \text{ub} \\ \text{green} & \text{if } \text{lb} \le x_1 \le \text{ub} \\ \text{blue} & \text{if } x_1 < \text{lb} \end{cases}$$

Block category: Signal Consumer

The `light` block is a tri-state alarm that glows a color, displays a bitmap image, or plays sound when supplied with a signal. By default, the `light` block glows red when the signal is greater than the upper bound; blue when the signal is less than the lower bound; and green when the signal is less than or equal to the upper bound and greater than or equal to the lower bound.

Associating an action with a state: To associate an action — for example, the display of a bitmap image file — with a given state, select the state from the Settings box; then click on the Bitmap button and choose the .BMP file to be associated with the state.

Setting up a light block: The `light` block's Properties dialog box lets you control its audio and visual alarms.



Light Properties

Properties
Value: 0
Lower Bound: 0
Upper Bound: 0.5

☐ Beep if Value Exceeds Upper Bound

Settings
(●) Lower
( ) Safe
( ) Upper

Associations
Image...
Sound...
Color... 🟦

Properties: Establishes the lower and upper bounds for the signal, as well as the initial setting of the signal.

- **Value:** Indicates the initial setting for the signal. The default is 0.
- **Lower Bound:** Indicates the lower bound for the signal. When the signal is less than the specified lower bound, the `light` block performs the action (emits a color, sound, or image) associated with the Lower setting. The default is 0.
- **Upper Bound:** Indicates the upper bound for the signal. When the signal is greater than the specified upper bound, the `light` block performs the action (emits a color, sound, or image) associated with the Upper setting. The default is 0.5.

Settings: Indicates the setting to which color, sound, or an image is to be applied.

- **Lower:** The signal is less than the specified lower bound.
- **Safe:** The signal is less than or equal to the specified upper bound and greater than or equal to the specified lower bound.
- **Upper:** The signal is greater than the specified upper bound.

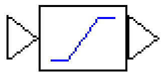
Associations: Indicates whether an image, sound, or color is to be applied to the specified setting.

- **Image:** Opens the File Select dialog box in which to choose a .BMP file to associate with the selected setting.
- **Sound:** Opens the File Select dialog box in which to choose a .WAV file to associate with the selected setting.
- **Color:** Opens the Color dialog box in which to choose a color to associate with the selected setting.

Play Sound: Plays the sound for the selected setting.

Beep If Value Exceeds Upper Bound: Forces the `light` block to beep when the signal exceeds the specified upper bound.

limit

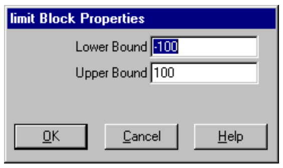


$$y = \begin{cases} x_1 & \text{if } \mathrm{lb} \le x_1 \le \mathrm{ub} \\ \mathrm{lb} & \text{if } x_1 < \mathrm{lb} \\ \mathrm{ub} & \text{if } x_1 > \mathrm{ub} \end{cases}$$

Block Category: Nonlinear

The `limit` block limits the output signal to a specified upper and lower bound. The `limit` block accepts a scalar input. If the input is less than the lower bound, the `limit` block limits the output to the lower bound. Similarly, if the input is greater than the upper bound, the `limit` block limits the output to the upper bound. If the input falls within the specified bounds, the input is transferred to the output unchanged.

The `limit` block is particularly useful for simulating variables or processes that reach saturation.


limit Block Properties

Lower Bound -100

Upper Bound 100

Lower Bound: Indicates the lowest value that the output signal can attain. The default is -100.

Upper Bound: Indicates the highest value that the output signal can attain. The default is 100.

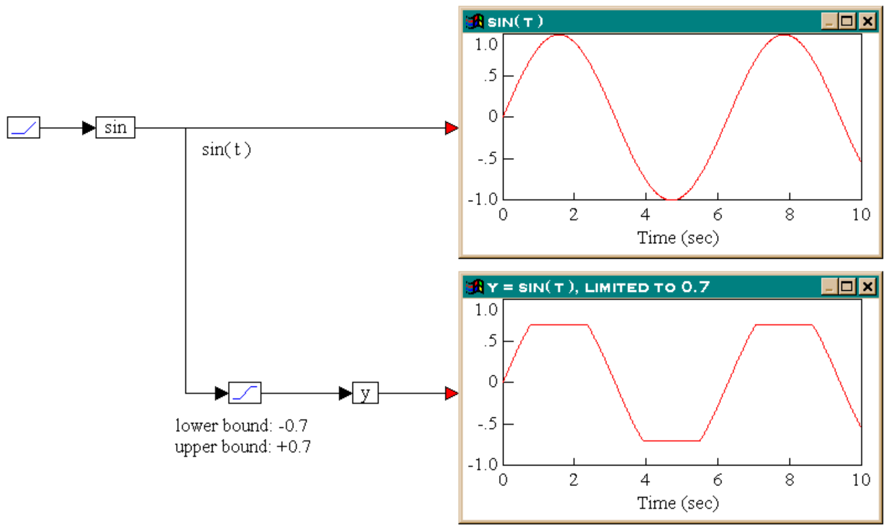
Examples

1. Simulation of saturation

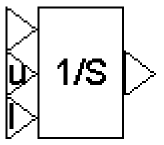
Consider a variable y such that:

$$y = \sin(t)$$

Furthermore, assume that y reaches saturation at +0.7 and -0.7. This equation can be realized as shown on the next page.



From the results in the two `plot` blocks, the output of the `limit` block is identical to the input, when the input is within the bounds (-0.7 to +0.7). When the input is out of these bounds, the output is limited to the upper or lower bound values.



limitedIntegrator (1/S)

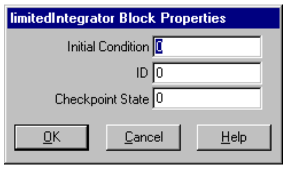
$$y = \begin{cases} \int_{t_{start}}^{t_{end}} x_1 dt & \text{if } x_2 \geq \int_{t_{start}}^{t_{end}} x_1 dt \geq x_3 \\ x_2 & \text{if } \int_{t_{start}}^{t_{end}} x_1\, dt > x_2 \\ x_3 & \text{if } \int_{t_{start}}^{t_{end}} x_1 dt < x_3 \end{cases}$$

Block Category: Integration

The `limitedIntegrator` block integrates the input value and limits the internal state to specified upper and lower limits. If the integral state reaches its limit, it backs off the limit as soon as the derivative changes sign. You set the integration algorithm with the Simulate > Simulation Properties command. Available algorithms are Euler, trapezoidal, Runge Kutta 2nd and 4th orders, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff).

The inputs to the block are x_1, the derivative; x_2 (U), the upper limit; and x_3 (L), the lower limit.

The `limitedIntegrator` block is used in the prevention of wind-up in PI and PID controllers in control applications. It is also used in kinematics, electrical circuits, process control, and fluid dynamics.


limitedIntegrator Block Properties

Initial Condition	0
ID	0
Checkpoint State	0

Initial Condition: Indicates the initial value of the integrator. The default is 0.

ID: Represents an identification number for the block, which holds the state number that VisSim assigns to the integrator. The number of states in any block diagram equals the number of integrators. The default is 0.

Checkpoint State: Contains the value of the integrator state at the checkpoint. If you have not checkpointed your simulation using the Simulate > Simulation Properties command, the value is 0.

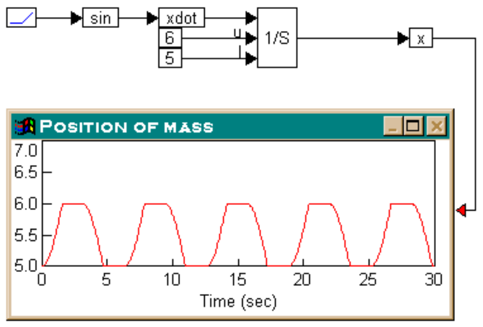
Examples

1. Integration with constant limits

Consider a system whose dynamics are given by the differential equation:

$$\dot{x} = \sin(t)$$

Furthermore, assume that x must lie in the limits $5 \leq x \leq 6$ and that $x(0) = 5$. This system can be realized as shown on the next page.



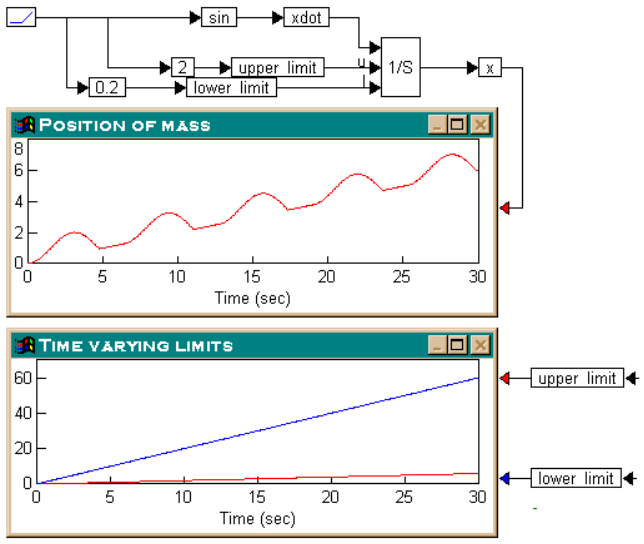
During simulation, the `limitedIntegrator` block limits the output to be within the upper and lower limits, namely 6 and 5, respectively.

2. Integration with time-varying limits

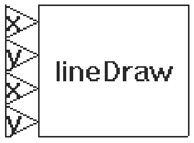
Consider a system whose dynamics are given by the differential equation:

$$\dot{x} = \sin(t)$$

Furthermore, assume that x must lie in the limits $0.2t \le x \le 2t$ and that $x(0) = 0$. This system can be realized as:



A `ramp` block is used to access simulation time, t; simulation time is then used to feed the `sin` block, and two `gain` blocks, set to 2 and 0.2, to generate the time-varying upper and lower limits. During simulation, the time-varying limits and the output of the `limitedIntegrator` block are displayed in the `plot` blocks.



lineDraw

Block Category: Animation

The `lineDraw` block lets you animate a line during simulation. You define the line by specifying two sets of *x,y* coordinate endpoints. You can also set the color, thickness, and style of the line. For more information, see page 81.

log10

$y = \log_{10}x$

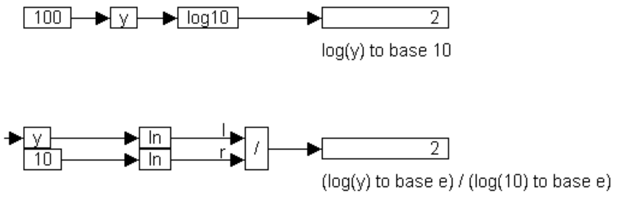
Block Category: Transcendental

The `log10` block generates the log base 10 of the input signal. The logarithm of 0 to any base is undefined. The logarithm of any number, when the base is the same number, is 1.

Examples

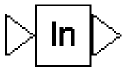
1. Computation of $\log_{10} y = \log_e y / \log_e 10$

With *y* chosen to be 100, and *e* as the base of the natural logarithm, this equation can be realized as:



From the results obtained, $\log_{10} y = \log_e y / \log_e 10$ where *e* is the base of the natural logarithm. It can further be proved that $\log_a y = \log_b y / \log_b a$, where *a*, *b*, and *y* are any positive non-zero numbers.

ln



$y = \log_e x$

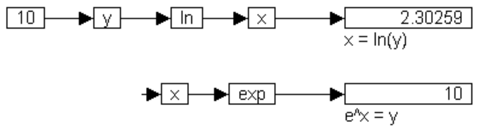
Block Category: Transcendental

The `ln` block generates the natural (Naperian) log of the input signal.

Examples

1. If ln(y) = x, then e^x = y

With *y* chosen to be 10, and *e* as the base of the natural logarithm, this equation can be realized as:



The `exp` block raises its input as a power of *e*, the base of natural logarithm. The quantity *e* is an irrational number, which is approximately equal to 2.718281828. From the results obtained, if ln(*y*) = *x*, then $e^x = y$, where *e* is the base of the natural logarithm.

map

$$y_1(y_2 \ldots y_n) = \text{table lookup}\, x \quad (1-\text{D})$$
$$y = \text{table lookup}(x_1, x_2) \quad (2-\text{D})$$
$$y = \text{table lookup}(x_1, x_2, x_3) \quad (3-\text{D})$$

Block Category: Nonlinear

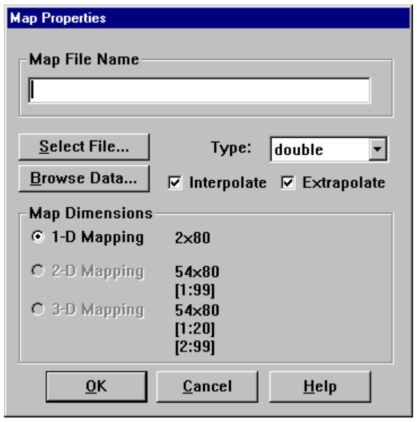
The `map` block performs piecewise linear interpolated 1- 2-, and 3-dimensional table look-ups. This means that you can encapsulate nonlinear behaviors through direct measurements. You can, for example, use laboratory data or a manufacturer's component performance data directly in a simulation.

The `map` block searches the input vector, starting at the last look-up input value to avoid table search overhead.

The `map` block uses a multi-column ASCII data file to map input signals to a desired output domain. Numbers can be separated by commas, spaces, tabs, vertical bars, colons, semicolons or slashes. One-dimensional maps have one independent variable, but can have from one to 16 dependent variable outputs. Two-dimensional maps have two independent variables and one dependent variable output. Three-dimensional maps have three independent variables and one dependent variable output.

Dependent variables are linearly interpolated for independent variable values between map points, and linearly extrapolated for values beyond the bounds of the table using the last two points in the table. This feature can be used for static function approximation with measured data or for device calibration, such as thermocouple-voltage-to-temperature conversion.

Use the `export` block to create map files in VisSim.


Map Properties

Map File Name:

Type: double

Interpolate [✓] Extrapolate [✓]

Map Dimensions

[✓ 1-D Mapping] 2x80

2-D Mapping 54x80 [1:99]

3-D Mapping 54x80 [1:20] [2:99]

Map File Name: Indicates the name of the map file. You can type in a file name directly into this box or select one using the Select File button.

To open the specified file with the default text editor, click on the Browse Data button.

Map Dimensions: Controls the dimensionality of the map file.

- **1-D Mapping:** Indicates 1-D mapping capability. VisSim includes the number of columns and rows in the selected map file, and the first and last numbers in the first column of the selected map file to the right of this parameter.

  In 1-D mapping, the first column is an independent variable range. The numbers in the independent variable column must be either in increasing order or decreasing order, but not both. Each additional data column you supply in the map file yields an additional dependent variable. Use the Edit > Add Connector command to add an output connector tab for each dependent variable column in the `map` block. The topmost output connector tab corresponds to the leftmost dependent variable column in the table, the second from the top corresponds to the second from the left, and so on.

A 1-D matrix is limited to 8000 rows.

The numbers to the right of the 1-D Mapping parameter refer to the dimensionality and range of the map vector. For example, 10x1[1:100] represents a 1-D table with 10 elements ranging from 1 to 100.

Lines that begin with a prefix of ";" are treated as comments.

- **2-D Mapping:** Provides simultaneous mapping for two independent variables. The format of a 2-D map file is as follows: the first row contains the domain points for the first independent variable (the topmost connector tab on the `map` block), the first column (excluding the column member in row 1) represents the second independent variable, and the (1,1) position must be left blank. Like 1-D mapping, the independent variable values must be either monotonically increasing or decreasing.

  A 2-D matrix is limited to 90 rows by 90 columns (or, a maximum of 89 * 89 data points).

  Lines that begin with a prefix of ";" are treated as comments.

  An example of a 2-D map file is shown below.

	10	***11***	***20***	***25***
-5	-5	-2	1	20
2	2	5	7	10
3	3	7	8	5
4	4	9	10	2
5	5	11	15	-5

  In the above matrix, the first row represents the domain points of the first independent variable, and the first column represents the domain points of the second independent variable. The entries represent the dependent variable values at the corresponding values of independent variables 1 and 2. For example, for $x_1 = 10$, $x_2 = 2$, the output is 2; for $x_1 = 10.5$, $x_2 = 2.5$, the output is 4.25.

  The numbers to the right of the 2-D Mapping parameter refer to the dimensionality and range of the map vector. For example, 10x50[10:20, -10:10] represents a 2-D table with 10 columns and 50 rows, where the minimum column is 10, the maximum column is 20, the minimum row is -10, and the maximum row is 10.

- **3-D Mapping:** Provides simultaneous mapping of three independent variables. The format of the first seven lines is as follows:

Line	Format
Line 1	Starts with #3D
Line 2	Indicates the size of dimension 1
Line 3	Indicates the interpolation points of dimension 1
Line 4	Indicates the size of dimension 2
Line 5	Indicates the interpolation points of dimension 2
Line 6	Indicates the size of dimension 3
Line 7	Indicates the interpolation of dimension 3

Lines 8 through Line *n* are elements of dimension 3 matrices of (dimension 1 columns × dimension 2 rows). Lines that begin with a prefix of "--", ";", or "//" are treated as comments.

Type: Indicates the type of data read in from the map file.

Interpolate: Allows dependent variables to be linearly interpolated for independent variable values between data points. This feature can be used for static function approximation with measured data or for device calibration, such as thermocouple-voltage-to-temperature conversion.

Extrapolate: Allows dependent variables to be linearly extrapolated for values beyond the bounds of the table using the last two data points in the table. This feature can be used for static function approximation with measured data or for device calibration, such as thermocouple-voltage-to-temperature conversion.

Examples

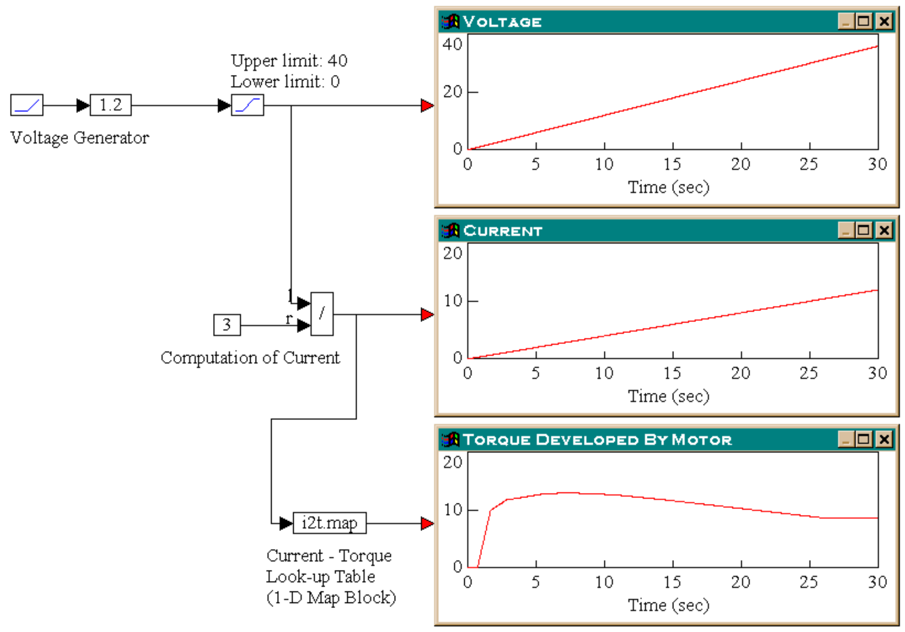
1. 1-D look-up table

Consider a hypothetical electrical motor that accepts DC input voltage in the range of 0 to 40 V. Furthermore, assume that the current drawn by the motor is equivalent to that of an ideal 3Ω resistor. The motor manufacturer has specified the following current-torque curve for the motor:

Current (A)	Torque (N-M)
0.	0.
.3	0.
.68	10.

Current (A)	Torque (N-M)
1.15	11.8
2.16	12.77
2.86	13.04
3.7	12.86
4.36	12.66
5.74	11.84
6.73	11.18
10.5	8.62
11.5	8.62

Assuming that the voltage is applied at the rate of 1.2 *t*, where *t* is time in sec. This system can be realized as shown below:



To generate the voltage, wire a `ramp` block, used to generate simulation time *t*, to a `gain` block set to 1.2. The output of the `gain` block passes through a `limit` block with its lower and upper limits set to 0 and 40, respectively. The output of the `limit` block is the voltage applied to the motor and is monitored in the upper plot.

To compute the current, divide the output of the `limit` block is by a constant value 3. The current is monitored in the middle plot.

A `map` block points to the data file I2T.MAP, with two columns of data containing the current-torque curve for the motor. (The sample data used in this example is shown in the table above.)

The `map` block monitors the input value and compares it with the data in the first column. For example, if the input value is 3, the `map` block recognizes that the input is between the two points 2.86 and 3.7 in the input column. The `map` block performs a linear interpolation between the corresponding values in the second column, namely 13.04 and 12.86. Consequently, for an input of 3, the output of the `map` block is:

13.04 + (3 - 2.86) * ((12.86 - 13.04) / (3.7 - 2.86))

which is equal to 13.01.

2. 2-D look-up table

Using the same hypothetical motor described above, make the following assumptions:

- The torque developed by the motor is a function of the current, as well as the operating temperature of the motor.
- The current-temperature-torque has the following profile:

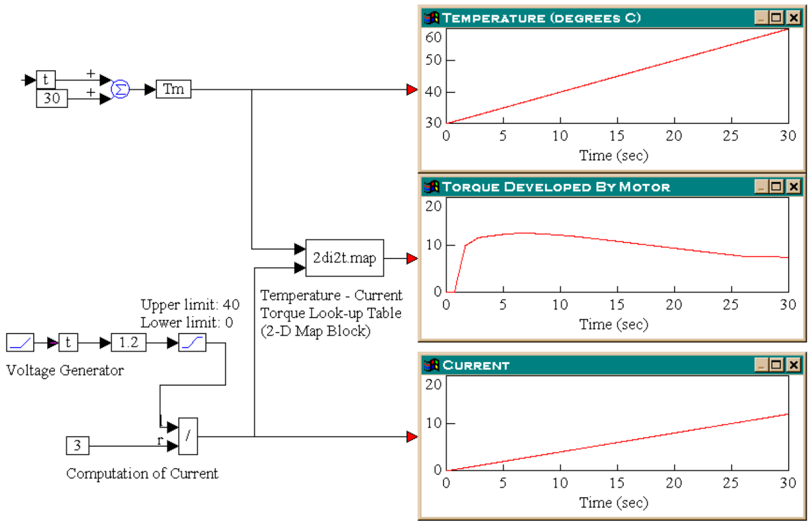
	Torque (Temperature Dependent) (N-M)			
Current (A)	**Torque at 30^o C**	**Torque at 40^o C**	**Torque at 50^o C**	**Torque at 60^o C**
0.	0.	0.	0.	0.
0.3	0.	0.	0.	0.
0.68	10	9.5	9.10	8.7
1.15	11.8	11.21	10.74	10.27
2.16	12.77	12.13	11.62	11.19
2.86	13.04	12.39	11.87	11.35
3.7	12.86	12.22	11.70	11.18
4.36	12.66	12.03	11.52	11.01
5.74	11.84	11.25	10.77	10.30
6.73	11.18	10.62	10.17	9.73
10.5	8.62	8.19	7.84	7.50
11.5	8.62	8.19	7.84	7.50

- The motor temperature profile is given by $T_m = (30 + t)^0$ C, where T_m is the motor temperature and *t* is time in seconds.
- A data file named 2DI2T.MAP is formatted as shown below:

	30.	**40.**	**50.**	**60.**
0.	0.	0.	0.	0.
0.3	0.	0.	0.	0.
0.68	10	9.5	9.10	8.7
1.15	11.8	11.21	10.74	10.27
2.16	12.77	12.13	11.62	11.19
2.86	13.04	12.39	11.87	11.35
3.7	12.86	12.22	11.70	11.18
4.36	12.66	12.03	11.52	11.01
5.74	11.84	11.25	10.77	10.30
6.73	11.18	10.62	10.17	9.73
10.5	8.62	8.19	7.84	7.50
11.5	8.62	8.19	7.84	7.50

The values of the temperature are entered in row 1, starting with column 2. The values of current are entered in column 1, starting with row 2. The values of the current and the temperature are shown in bold type for clarity.

Using a 2D `map` block, the system simulation can be realized as:



In addition to the blocks used in Example 1, a `variable` *t*, defined as simulation time, is connected to the output of the `ramp` block. A `map` block is used to access 2DI2T.MAP. Because this data file is a 2-D look-up table, the `map` block accepts two inputs: temperature (the independent variable in the first row) and current (the independent variable in the first column).

To generate the temperature profile, a `const` block of 30 is added to `variable` *t*, fed through another `variable` *Tm*, and monitored in the top `plot` block.

As before, the outputs of the `map` block, and the `/` block are monitored to observe the profiles of the motor torque and current, respectively.

During simulation, when the temperature is ≥30° C and <40° C, the second column of data is used to generate the torque profile. Similarly, for temperatures that are ≥40° C and <50° C or ≥50° C and <60° C, data in the third and fourth columns is used respectively.

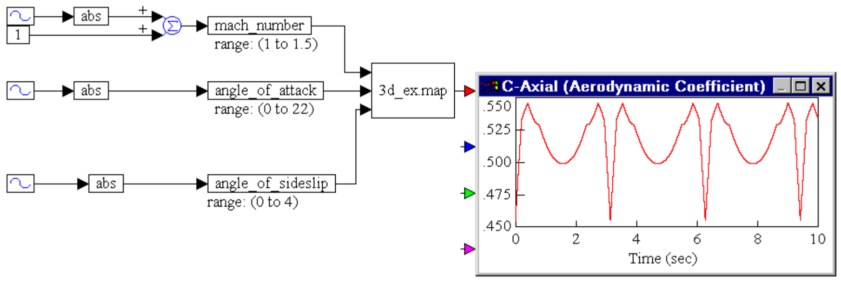
3. 3-D look-up table

The structure and usage of a VisSim diagram that includes a 3-D look-up table is very similar to a 2-D look-up table. The major difference is in the specification of the data file to be used by the 3-D `map` block. As an example, consider the following data file:

```
#3D_EX.MAP
--Table_Caxial_0 5 10 3
--Mach No. breakpoints
5
   1.05   1.10   1.20   1.35   1.50
--Angle_of_Attack breakpoints
10
   4.00   6.00   8.00  10.00  12.00  14.00  16.00  18.00  20.00  22.00
--Angle_of_Sideslip breakpoints
3
   0.00   2.00   4.00
--
-- Angle_of_Sideslip =   0.00

   0.493   0.533   0.550   0.529   0.496
   0.492   0.532   0.549   0.529   0.497
   0.491   0.530   0.547   0.529   0.497
   0.488   0.528   0.545   0.529   0.497
   0.485   0.525   0.541   0.529   0.497
   0.481   0.520   0.536   0.529   0.497
   0.474   0.513   0.528   0.530   0.498
   0.465   0.504   0.517   0.530   0.498
   0.452   0.488   0.482   0.530   0.498
   0.430   0.439   0.444   0.531   0.499
--
-- Angle_of_Sideslip =   2.00
   0.493   0.533   0.550   0.529   0.496
   0.492   0.532   0.549   0.529   0.497
   0.490   0.530   0.547   0.529   0.497
   0.488   0.528   0.544   0.529   0.497
   0.485   0.525   0.541   0.529   0.497
   0.480   0.520   0.535   0.529   0.497
   0.474   0.513   0.528   0.530   0.498
   0.465   0.503   0.516   0.530   0.498
   0.451   0.487   0.480   0.530   0.498
   0.429   0.437   0.442   0.531   0.499
--
-- Angle_of_Sideslip =   4.00
   0.493   0.532   0.549   0.529   0.496
   0.491   0.531   0.548   0.529   0.497
   0.490   0.530   0.546   0.529   0.497
   0.487   0.527   0.543   0.529   0.497
   0.484   0.524   0.540   0.529   0.497
   0.479   0.519   0.534   0.529   0.497
   0.473   0.512   0.527   0.530   0.498
   0.463   0.502   0.514   0.530   0.498
   0.449   0.485   0.476   0.530   0.498
   0.426   0.433   0.437   0.531   0.499
--
```

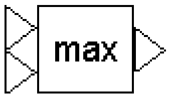
This data corresponds to one of the aerodynamic coefficients of a projectile in motion, traveling at speeds ranging from 1 to 1.5 mach. The value of the coefficient varies with three parameters: mach number, angle of attack, and angle of sideslip. Assuming sinusoidal variations in all three parameters, a diagram that uses this data file can be realized as shown on the next page.



Three `sin` blocks produce sinusoidal variations of amplitudes 0.5, 22, and 4 for the three `variables` *mach_number*, *angle_of_attack*, and *angle_of_slideslip*. Three `abs` blocks ensure that the values attained by the `variables` are strictly positive. A constant value of 1 is used to obtain a variation in the range (1, 1.5) for *mach_number*.

The outputs of the three `variable` blocks are fed into a `map` block that points to the map file 3D_EX.MAP, whose contents are shown above. The resulting value of the C-axial aerodynamic coefficient are shown in the `plot` block.

max



$$y = \begin{cases} x_1 & \text{if } x_1 > x_2 \\ x_2 & \text{if } x_1 < x_2 \end{cases}$$

Block Category: Nonlinear

The `max` block compares scalar inputs for a higher value and generates an output signal with the higher value.

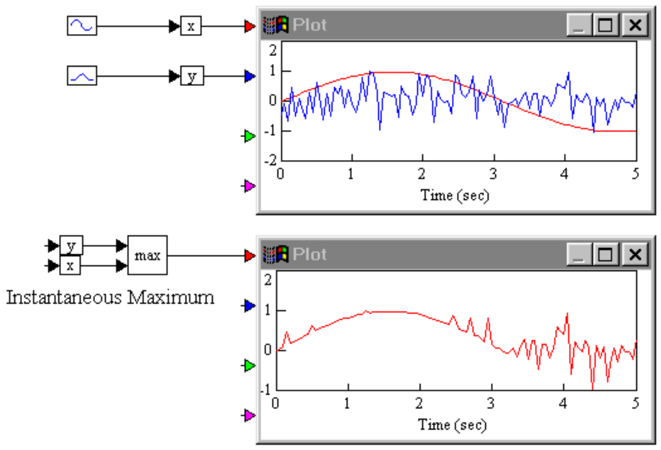
Examples

1. Comparison of two values

Consider the equation:

$z = \max(x,y)$

If x is a sinusoid that varies between -1 and +1, and y is a uniform random variable that varies between -1 and +1, this equation can be realized as shown on the next page.

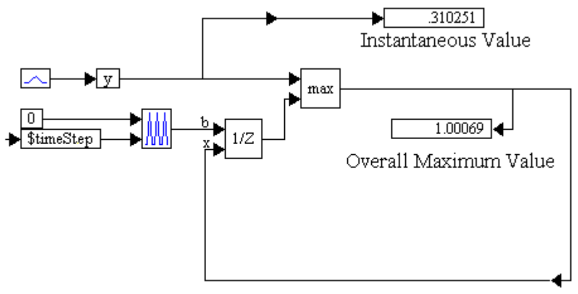


2. Computation of the maximum value of a given time-varying signal

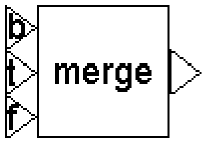
Consider the equation:

$z = max(y)$

where y is a uniform random variable that varies between -1 and +1. To find the maximum value that z attains, create the following diagram:



A `unitDelay` block stores the previous value of y. The `max` block compares the current and previous values of y. The larger of the current and previous values is fed back into the `unitDelay` block for the next round of comparisons. This way, the output of the `max` block is always the largest encountered value of y. The working details of this procedure are best examined by single-stepping through the simulation.



merge

$$y = \begin{cases} x_2 & \text{if } |x_1| \geq 1 \\ x_3 & \text{if } |x_1| < 1 \end{cases}$$

Block Category: Nonlinear

The `merge` block examines x_1 (Boolean signal) to determine the output signal. The letters b, t, and f on the input connector tabs stand for Boolean, True, and False. The `merge` block accepts scalar, vector, and matrix input.

The `merge` block is particularly well-suited for performing if-then-else decisions.

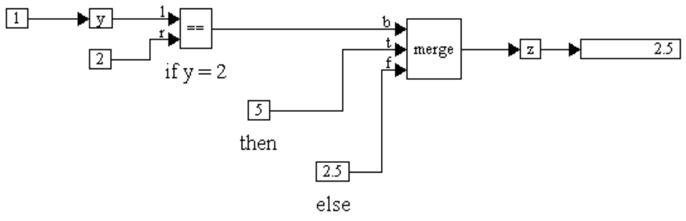
Examples

1. Simple merge

Consider the equation:

If y = 2, then z = 5, else z = 2.5

This equation can be realized as:

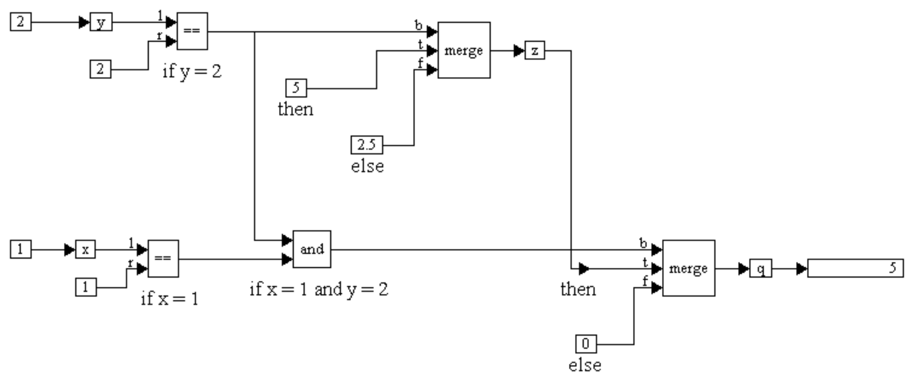


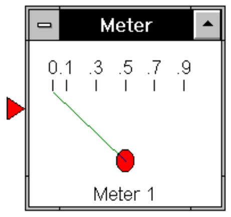
2. Cascade merge

Consider the equation:

If x = 1 and y = 2, then q = z, else q = 0

where (if y = 2, then z = 5, else z = 2.5). This logical relation can be realized as shown on the next page.



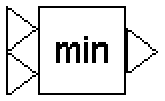


meter

display = x_1

Block Category: Signal Consumer

The `meter` block displays signals in either a gauge- or bar-style display. Initially, the `meter` block appears as a gauge-style display with one input connector tab. For more information, see page 75.



min

$$y = \begin{cases} x_1 & \text{if } x_1 < x_2 \\ x_2 & \text{if } x_1 > x_2 \end{cases}$$

Block Category: Nonlinear

The `min` block compares two scalar inputs for a lower value and generates an output signal with the lower value.

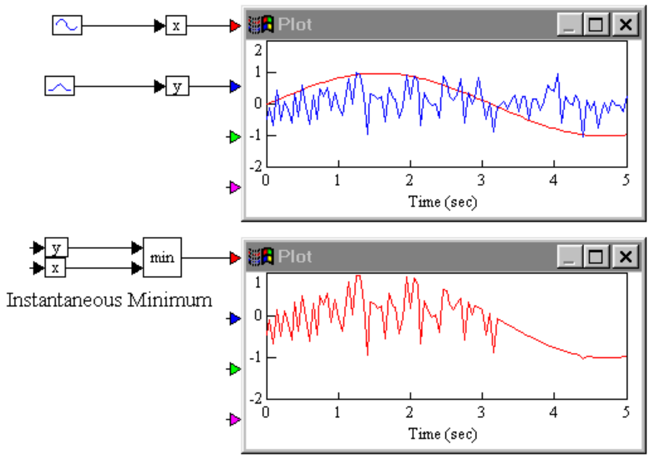
Examples

1. Comparison of two values

Consider the equation:

$z = \min(x,y)$

If *x* is a sinusoid that varies between -1 and +1, and *y* is a uniform random variable that varies between -1 and +1, this equation be realized as:

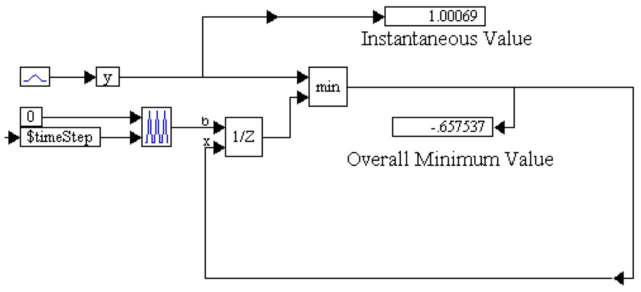


2. Computation of the minimum value of a given time-varying signal

Consider the equation:

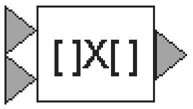
$z = \min(y)$

where *y* is a uniform random variable that varies between -1 and +1. To find the minimum value that *z* attains, create the following diagram:



A `unitDelay` block stores the previous value of *y*. The `min` block compares the current and previous values of *y*. The smaller of the current and previous values is fed back into the `unitDelay` block for the next round of comparisons. This way, the output of the `min` block is always the smallest encountered value of *y*. The working details of this procedure are best examined by single-stepping through the simulation.

multiply



Block Category: Matrix Operation

The `multiply` block performs a matrix multiplication. The `multiply` block accepts two vector inputs and produces one vector output.

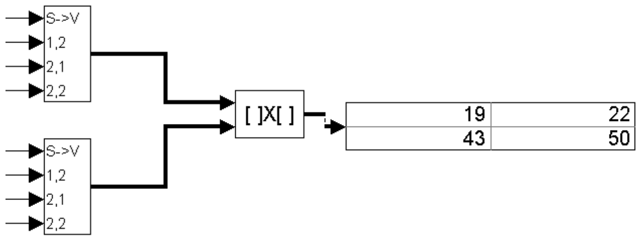
Multiplying scalars and vectors

To multiply two or more scalars, use the `*` block, as described on page 142.

To perform a single value summation of an element-by-element multiply of two vectors, use the `dotProduct` block, as described on page 181.

Examples

1. Simple matrix multiply



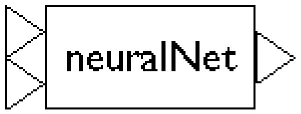
Here

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Then

$$\mathbf{AB} = \begin{bmatrix} 1(5)+2(7) & 1(6)+2(8) \\ 3(5)+4(7) & 3(6)+4(8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

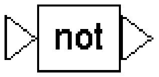
neuralNet



The `neuralNet` block excels at nonlinear system identification, problem diagnosis, decision-making, prediction, and other problems where pattern recognition is important and precise computational answers are not readily available. Typical uses of the `neuralNet` block include the identification of a chemical plant and the training of a moving cart to balance a vertical pole.

To use the `neuralNet` block, you must install the VisSim/Neural-Net software on your computer. For more information on the `neuralNet` block, see the *VisSim/Neural-Net User's Guide.*

not



$$y = \begin{cases} 1 & \text{if } x_1 = 0 \\ 0 & \text{otherwise} \end{cases}$$

Block Category: Boolean

The `not` block produces the Boolean NOT of the input signal. The output is true when the input is false; and the output is false when the input is true.

If you click the right mouse button over the `not` block, the Boolean block menu appears allowing you to assign a different function to the block.

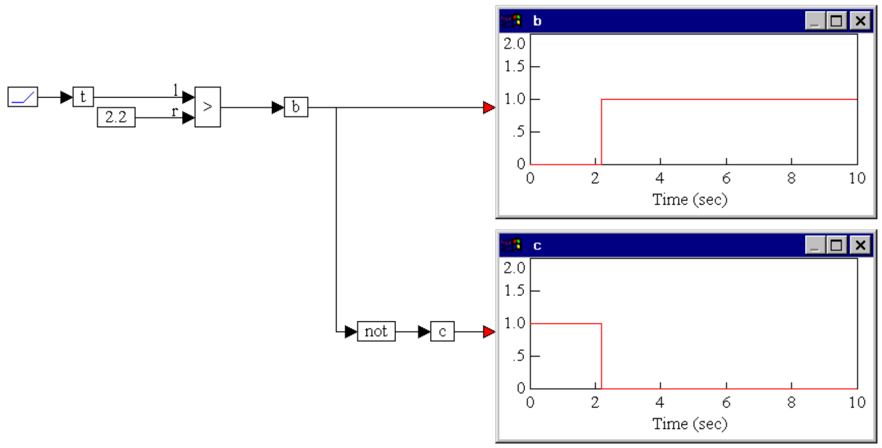
Examples

1. Using a not block

Consider a variable *c* such that:

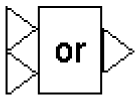
$$c = \overline{b}$$

or in other words, *c* = not(*b*). Furthermore, assume that *b* is true if $t > 2.2$; else *b* is false, where *t* is the simulation time. This system can be realized as shown below.



From the outputs obtained in the two `plot` blocks, *b*, given by the output of the > block is true only when *t* is > 2.2. This requires that *c*, which is defined to be `not`(*b*), be true only the range $t < 2.2$, as obtained in the bottom `plot` block.

or



$y = x_1$ bitwise OR x_2

Block Category: Boolean

The `or` block produces the bitwise OR of two to 256 scalar input signals. The output of the `or` block is true when at least one of the inputs is true. When all the inputs are false, the output is false.

If you click the right mouse button over the `or` block, the Boolean block menu appears allowing you to assign a different function to the block.

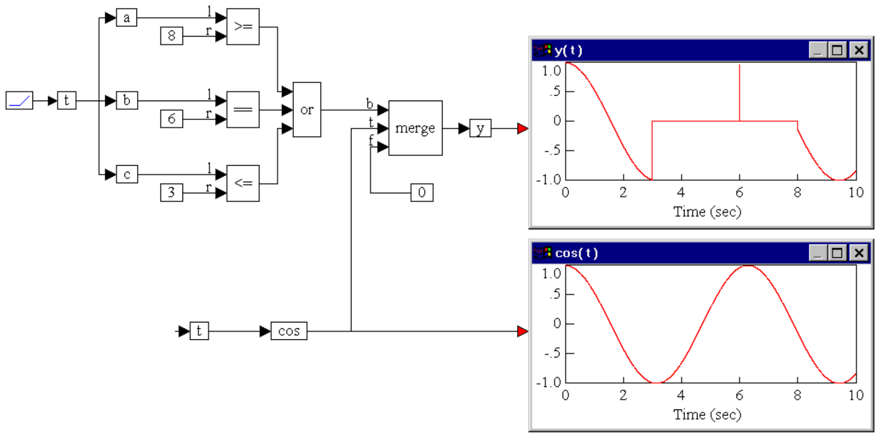
Examples

1. Computation of three inputs

Consider a variable *y* such that:

If $a \geq 8$ or $b = 6$ or $c \leq 3$, then $y = \cos(t)$; else $y = 0$

where *t* is simulation time. Furthermore, let *t* be the input to all three parameters *a*, *b*, and *c*. This system can be realized as shown on the next page.



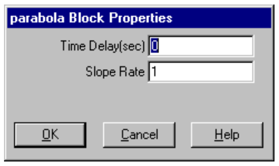
During simulation, the `or` block evaluates to false in the interval $t = (3,8)$, except for the instant $t = 6$. In this case, the `variable` y takes on the value of 0. The output of `or` evaluates to true in the remaining parts of the simulation, and as a result, y takes on the value of cos(t) in these periods, including the instant $t = 6$.

parabola

$$y = \text{slope} \cdot (t - t_{delay})^2$$

Block Category: Signal Producer

The `parabola` block creates a parabolic signal.


parabola Block Properties

Time Delay(sec) 0

Slope Rate 1

Time Delay: Indicates an offset that is used in the calculation of a signal. For a constant-valued delay, wire the block into a `unitDelay` or `timeDelay` block with an initial condition of the desired constant value.

Specify the offset in seconds. The default value is 0.

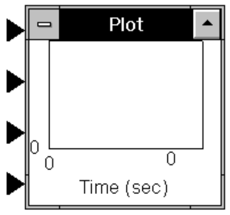
Slope Rate: Scales the curvature of the parabola. The default value is 1.

parameterUnknown

Block Category: Optimization

The `parameterUnknown` block works with the `cost` block to find globally optimal values that minimize a scalar cost function. For more information, see Chapter 8, "Performing Global Optimization."

plot

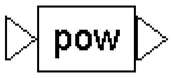


$$\text{plot} = x_1 \ldots x_4$$

Block Category: Signal Consumer

The `plot` block displays simulation data graphically in a customizable plots. For more information, see page 59.

pow



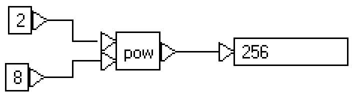
$$y = \begin{cases} x^{\text{exponent}} \\ or \\ {x_1}^{x_2} \end{cases}$$

Block Category: Arithmetic

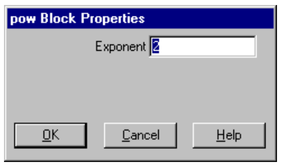
The `pow` block creates an output signal based on the value of the input signal raised to the power of a specified exponent. Inputs can be scalars, vectors, or matrices. When the input is a vector or matrix, the `pow` block computes the output on an element-by-element basis.

The pow block is useful for solving equations of the type $y = x^z$. Do not use the pow block to compute matrix dot products, such as $\mathbf{Y} = \mathbf{A}^2$, where the dot product is implied. Instead use the dotProduct block, as described on page 181.

By adding an input connector tab to the pow block, you can specify an external exponent parameter to override the block's exponent parameter. For example:



This diagram raises two to the eighth power. The display block verifies the results. The main advantage of setting the exponent externally is that the value of the exponent can be varied dynamically as the simulation progresses.


pow Block Properties

Exponent 2

Exponent: Specifies the power to which the input signal is raised. The default is 2.

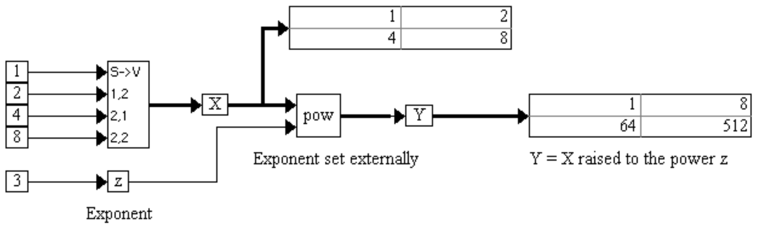
Examples

1. Raising a matrix input to a power

Consider the equation:

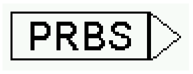
$$\mathbf{Y} = \mathbf{X}^Z$$

where $\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 4 & 8 \end{bmatrix}$ and z = 3. This equation can be realized as:



The pow block raises each element of the incoming matrix to power 3.

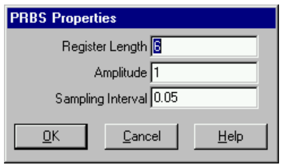
PRBS



Block Category: Random Generator

The PRBS block produces a pseudo-random sequence of unit amplitude pulses. You can control the frequency of oscillation and the register length.

The PRBS block can be used to see how random perturbations affect a system. System Identification software can use the output of a PRBS block to create a mathematical model of the system.


PRBS Properties

Register Length 6

Amplitude 1

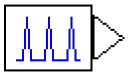
Sampling Interval 0.05

Register Length: Controls when the sequence of pulses repeats. The default is 6

Amplitude: Specifies the maximum strength of the output signal. The default is 1.

Sample Interval: Indicates the frequency of oscillation. The default is 0.05.

pulseTrain

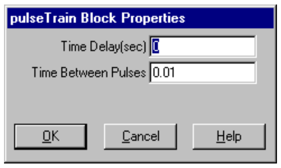


$$y = \begin{cases} 1 & \text{if } t \bmod_{\text{time pulse}} == 0 \\ 0 & \text{otherwise} \end{cases}$$

Block Category: Signal Producer

The pulseTrain block produces a sequence of unit amplitude pulses separated by zeros. You cannot control the duration of the pulse; you can only control the time between pulses.

You can add two input connector tabs to the pulseTrain block. The top input connector tab lets you specify an external time delay; the bottom one lets you specify an external time between pulses. These additional inputs override the existing parameters. If you add only one input connector tab, it corresponds to the external delay.

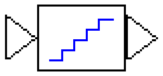


Time Delay(sec) 0
Time Between Pulses 0.01

Time Delay (sec): Specifies, in seconds, how long to delay before calculating the value of the output signal. The default is 0.

Time Between Pulses: Specifies the time between pulses. This is useful for clocking delays and sample holds. The default is 0.01.

quantize



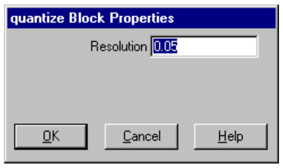
$$y = \left[\text{integer part}\left(\frac{x}{\text{resolution}}\right)\right]\text{resolution}$$

Block Category: Nonlinear

The `quantize` block is useful for simulating approximations of a continuously varying signal that possibly requires the use of an infinite number of values or levels by a discontinuous signal with a finite number of values.

The `quantize` block rounds the precision of input signal based on the signs of the input and the resolution. When the resolution is positive, the signal is rounded down to -∞. For example, 1.9 quantized to a resolution of 1 becomes 1, and -1.9 quantized to a resolution of 1 becomes 2. When the resolution is negative, the signal is rounded to +∞. For example, 1.1 quantized to a resolution of -1 becomes 2, and -1.1 quantized to a resolution of -1 becomes -1.

The `quantize` block is applicable to simulations that involve the conversion of analog signals to digital signals.



Resolution 0.05

Resolution: Specifies the value to which the input signal is rounded or truncated. The default is 0.05.

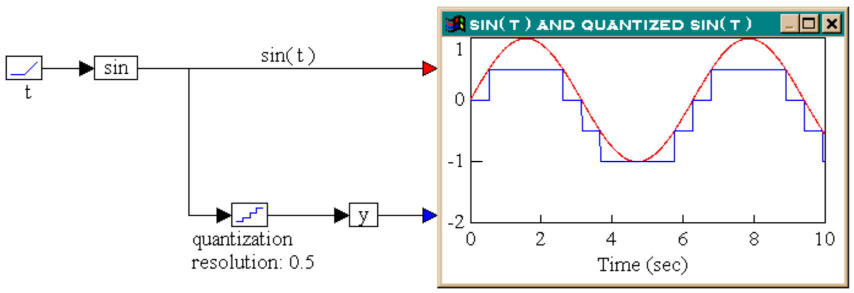
Examples

1. Quantization of a sinusoid: positive resolution

Consider a variable y such that:

$y = \sin(t)$

quantized with a resolution of +0.5. This equation an be realized as:



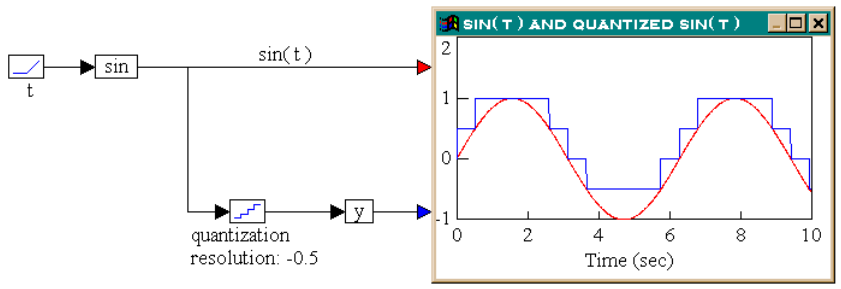
The `quantize` block approximates the sinusoid input using four values (0, +0.5, -0.5, and -1).

2. Quantization of a sinusoid: negative resolution

Consider a variable y such that:

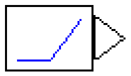
$y = \sin(t)$

quantized with a resolution of -0.5. This equation can be realized as:



The `quantize` block approximates the sinusoid input using four values (0, +0.5, +1, and -0.5). By comparing these results with those in Example 1, the effects of using positive and negative resolutions in a `quantize` block becomes clear.

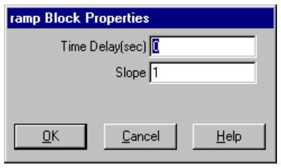
ramp



$y = \text{slope} \cdot (t - t_{delay})$

Block Category: Signal Producer

The `ramp` block creates a unit ramp signal based on simulation time.



Time Delay(sec) 0
Slope 1

Time Delay(sec): Indicates an offset that is used in the calculation of a signal. For a constant-valued delay, wire the `ramp` block into a `unitDelay` or `timeDelay` block with an initial condition of the desired constant value.

Specify the offset in seconds. The default is 0.

Slope: Specifies the ramp slope. The default is 1.

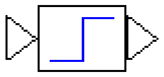
realTime

$y = t$

Block Category: Signal Producer

The `realTime` block provides the current time in milliseconds since the start of your VisSim session. Note that this is not simulation time.

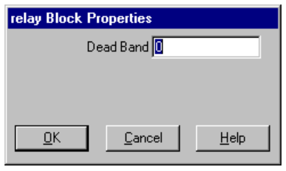
relay



$$y = \begin{cases} -1 & \text{if } x < \dfrac{\text{-deadband}}{2} \\ 1 & \text{if } x > \dfrac{\text{deadband}}{2} \\ 0 & \text{otherwise} \end{cases}$$

Block Category: Nonlinear

The `relay` block simulates a tri-state relay operator. This block is useful for simulation switches or switching operators.



relay Block Properties

Dead Band: 0

OK Cancel Help

Dead Band: Indicates the width of the zone of lost motion about the input signal's 0 value, thereby creating a tri-state relay operator (-1, 0, 1). When input is less than half the negative Dead Band value, the `relay` block outputs -1. When input is greater than half the positive Dead Band value, the `relay` block outputs +1. When input lies within the range (-Dead Band/2, +Dead Band/2), the `relay` block outputs 0. You cannot specify a negative value for this parameter. The default is 0.

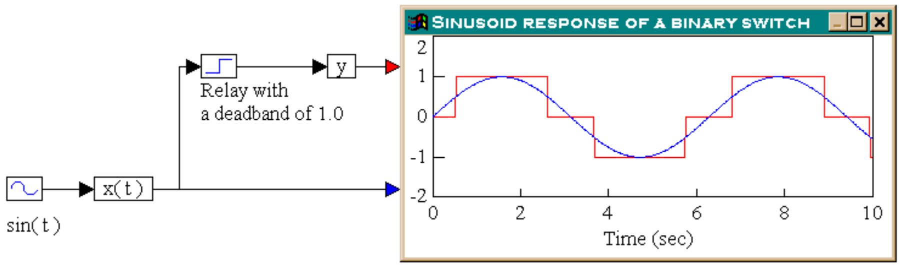
Examples

1. Constructing a tri-state switch

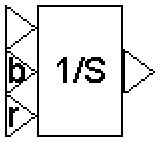
Consider a tri-state variable *y* such that:

$$y = \begin{cases} +1 \, if \;\; x(t) > 0.5 \\ -1 \, if \;\; x(t) < -0.5 \\ \;\; 0 \;\; otherwise \end{cases}$$

Assuming that *x*(*t*) = sin(*t*), this equation can be realized as shown on the next page.



The Dead Band of the `relay` block is set to 1.0. During simulation, the `relay` block changes its output state based on whether the input signal is greater than or less than Dead Band/2.



resetIntegrator (1/S)

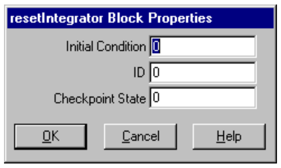
$$y = \begin{cases} \displaystyle\int_{t_{\text{start}}}^{t_{\text{end}}} x_1\, dt & \text{if } |x_2| < 1 \\ x_3 & \text{if } |x_2| \geq 1 \end{cases}$$

Block Category: Integration

The `resetIntegrator` block integrates the input signal with an optional reset capability. When the Boolean input (b) is 0, the `resetIntegrator` behaves like a normal integrator. When the Boolean input goes to 1, the `resetIntegrator` takes the value of the reset input (r) for as long as the Boolean value stays high.

The `resetIntegrator` block integrates the input signal using the integration algorithm established in the dialog box for the Simulate > Simulation Properties command. The available algorithms are Euler, trapezoidal, Runge Kutta 2d and 4th orders, adaptive Runge Kutta 5th order, adaptive Bulirsh-Stoer, and backward Euler (Stiff).

The inputs to the `resetIntegrator` block are x_1, x_2 (b), and x_3 (r).


resetIntegrator Block Properties

Initial Condition 0
ID 0
Checkpoint State 0

Initial Condition: Indicates the initial value of the integrator upon simulation start-up. This parameter can be overridden if x_2 is non-zero on the first step of the simulation. The default is 0.

ID: Represents an identification number for the block. This number keeps track of the state number that VisSim assigns to the integrator. The number of states in any VisSim diagram equals the number of integrators. The default is 0.

Checkpoint State: Contains the value of the integrator state at the checkpoint. If you have not checkpointed your simulation via the Simulate > Simulation Setup command, the default is 0.

Examples

1. Instantaneous momentary reset

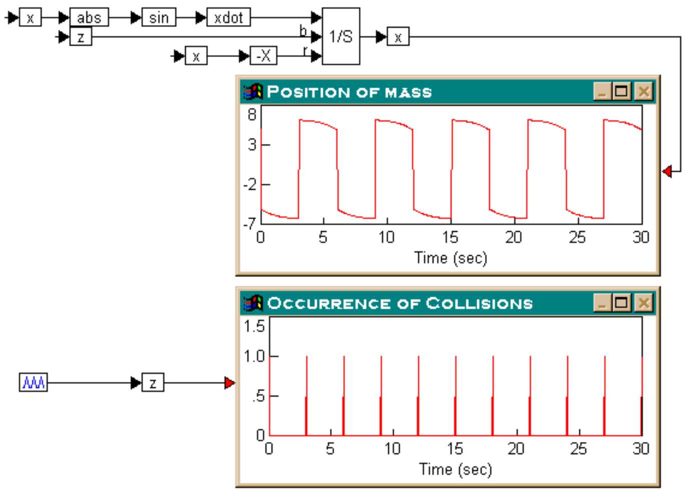
Consider a system whose dynamics are given by the differential equation:

$$\dot{x} = \sin|x|$$

When a particular variable *z* equals 1, *x* must be reset to -*x*. To make matters simple, assume that *z* becomes momentarily equal to 1, every three seconds, and that *x*(0) = 5.

Equations of this type are frequently used in kinematic systems that undergo collisions, electrical circuits that involve switching phenomena, chemical processes, and fluid dynamics.

This system can be realized as shown below.



As with any differential equation, the right-hand side of the equation is realized first by creating a `variable` *x*, and then connecting it successively to an `abs` block, a `sin` block, and another `variable` *xdot*.

At this point, only *xdot* is defined in terms of *x*. To define the relationship between *xdot* and *x*, *xdot* is fed into the top input tab of the `resetIntegrator` block, which is fed into the `variable` *x*.

The Boolean input tab of the `resetIntegrator` is fed by `variable` *z*, which generates pulses that are three seconds apart. The negative value of a given signal can be directly generated using the `-X` block, and then fed into the reset input tab of the `resetIntegrator`.

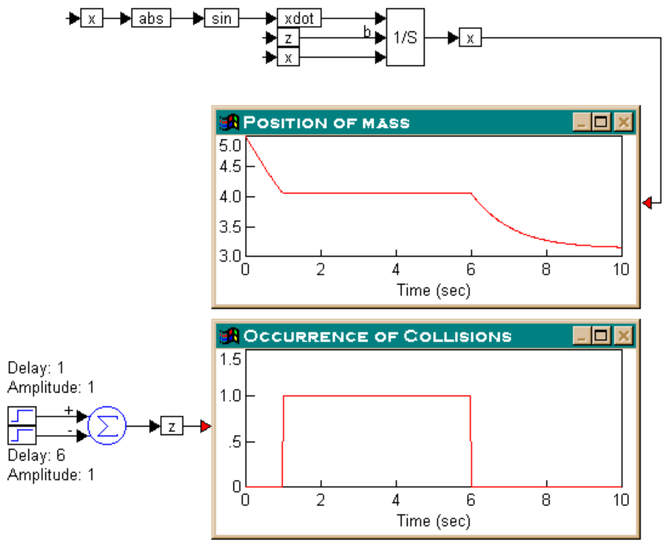
From the results of simulation shown in the two `plot` blocks, *z* becomes high every three seconds, and at each of these instances, the output of the `resetIntegrator` is reset to -*x*.

2. State reset for a duration

As mentioned above, the `resetIntegrator` output is held at the reset value as long as the Boolean input is high. To illustrate this property, consider the differential equation:

$$\dot{x} = \sin|x|$$

When a particular variable *z* is equal to 1, *x* must be held at its current value. Assume that $z = 1$ when $1 \le t \le 6$ and that $x(0) = 5$. This case can be realized as shown below.



To construct *z*, two `step` blocks and a `summingJunction` block are used. The delay and amplitude of the top `step` block are both set to 1; for the bottom `step` block, they are set to 6 and 1, respectively. By subtracting the outputs of the two `step` blocks and defining the output of the `summingJunction` block as *z*, $z = 1$ when $1 \le t \le 6$.

By coding *z* in this manner, *z* is redefined as $z(t) = u(t - 1) - u(t - 6)$ where $u(t)$ represents a unit step. Consequently, $u(t - 1)$ is a unit step delayed by 1 sec, and $u(t - 6)$ is a unit step delayed by 6 sec.

During simulation, the output of the `resetIntegrator` holds constant at $x(1)$ for the duration $1 \le t \le 6$.

rt-DataIn

Block Category: Real Time

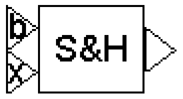
The rt-`DataIn` block, in conjunction with the File > Real Time Config command, lets you connect to an I/O real-time data card. To use this block and menu command, you must install the VisSim/Real-TimePRO or VisSim DACQ software on your computer. For information, see the *VisSim/Real-TimePRO User's Guide.*

rt-DataOut

Block Category: Real Time

The `rt-DataOut` block, in conjunction with the File > Real Time Config command, lets you to connect to an I/O real-time data card. To use this block and menu command, you must install the VisSim/Real-TimePRO or VisSim DACQ software on your computer. For information, see the *VisSim/Real-TimePRO User's Guide.*

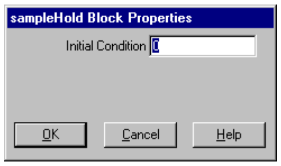
sampleHold



$$y = \begin{cases} x_2 & \text{if} \left| x_1 \right| \geq 1 \\ y_{\text{previous}} & \text{otherwise} \end{cases}$$

Block Category: Nonlinear

The `sampleHold` block latches an input value under the control of a clock signal, x_1, which is represented as Boolean input (b). When b is true, input signal x_2, which is represented as input (x) is sampled and held until b is true again. Boolean inputs can be regularly or irregularly spaced.


sampleHold Block Properties

Initial Condition: 0

Initial Condition: Indicates the initial condition for the `sampleHold`. The default is 0.

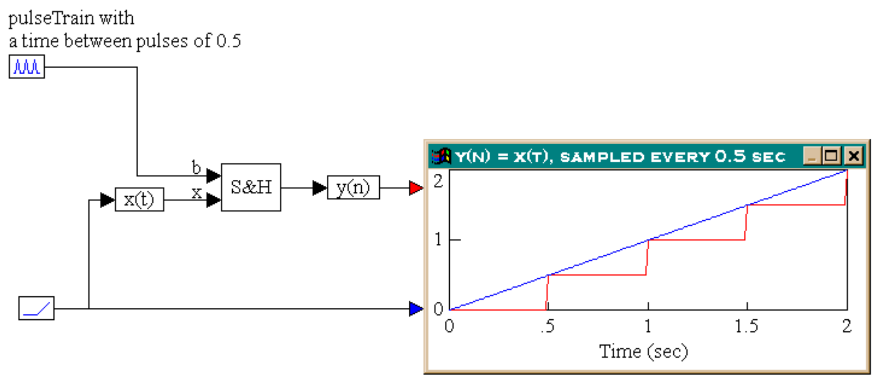
Examples

1. Sample and hold with regularly-spaced clock

Consider the equation:

$y(n) = x(t)$

sampled every 0.5 sec. Furthermore, let $x(t)$ be a ramp signal. This system can be realized as shown on the next page.


pulseTrain with
a time between pulses of 0.5
b
x
x(t)
S&H
y(n)
Y(N) = X(T), SAMPLED EVERY 0.5 SEC
2
1
0
0
.5
1
1.5
2
Time (sec)

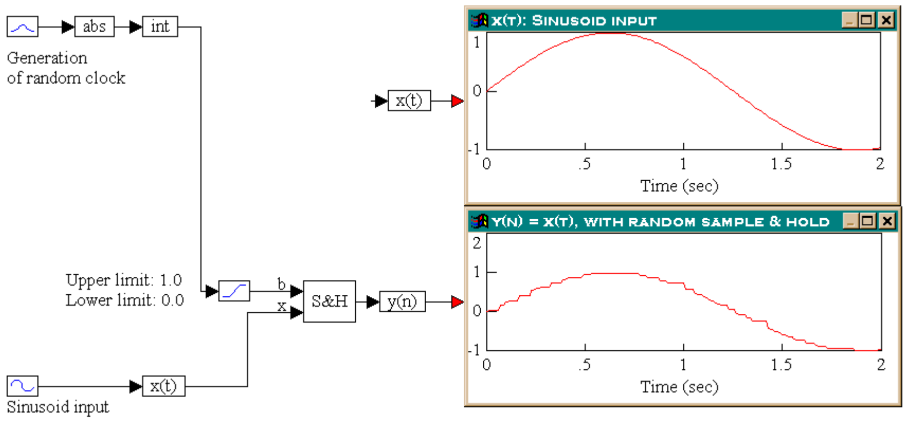
As seen in the `plot` block, the first clock pulse occurs at 0.5 sec. Until this time, the output of the `sampleHold` block is zero. At 0.5 sec, the input signal is sampled and the value is used as output for the `sampleHold` block. The output of the `sampleHold` block is held at this value until the occurrence of the next clock pulse at 1.0 sec. At this time, the input signal is again sampled and the new value is presented to the output of the `sampleHold` block, and the process repeats itself.

2. Sample and hold with irregularly-spaced clock

Consider the equation:

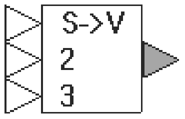
$y(n) = x(t)$

sampled randomly. Furthermore, let $x(t)$ be a sinusoid signal with a frequency of 2.5 rad/sec. This system can be realized as shown below.



A `sinusoid` block with a frequency of 2.5 rad/sec generates the sinusoid signal and a `gaussian` block produces a randomly varying signal. The randomly varying signal is converted to a random clock by taking the absolute value of the random signal and then using only the integer portion of it. The output of the `int` block is passed through a `limit` block to restrict the signal to the range (0, 1). The output of the `limit` block is connected to the top input of the `sampleHold` block. The output of the `sampleHold` block is connected to the `variable` *y*(*n*),which is connected to a `plot` block. The actual input, *x*(*t*) is monitored separately in another `plot` block.

By comparing the outputs in the two `plot` blocks, the output of the `sampleHold` block is a randomly sampled and held version of the input sinusoid.



scalarToVec

Block Category: Annotation

The `scalarToVec` block reduces wiring clutter by letting you combine input signals into a single vector wire. This is usually a prerequisite for performing vector and matrix algebra. Use the `vecToScalar` block to unbundle vector wires.

Examples

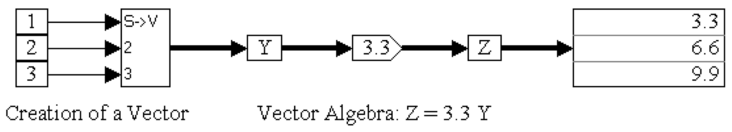
1. **Creation of a vector**

Consider the equation:

Z = 3.3 **Y**

where **Z** and **Y** are vectors. Further, assume that **Y** = [1 2 3]^T.

This equation can be realized as:



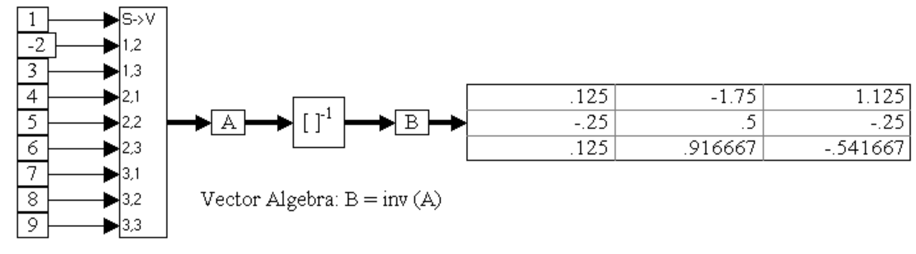
The `display` block displays all the elements of the incoming vector line. The results indicate that the vector operation is performed correctly.

2. **Creation of a matrix**

Consider the equation B = **A**$^{-1}$, where

$$\mathbf{A} = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

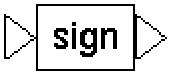
The above equation can be realized as:



Creation of a Matrix

The display block displays all the elements of the incoming matrix line. The results indicate that the matrix operation is performed correctly.

sign



$$y = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

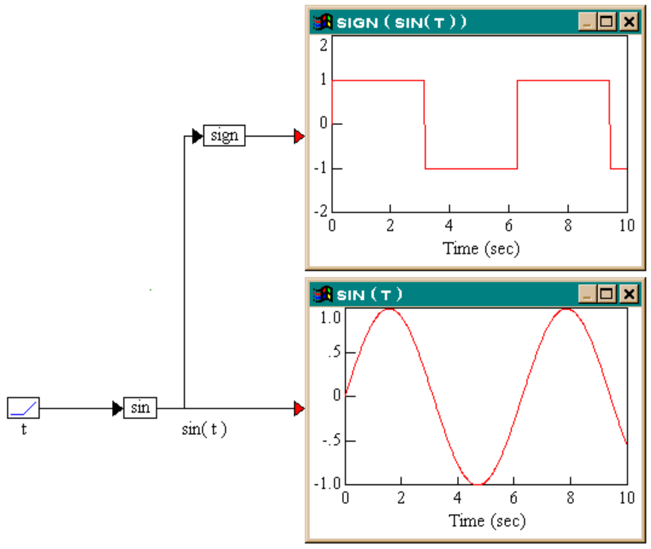
Block Category: Arithmetic

The sign block determines the sign of the scalar input signal. The sign block outputs +1 when the input is greater than zero; -1 when the input is less than zero; and 0 when the input is zero.

Examples

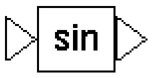
1. Computation of the sign of sin(*t*)

This equation can be realized as:



The results obtained indicate that when the input is greater than zero, the `sign` block outputs +1, when the input is less than zero the output is -1, and when the input is zero, the output is 0.

sin



$y = \sin x$

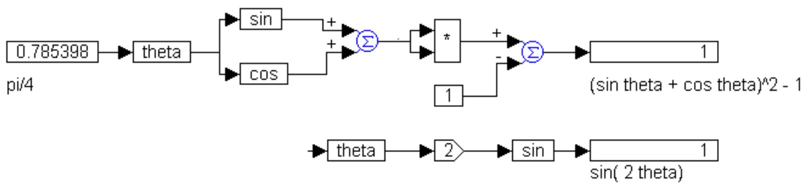
Block Category: Transcendental

The `sin` block produces the sine function of the input signal. The input signal is represented in radians.

Examples

1. Computation of sin(2θ) = (cos(θ) + sin(θ))^2 - 1

With θ chosen to be π/4, the above trigonometric identity can be realized as:



sinh

$$y = \frac{e^x - e^{-x}}{2}$$

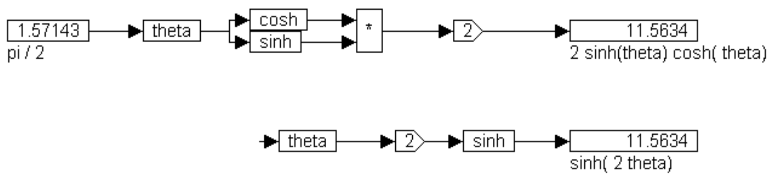
Block Category: Transcendental

The `sinh` block produces the hyperbolic sine function of the input signal. The input signal is represented in radians.

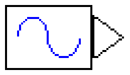
Examples

1. Computation of sinh(2θ) = 2 sinh(θ) cosh(θ)

With θ chosen to be π/2, the above trigonometric identity can be realized as:



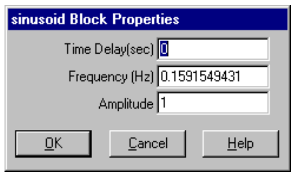
sinusoid



$$y = A \cdot \sin\left(\omega \cdot (t - t_{delay})\right)$$

Block Category: Signal Producer

The `sinusoid` block creates a unit sine wave signal.



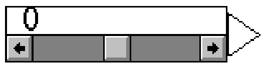
Time Delay(sec) 0
Frequency (Hz) 0.1591549431
Amplitude 1

Time Delay (sec): Indicates an offset that is used in the calculation of a signal. For a constant-valued delay, wire the `sinusoid` block into a `unitDelay` or `timeDelay` block with an initial condition of the desired constant value. Specify the offset in seconds. The default is 0.

Frequency (rad/sec): Controls the frequency of oscillation of the output signal. Specify the frequency in radians per second. For example, if you specify a frequency of 1, one oscillation completes in 2π seconds. If you specify a frequency of π, one oscillation completes in 0.5 seconds. The default is 1.

Amplitude: Specifies the maximum strength of the output signal. The default is 1.

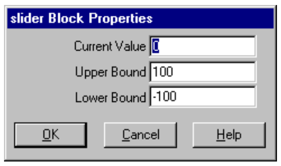
slider



Block Category: Signal Producer

The `slider` block allows mouse input to dynamically modify a signal value during a simulation, between a lower and upper bound in 1% and 10% increments. The `slider` block displays the current value applied to the signal. Use the scroll bar to adjust the signal value.

Slider precision is affected by the High Precision Display parameter under Preferences in the dialog box for the Edit > Preferences command. When activated, slider precision is shown at up to 15 significant digits; when de-activated, slider precision is shown at up to 6 significant digits.


slider Block Properties

Current Value 0
Upper Bound 100
Lower Bound -100

Current Value: Specifies the initial value of the slider output signal. The default is 0.

Upper Bound: Specifies the largest value the slider output signal can attain. The default is 100.

Lower Bound: Specifies the smallest value the slider output signal can attain. The default is -100.

sqrt

$y = \sqrt{x}$

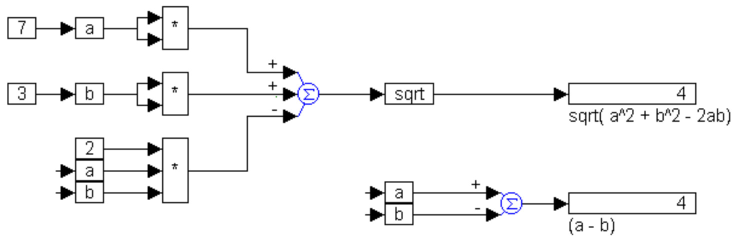
Block Category: Transcendental

The `sqrt` block produces an output signal that is the square root of a positive input signal. The `sqrt` block does not accept negative inputs. And, there is no square root of 0.

Examples

1. Computation of the sqrt($a^2 + b^2 - 2ab$) = ($a - b$)

With *a* and *b* chosen to be 7 and 4 respectively, the above equation can be realized as:



stateSpace

Block Category: Linear system

The `stateSpace` block is used to represent a multi-input multi-output linear system in state-space form. The state-space matrices can be specified in the following ways:

- **As an .M file created with VisSim:** The Analyze > Linearize command generates ABCD state-space matrices from a nonlinear system by numerically evaluating the matrix perturbation equations at the time the simulation was halted. For more information, see the *VisSim/Analyze User's Guide.*
- **As an .M file created with a text editor:** When you create a .M file with a text editor, follow these rules: start each matrix on a new line; enclose matrix elements in square brackets and terminate with a semi-colon; separate matrix elements with spaces; separate matrix rows with semi-colons.

The following is an example of a user-written .M file:

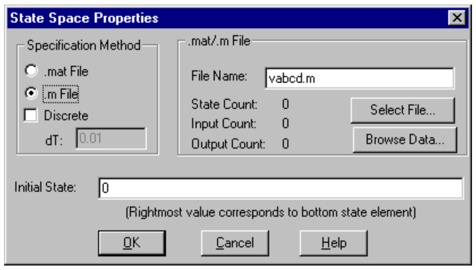
```
function [a,b,c,d] =vabcd
a = [-.396175 -1.17336 ; 5.39707 .145023 ];
b = [-.331182  ; -1.08363 ];
c = [0 1 ];
d = [0 ];
```

Note that MatLab commands other than array initialization are not allowed.

- **As a .MAT file created with MatLab:** Generating .MAT files is described in the MatLab documentation. Note that when you save the ABCD matrices to a file, the names of the matrices are not important; however, the order in which they appear is.

When you simulate the block diagram, VisSim numerically solves the `stateSpace` block.

VisSim supports state-space systems up to the 90th order.


State Space Properties

Specification Method
.mat File
[✓ .m File]
Discrete
dT: 0.01

.mat/.m File
File Name: vabcd.m
State Count: 0
Input Count: 0
Output Count: 0

Initial State: 0
(Rightmost value corresponds to bottom state element)

Specification Method: You have the choice of three specification methods:

- **Discrete:** Indicates a discrete Z-domain system. Enter the time step for the discrete transfer function in the dT box. By default, this parameter is de-activated, which indicates a continuous transfer function.
- **.mat File:** Indicates that the system is to be specified as a MatLab .MAT file. Specify the name of the .MAT file in the .mat/.m File group box.
- **.m File:** Indicates that the system is to be specified as an .M file. Specify the name of the .M file in the .mat/.m File group box.

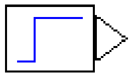
dT: Specifies the time step for the discrete system. By default, VisSim uses step size parameter from the Simulate menu’s Simulation Setup command.

Initial State: Specifies initial values for the states in the block. The values are right-adjusted. The right-most value corresponds to the lowest order state. Unspecified states are set to 0.

File Name: Indicates the name of the .M or .MAT file to be used as input to the `stateSpace` block. You can type the file name directly into this box or select one using the Select File button. To open the specified file with the default text editor, click on the Browse Data button.

Input Count, Output Count, and State Count: Indicate the number of inputs to the block, the number of outputs from the block, and the number of system states. The number of system states is determined by the size of the A matrix. These options are read-only.

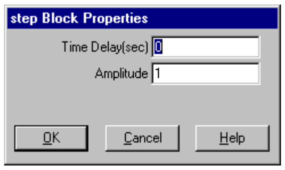
step



$$y = \begin{cases} 0 & \text{if } t < t_{\text{delay}} \\ \text{A} & \text{otherwise} \end{cases}$$

Block Category: Signal Producer

The `step` block creates a unit step signal.


step Block Properties

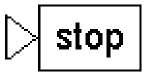
Time Delay(sec) 0

Amplitude 1

Time Delay(sec): Specifies, in seconds, how long to delay before calculating the value of the output signal. The default is 0.

Amplitude: Indicates the maximum strength of the output signal. The default is 1.

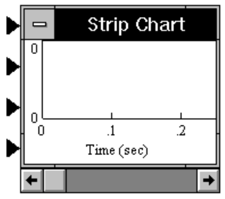
stop



$$\text{If } x \begin{cases} > 2 & \text{halt simulation unconditionally} \\ > 1 & \text{halt current run; start next run} \end{cases}$$
$$\text{Else} \quad \text{normal}$$

Block Category: Signal Consumer

The `stop` block conditionally halts a simulation when the input signal is non-zero. For a multi-run simulation, when the input value is 1, VisSim halts the current run, increments $runCount, and starts the next run if the Auto Restart parameter in the dialog box for the Simulate > Simulation Properties command has been activated. When the input value is 2, VisSim stops the multi-run sequence altogether.

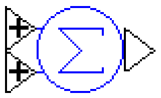


stripChart

$\text{strip chart} = x_1 \ldots x_4$

Block Category: Signal Consumers

The `stripChart` block displays up to four signals in a customizable scrolling window. For more information, see page 67.



summingJunction

$y = x_1 + x_2 + \ldots x_n$

Block Category: Arithmetic

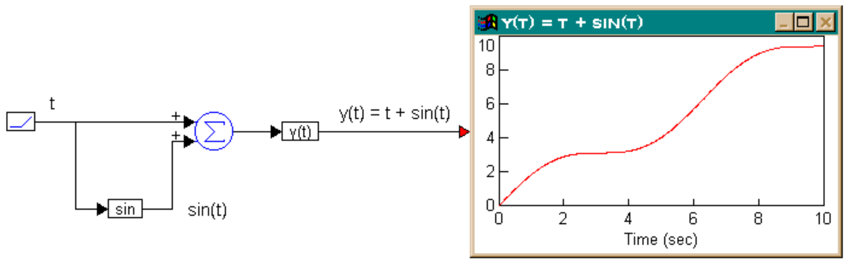
The `summingJunction` block produces the sum of two signed input signals. You can toggle the sign of the input signals (switch from positive to negative and vice versa) by holding down the CTRL key and clicking the right mouse button over the connector tab.

Inputs can be scalars, vectors, and matrices. When vector and matrix inputs are of unequal lengths, the `summingJunction` block defines the output vector or matrix to be the maximum composite size of all the incoming vectors or matrices and extends all other incoming vectors and matrices to match the length of the longest incoming vector or matrix, by padding each of them with the requisite number of zeros.

Examples

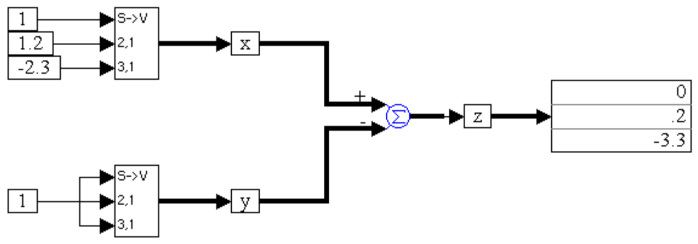
1. Addition of two scalar quantities

Consider the equation $y(t) = t + \sin(t)$. This can be realized as:



2. Subtraction of two vectors

Consider a vector **x** = [1 1.2 -2.3], and another vector **y** = [1 1 1]. The vector difference **z** = **x** - **y** can be computed directly by using a `summingJunction` block as:



The vector display shows that the result of the simulation is a direct element-by-element subtraction of vector **y** from vector **x**.

3. Matrix addition and subtraction

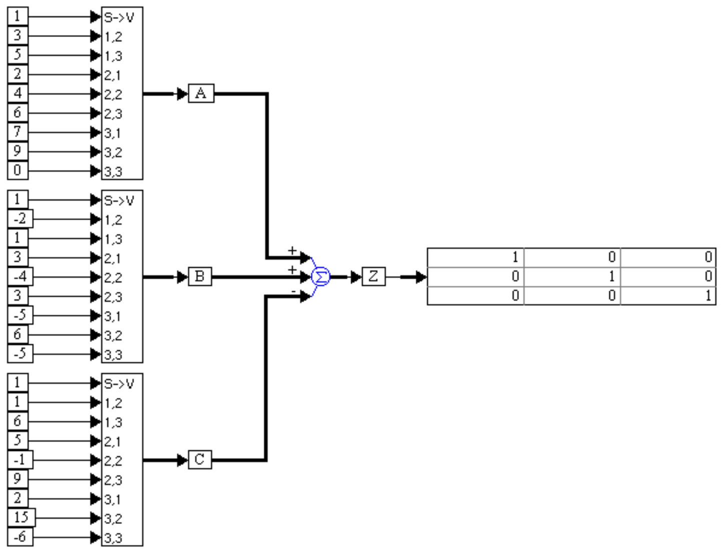
Consider the matrix equation:

$$\boldsymbol{Z} = \boldsymbol{A} + \boldsymbol{B} - \boldsymbol{C}$$

where:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 7 & 9 & 0 \end{bmatrix}; B = \begin{bmatrix} 1 & -2 & 1 \\ 3 & -4 & 3 \\ -5 & 6 & -5 \end{bmatrix}; C = \begin{bmatrix} 1 & 1 & 6 \\ 5 & -1 & 9 \\ 2 & 15 & -6 \end{bmatrix}$$

This equation can be realized as shown on the next page.



The matrix display shows that e result of the simulation is a direct element-by-element matrix operation of ***A*** + ***B*** - ***C***.

tan

$y = \tan x$

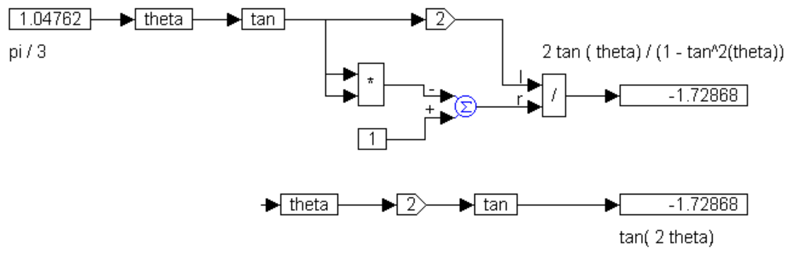
Block Category: Transcendental

The `tan` block produces the tangent of the input signal. The input signal must be represented in radians.

Examples

1. Computation of tan(2θ) = 2 tan(θ) / (1 - tan^2(θ))

With θ chosen to be π/3, the above trigonometric identity can be realized as:



tanh

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

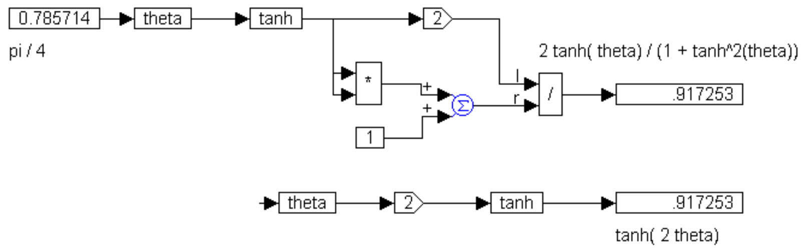
Block Category: Transcendental

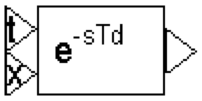
The `tanh` block produces the hyperbolic tangent of the input signal. The input signal must be represented in radians.

Examples

1. Computation of tanh(2θ) = 2 tanh(θ) / (1 + tanh^2(θ))

With θ chosen to be π/4, the above trigonometric identity can be realized as:



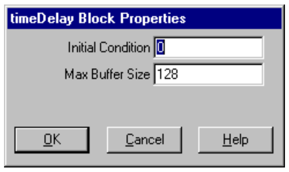


timeDelay

$y = x(t - T_d)$

Block Category: Time Delay

The `timeDelay` block delays the input signal for an absolute time. The input connector tabs are marked t (for the time delay) and x (for the main signal). This block is intended to model a continuous delay in a continuous simulation. Use the `unitDelay` block to model a digital delay.



timeDelay Block Properties

Initial Condition: 0

Max Buffer Size: 128

Initial Condition: Sets an initial condition for the delay. The default is 0.

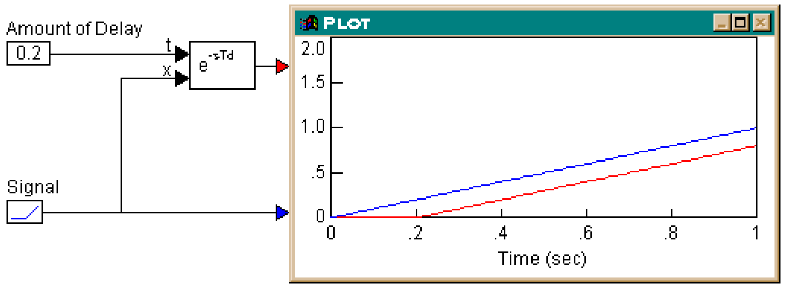
Max Buffer Size: Controls the granularity of the resulting timeDelay signal. If the signal is too granular, increase the value. The default is 128.

The `timeDelay` block requires a buffer element for each time step in the requested delay amount. The buffer size should be set to the maximum delay time you need divided by the simulation time step.

Examples

1. Introduction of a constant delay

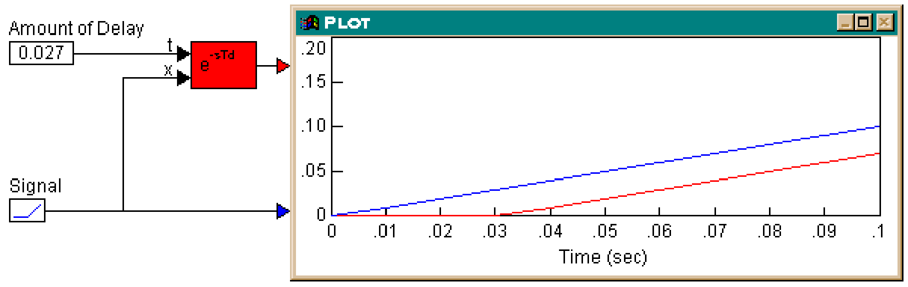
For a given signal, a constant delay can be introduced as:



Here, a `ramp` block is used to produce a test signal and a `const` block is used to produce a time delay of 0.2. For this example, the simulation step size is set to 0.01.

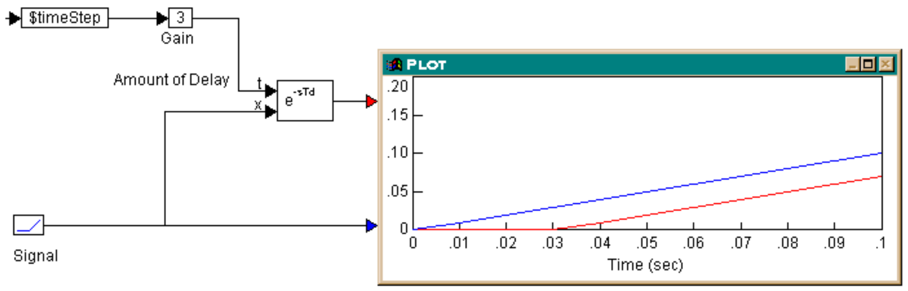
The amount of delay connected to the t input tab must be an integral multiple of the simulation step size. If you had entered 0.027 as the amount of delay and re-run the simulation, VisSim would have issued an error message. This error occurs because VisSim starts the simulation at *t* = 0, and steps through at intervals of 0.01. The time intervals that are hit are 0, 0.01, 0.02, 0.03, and so on. The time delay 0.027 is not honored.

If you choose to ignore the error (by clicking on the Retry or Ignore buttons in the message box), VisSim rounds off the delay to the next nearest time step, which in this case happens to be 0.03, as shown below:



2. Introduction of an integral-multiple delay

One way to achieve multi-step delays is by using the `timeDelay`, `$timeStep`, and `gain` blocks, as shown below:



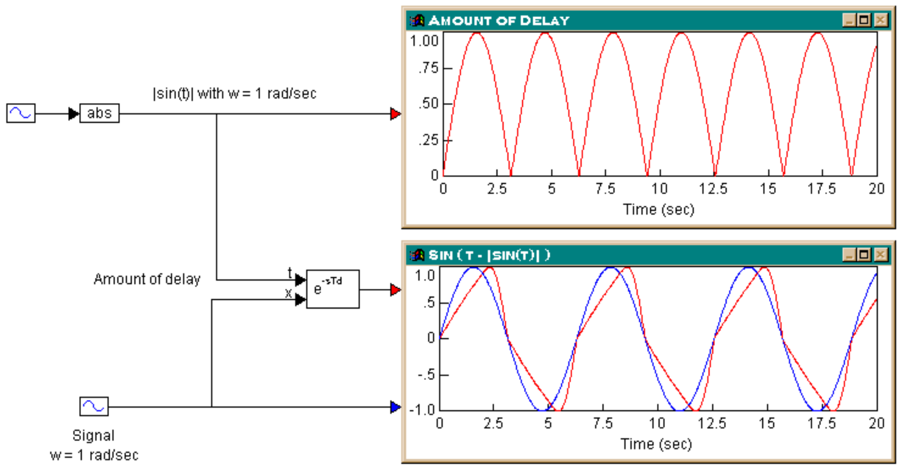
During simulation, the value sent to the t input of the `timeDelay` block is 3 * `$timeStep`, and as a result, the output of the `timeDelay` block is three steps behind the input signal.

3. Introduction of a time-varying delay

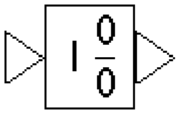
The real power of the `timeDelay` block becomes apparent when you implement time delays that are themselves time-varying. As an example, consider the following equation:

$y = \sin(t - |\sin(t)|)$

Here, the intent is to delay (or shift right) a sinusoid of frequency ω = 1 rad/sec, by a time-varying amount given by the absolute value |sin(*t*)|. This can be realized as:



An `abs` block computes the absolute value of sin(*t*), generated by a `sin` block. The output of the `abs` block is fed to the t input of the `timeDelay` block. Another `sin` block generates the actual signal to be delayed. The top `plot` block shows the time-varying delay being implemented. The bottom `plot` block shows the actual and delayed signals.



transferFunction

$$y = \frac{a_n s^n + a_{n-1} s^{n-1} \ldots a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} \ldots b_1 s + b_0} x$$

Block Category: Linear System

The `transferFunction` block executes a single-input single-output linear transfer function specified in the following ways:

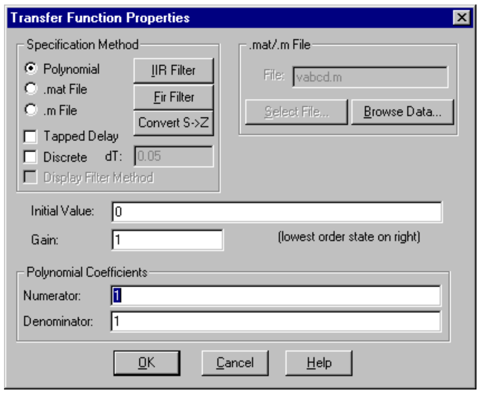
- **As an .M file created with VisSim:** The Linearize command in the Analyze menu generates ABCD state-space matrices from the nonlinear system by numerically evaluating the matrix perturbation equations at the time the simulation was halted. For more information, see the *VisSim/Analyze User's Guide*.
- **As an .M file created with a text editor:** The following is an example of a user-written .M file:

  ```
  function [a,b,c,d] =vabcd
  a = [-.396175 -1.17336 ; 5.39707 .145023 ];
  b = [-.331182 ; -1.08363 ];
  c = [0 1 ];
  d = [0 ];
  ```
- **As a .MAT file created with MATLAB:** Generating .MAT files is described in the MatLab documentation. Note that when you save the ABCD matrices to file, the names of the matrices are not important; however, the order in which they appear is.

When you simulate the block diagram, VisSim numerically solves the `transferFunction` block.

Digital filter design: The `transferFunction` block supports IIR and FIR digital filter design. For more information, see Chapter 9, "Designing Digital Filters."

Setting up a transfer function: The `transferFunction` block's Properties dialog box allows you to control how the numerator and denominator polynomials are entered.



Specification Method
(● Polynomial) ( ) .mat File ( ) .m File
[ ] Tapped Delay
[ ] Discrete dT: 0.05
[ ] Display Filter Method

.mat/.m File
File: vabcd.m

Initial Value: 0
Gain: 1
(lowest order state on right)

Polynomial Coefficients
Numerator: 1
Denominator: 1

Specification Method: You have three choices of specification method:

- **Polynomial Coefficient:** Indicates that the transfer function is to be specified as numerator and denominator polynomials. Supply the numerator and denominator polynomials and gain under the Polynomial Coefficients group box.
- **.mat File:** Indicates that the transfer function is to be specified as a .MAT file. Specify the name of the .MAT file in the .mat/.m File group box.
- **.m File:** Indicates that the transfer function is to be specified as an .M file. Specify the name of the .M file in the .mat/.m File group box.

Discrete: Indicates a discrete Z-Domain transfer function. Enter the time step for the discrete transfer function in the dT box. By default, VisSim uses the step size established with the Simulate > Simulation Properties command.

When Discrete is de-activated, a continuous transfer function is created.

Tapped Delay: Provides tapped delay implementation for high order FIR filters. For more information, see Chapter 9, "Designing Digital Filters."

dT: Specifies the time step for the discrete transfer function. By default, VisSim uses step size parameter from the Simulate > Simulation Properties command.

Display Filter Method: Displays the filter specification on the block. When Display Filter Method is not activated, VisSim displays the polynomial coefficients.

IIR Filter: Opens the IIR Filters Setup dialog box to design a suitable filter using analog prototypes. For more information, see Chapter 9, "Designing Digital Filters."

FIR Filter: Opens the FIR Filter Setup dialog box to construct Regular Finite Impulse Response filters, differentiators, and Hilbert Transformers. For more information, see Chapter 9, "Designing Digital Filters."

Convert S ->Z: Uses bilinear transformation to convert a continuous transfer function to an equivalent discrete transfer function with a sampling interval of *dT*. VisSim requests a discrete sampling rate prior to performing the conversion.

An example of the conversion is shown below.

$$H(s) = \frac{a}{s+a}$$

The bilinear transformation can be implemented by the substitution:

$$\frac{2}{dT}\frac{z-1}{z+1} \rightarrow s$$

The above transfer function becomes:

$$H_{dT}(z) = \frac{a}{\left(\frac{2}{T}\right)\left\{\frac{z-1}{z+1}\right\}+a}$$

VisSim automatically simplifies this representation and enters the appropriate coefficients for the numerator and denominator polynomials.

Convert Z ->S: Uses bilinear transformation to convert a discrete transfer function to an equivalent continuous transfer function. For example, consider:

$$H_{dT}(z) = \frac{z}{z+b}$$

The bilinear transformation can be implemented by the substitution:

$$\frac{2+dT.s}{2-dT.s} \rightarrow z$$

The above discrete transfer function becomes:

$$H_{dT}(s) = \frac{2+dT.s}{(2+2b)+(dT-b.dT)s}$$

VisSim automatically simplifies this representation and enters the appropriate coefficients for the numerator and denominator polynomials.

It is important to note that in both transformations, the results obtained are dependent on the sampling interval *dT*. In other words, for a given continuous or discrete transfer function, an infinite number of equivalent discrete or continuous transfer functions may be obtained by varying the sampling interval *dT*.

File: Indicates the name of the .M or .MAT file to be used as input to the `transferFunction` block. You can type the file name directly into this box or select one using the Select File button.

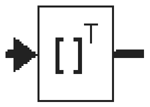
Initial Value: Specifies initial values for the states in the block. The values are right-adjusted. The right-most value corresponds to the lowest order state. Unspecified states are set to 0.

Gain: Indicates the transfer function gain. If the leading terms of the numerator and denominator coefficients are not unity, VisSim will adjust the gain to make it so. The default value is 1.

Denominator: Indicates the denominator polynomial for the `transferFunction` block. VisSim determines the order of the transfer function by the number of denominator coefficients you enter. For example, an nth order transfer function will have n + 1 coefficients. Separate coefficients with spaces.

Numerator: Indicates the numerator polynomial for the `transferFunction` block. Separate coefficients with spaces.

transpose



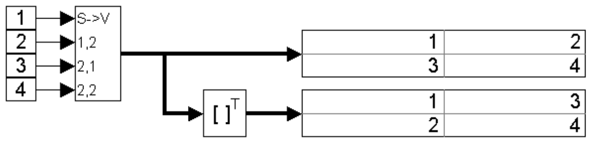
$$\left[a_{ij}\right]^T = \left[a_{ji}\right]$$

Block Category: Matrix Operation

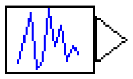
The `transpose` block interchanges each row with the column of the same index number. Thus, if $\mathbf{A} = \left[a_{ij}\right]$, then the transpose of $\mathbf{A}$ is: $\mathbf{A}^T = \left[a_{ji}\right]$

The `transpose` block accepts one vector input and produces one vector output.

Examples

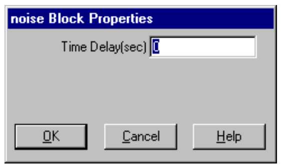


uniform



Block Category: Random Generator

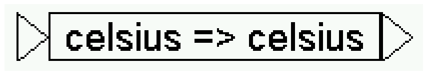
The `uniform` block creates a uniformly distributed random noise signal with values between zero and one. The random seed is set under Preferences in the dialog box for the Simulate > Simulation Properties command.



Time Delay(sec) 0

Time Delay(sec): Indicates, in seconds, how long to delay before calculating the value of the noise signal. The default is 0.

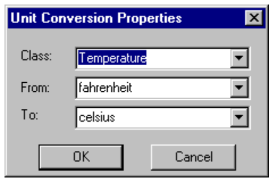
unitConversion



Block Category: Arithmetic

The `unitConversion` block changes the unit of measurement of the data. You can convert the unit of measurement within numerous categories, including: acceleration, area, capacitance, charge, conductivity, current, energy, flow rate, force, inductance, magnetic flux, mass, position, power, pressure, speed, temperature, volume, and more. For example, you can convert from Fahrenheit to Celsius, watts to kilowatts, or joules to BTUs.

Conversions are always displayed on the block.



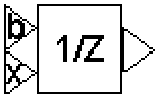
Class: Temperature
From: fahrenheit
To: celsius

Class: Indicates the category of measurement.

From: Indicates the unit of measurement for the data exiting the block. Click on the DOWN ARROW to select a unit of measurement.

To: Indicates the unit of measurement for the data entering the block. Click on the DOWN ARROW to select a unit of measurement.

unitDelay

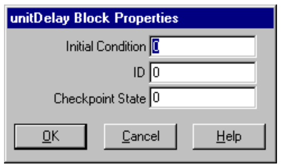


$$y = \begin{cases} y_{\text{buffer}}, y_{\text{buffer}} = x_2 & \text{if } |x_1| \geq 1 \\ y_{\text{previous}} & \text{otherwise} \end{cases}$$

Block Category: Time Delay

The `unitDelay` block specifies a clocked unit delay. The input connector tabs are marked b (for Boolean clock) and x (for main signal). When the Boolean clock does not equal zero, the value contained in the single element buffer is copied to the block output (where it holds this value until the next non-zero Boolean clock). The current value of the main signal is stored in the unit buffer.

The `unitDelay` block is intended for modeling a digital delay in a continuous simulation. A typical digital delay is modeled by wiring a `pulseTrain` block to the Boolean input connector tab of the `unitDelay` block. Use the `timeDelay` block to model a continuous delay.


unitDelay Block Properties

Initial Condition 0
ID 0
Checkpoint State 0

Initial Condition: Sets an initial value for the output signal. The default is 0.

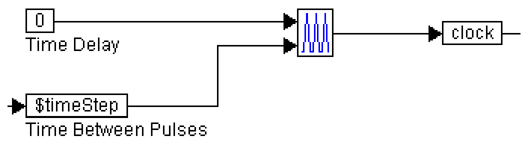
ID: Reserved for future use.

Checkpoint State: Contains the value of the unit delay at the checkpoint. If you have not checkpointed your simulation via the Simulate > Simulation Properties command, the value is 0.

Examples

1. Clocking the `unitDelay` block

If you are working with `unitDelay` blocks, it is good programming practice to create a clock signal that you can use in every simulation. A typical clock signal can be generated as:



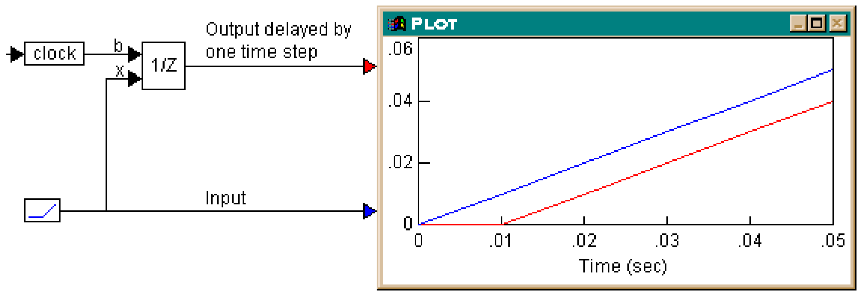
Here, a `pulseTrain` block is assigned two external inputs:

- The top input is the time delay for the `pulseTrain` block. The time delay value for the `pulseTrain` block is the amount of time the `pulseTrain` block waits before producing pulses. This time delay value must not be confused with the amount of time delay generated by the `unitDelay` block.
- The bottom input is the time between pulses.

The output of the `pulseTrain` block is fed to the `variable` *clock*. This variable can be used anywhere in the simulation to clock `unitDelay` blocks.

2. Introduction of a one-step delay

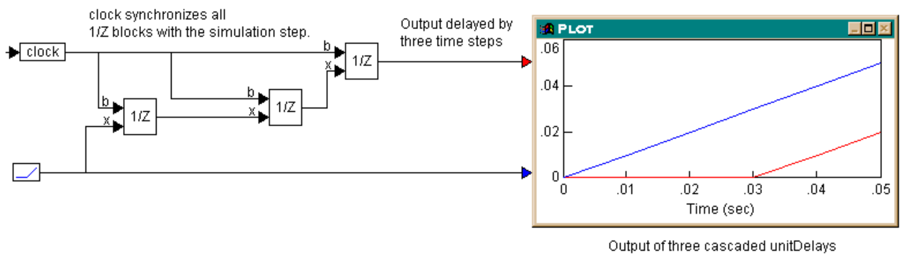
For a given signal, a one-step delay can be introduced as:



During simulation, the actual and delayed signals are plotted in the `plot` block. The output of the `unitDelay` block is delayed by one step (equal to 0.01 in this case) as compared to the input.

3. Using a multi-step delay with cascaded `unitDelay` blocks

To achieve multi-step delays, `unitDelay` blocks that implement one-step delays, can be cascaded. Consider the example where a three-step delay is introduced:



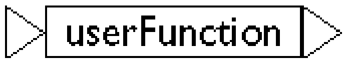
Three `unitDelay` blocks, all clocked at the simulation step, are cascaded. Since each `unitDelay` introduces a one-step delay between its input and output, the output of the third `unitDelay` block is delayed by three steps compared to the input. The `plot` block shows this behavior, with a simulation step size of 0.01.

unknown

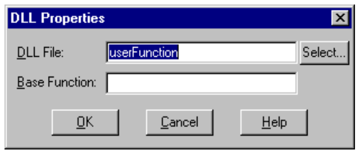
Block Category: Optimization

The `unknown` block works in conjunction with `constraint` blocks to solve equations for unknowns using Newton-Raphson iteration. For each `unknown`, there should be a `constraint` block that is fed directly or indirectly by the `unknown`. The maximum iteration count, error tolerance, and perturbation are established under the Implicit Solver tab in the dialog box for the Simulate > Simulation Properties command. For more information, see Chapter 7, “Solving Implicit Equations.”

userFunction



The `userFunction` block lets you create blocks bound to Dynamic Link Library (DLL) functions. For more information, see Appendix B, “Extending the Block Set.”



DLL File: userFunction

Base Function:

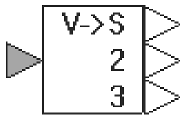
DLL File: Indicates the name of the DLL file containing the user function.

Base Function: Indicates the base name of the function.

variable

Block Category: Annotation

The `variable` block lets you name a signal and transmit it throughout your diagram without the use of wires. For more information, see page 129.



vecToScalar

Block Category: Annotation

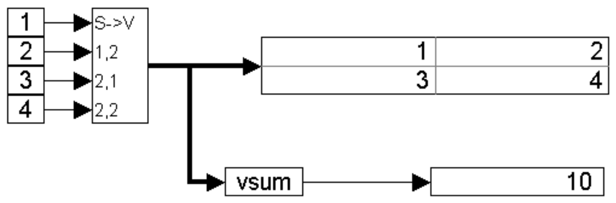
The `vecToScalar` block separates a single vector wire into individual output signals. Use the `scalarToVec` block to bundle signals into a single vector wire.

vsum

Block Category: Matrix Operation

The `vsum` block produces a single value summation of all the elements in the matrix. The `vsum` block accepts one vector input and produces one scalar output.

Examples



wirePositioner

Block Category: Annotation

The `wirePositioner` block lets you create a specific wiring path. A `wirePositioner` block is essentially an input connector tab and an output connector tab that are attached by a flexWire. Since `wirePositioner` blocks don't take any additional computation time, you won't see a decrease in performance during a simulation.

Input to the `wirePositioner` block can be scalar or vector.

xor

$y = x_1 \text{ bitwise XOR } x_2$

Block Category: Boolean

The `xor` block produces the bitwise exclusive OR of two to 256 scalar input signals.

If you click the right mouse button over the `xor` block, the Boolean block menu appears allowing you to assign a different function to the block.

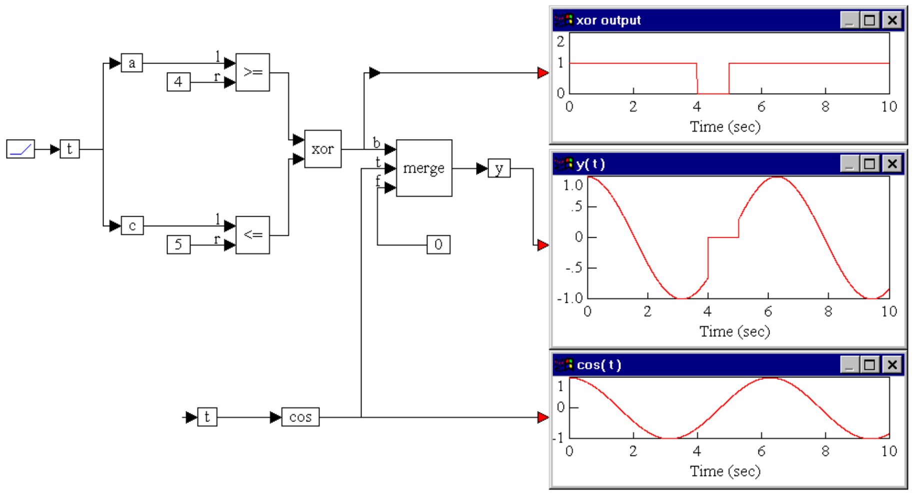
Examples

1. Using the `xor` block

Consider a variable *y* such that:

If $a \geq 4$ or $c \leq 5$, then $y = \cos(t)$; else $y = 0$. Also, if $a \geq 4$ and $c \leq 5$, y = 0

where *t* is simulation time. Furthermore, let *t* be the input to parameters *a* and *c*. This system can be realized as:



As shown in the two `plot` blocks, the output of the `xor` block evaluates to false in the interval *t* = (4, 5), since both the inputs to the `xor` block are true in this interval. Consequently, *y* takes on the value of 0. The output of `xor` evaluates to true in the remaining parts of the simulation, and as a result, *y* takes on the value of cos(*t*) in these periods.

Appendix A

Customizing VisSim

This chapter covers the following information:

- Customizing VisSim start-up
- Customizing the VisSim window
- Creating custom implicit solvers
- Creating custom global optimizers

Customizing VisSim start-up

By adding arguments to the VisSim start up command, you can control such things as how VisSim starts up, the block diagram file opened at start up, and whether VisSim immediately simulates the opened diagram.

▶ **To customize VisSim start-up**

1. Refer to your Windows documentation for instructions on displaying the properties of a program.
2. Under Windows 3.1+, enter one or more of the following arguments in the Command Line box. Under Windows 95 and NT, click on the Shortcuts tab and enter one or more of the following arguments in the Target box.

If you enter more than one argument, separate them with spaces. If a block diagram name is included in the argument list, it must be specified last.

Use this argument	To
block-diagram-name	Start VisSim and open the specified block diagram.
-i [*block-diagram-name*]	Start VisSim as an icon and optionally open a block diagram.
-nb [*block-diagram-name*]	Start VisSim without the start-up banner and optionally open a block diagram.
-ne	Suppress the simulation completion dialog box.
-r *block-diagram-name*	Run a simulation read in from the specified block diagram upon start up.
-re *block-diagram-name*	Run a simulation read in from the specified block diagram and exit VisSim upon completion.

5. Click on the OK button.

Customizing the VisSim window

The VisSim window contains a menu bar, toolbars, scroll bars, a status bar, and a diagram tree. You can display or hide these items at any time during a VisSim session. For example, if you're working on a large a block diagram, you may want to hide the status bar and diagram tree so you can see as much of the diagram as possible. When you simulate the diagram, you may want to display the status bar to keep track of progress of the simulation.

▶ **To hide or display VisSim window elements**

- Do one or more of the following:

To display or hide	Do this
Scroll bars	Choose Edit > Preferences. Select the Preferences tab, and select or clear the Show Horizontal Scroll Bar or Show Vertical Scroll Bar check box.
Status bar	Choose View > Status Bar.
Toolbar	Choose View > Toolbar. Select or clear each toolbar check box.
Diagram tree	Drag the right edge of the diagram tree.

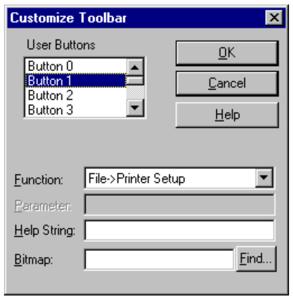
Customizing the toolbar

You can create a custom toolbar that contains buttons for commands and blocks you use most frequently. You can also create your own images for the buttons. The custom toolbar is named User.

▶ **To create a custom toolbar button**

1. Choose View > Toolbar.
2. Activate the User option, if it is not already activated, then click on the OK button, or press ENTER.
3. Choose Edit > Toolbar.

   The Customize Toolbar dialog box appears.



Function: File->Printer Setup
Parameter:
Help String:
Bitmap:

4. Under User Buttons, select a button number.
5. In the Function box, click on the DOWN ARROW and select the command to be assigned a toolbar button. If you select:

   - Blocks→ (P), click on the DOWN ARROW in the Parameter box and select a block to be associated with the toolbar button.
   - Edit→Find (P), File→Add (P), or File→Open (P), you can optionally enter a variable block name or file name in the Parameter box. If you do not enter anything, VisSim opens the Find, File Add, or File Open dialog box when you click on the toolbar button.

6. In the Help String box, you can optionally enter text that will appear in the status bar when you point to the toolbar button. If you do not enter anything, VisSim displays the default help string for the toolbar button.
7. Click on the Find button to choose a bitmap for the toolbar button. Pre-made button bitmap images can be found in \VISSIM30\BITMAPS\TOOLBAR.

   Custom bitmap should be 16-pixels wide by 15-pixels high for the best display.

8. Click on the OK button, or press ENTER.

▶ **To remove a custom toolbar button**

1. Choose Edit > Toolbar.
2. From the toolbar button list, select the toolbar number that corresponds to the toolbar button to be removed from the toolbar.
3. In the Functions box, click on the DOWN ARROW and select <none>.
4. Click on the OK button, or press ENTER.

Customizing other screen settings

In addition to changing general VisSim settings, you can also change many other settings to customize how VisSim looks, such as the use of colors and text fonts in your diagrams, the shape and color of connector tabs, and the amount of information displayed with each block in a diagram.

These settings are controlled with the Edit > Preferences command and the commands under the View menu. When you select a setting, it takes effect immediately and remains in effect until you change it.

▶ **To change the display settings in VisSim**

- Do one or more of the following:

To	**Do this**
Hide input connector tabs and shrink the size of output connector tabs	Choose View > Presentation Mode.
Hide wires and connector tabs, freeze blocks in place, and with the exception of interactive elements on `button` and `slider` blocks, lock block parameter values	Choose View > Display Mode. Typically this mode is used when there is animation in your simulation or you've constructed an instrumentation panel to monitor and control your simulation. Display mode can be turned on or off for individual block diagram levels.
Color connector tabs according to the type of data entering or exiting the block	Choose View > Data Types. The four types of data and their corresponding connector tab colors are double-floating point (red), signed integer (green), unsigned integer (blue), and vector (magenta).

To	Do this
Display connector labels on compound blocks	Choose View > Connector Labels.
Change how text is displayed on blocks	Choose View > Fonts. Select the text attributes. Character format can be selectively applied to `label` blocks, as described on page 198. Rich text format can be retained in `comment` blocks, as described below.
Retain character format in `comment` blocks	Choose Edit > Preferences. Select the Preferences tab. Activate Use Rich Text Format.
Change the color of the VisSim screen; plotting background on `plot`, `stripChart`, and `histogram` blocks; wires; and diagram text	Choose View > Colors. Select the color for the corresponding screen element. When you choose a default color for the plotting background, VisSim uses the specified color on all `meter`, `plot`, and `stripChart` blocks except those whose background colors were explicitly set in their Properties dialog boxes.
Display block names beneath each block	Choose Edit > Preferences. Select the Preferences tab. Activate Training Mode Labels. Then choose View > Block Labels.
Display parameter values, file names, and block names beneath each block	Choose Edit > Preferences. Select the Preferences tab. Clear Training Mode Labels. Then choose View > Block Labels.
Color compound blocks light blue	Choose Edit > Preferences. Select the Preferences tab. Activate Color Compound Blocks.
Display block diagrams in black and white	Choose Edit > Preferences. Select the Preferences tab. Clear Color Compound Blocks and Color Display.

Creating custom implicit solvers

You can write an implicit static solver as a .DLL file. VisSim recognizes and uses a user-written solver only if:

- It is named VSOLVER.DLL
- It resides in your current directory when you initiate implicit static solving

- It contains an exported function called userSolver()
- The User Solver parameter in the dialog box for the Simulate > Optimization Properties command is activated

Source files for building a custom implicit solver

The following table lists the source files for building an implicit static solver. These files are installed in \VISSIM30\VSOLVER and \VISSIM30\VSDK. They contain code for building a simplified Gauss-Seidel static solver. You may find it easier to edit the files to create your own static solver. To use these files, they must remain in the directories in which they currently reside.

Source file	**Description**
VSOLVER.FOR or VSOLVER.C	A Fortran or C source file for the implicit static solver. The heart of the solver is the vissimRequest() function that you call to obtain the inputs to the `constraint` blocks and to supply values to the outputs of the `unknown` blocks. Using vissimRequest(), you can write a wide variety of solvers. For more information, see the description below.
VSOLVER.DEF	A definition file with linker commands to build a .DLL file from object code. Windows requires that you use a definition file to link the object code.
VSOLVER.MAK	A make file with rules for automatically building a .DLL file.
VSUSER.H	A C language header file with function prototypes and command definitions for the vissimRequest() function.

Using vissimRequest() in a custom implicit solver

The vissimRequest() function is a general-purpose function for making requests to VisSim. A user-written solver uses vissimRequest() to read and write optimization information in a block diagram. The general format of vissimRequest() is:

```
long FAR vissimRequest(long req, long arg2, long arg3 )
```

The first argument (long req) is a message code describing the action for VisSim to take. The list of message codes is defined in the file named VSUSER.H, which is installed in \VISSIM30\VSOLVER. The message codes that pertain to writing a local static solver are listed in the table on the next page.

Message code	Description
VR_EXECUTE	Executes the diagram on iteration without moving time.
VR_GET_BLOCK_PARAMS	Returns a pointer to a block’s parameters.
VR_GET_CONSTRAINTS	Arg2 returns a vector of local constraint values. Ordering of the elements vector can be determined by the value of the ID parameter for the `constraint` block. VisSim sorts in sequential order, from low to high.
VR_GET_SOLVER_INFO	Arg2 returns information related to the diagram and the implicit solver dialog settings in the following manner: arg2[0] = number of constraints arg2[1] = number of unknowns arg2[2] = relaxation value arg2[3] = maximum iteration value arg2[4] = error tolerance value
VR_GET_UNKNOWNS	Arg2 returns a vector of current local unknown output values. Ordering of the elements vector can be determined by the value of the ID parameter for the `unknown` block. VisSim sorts in sequential order, from low to high.
VR_GET_UNKNOWNS_INPUT	Arg2 returns a vector of current inputs to the `unknown` blocks. Ordering of the elements vector can be determined by the value of the ID parameter for the `unknown` block. VisSim sorts in sequential order, from low to high. (This is useful for initial condition setting.)
VR_GET_VERSION	Returns the current version of VisSim.
VR_GET_VISSIM_STATE	Gets information related to the global state of VisSim. The information provided is a copy of the current internal state; modifying it will not change VisSim’s state. Arg2 should contain a pointer to a SIM_INFO structure, defined in VSUSER.H, which will be filled in by the vissimRequest() function. Arg3 should contain the size of this structure (sizeof(SIM_INFO)) to allow for version compatibility checking.
VR_SET_UNKNOWNS	Sets diagram unknowns based on the vector passed as arg2. Ordering of the elements vector can be determined by the value of the ID parameter for the `unknown` block. VisSim sorts in sequential order, from low to high.

Building a custom implicit solver

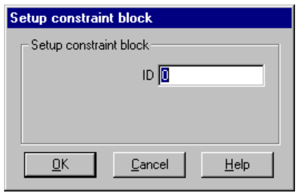
Most languages have a Project Build facility that automates the process of building a .DLL file. The following procedure guides you through the process of building a project in general terms. Refer to the documentation for the application language you're using for specific instructions.

▶ **To build a custom implicit solver**

1. Invoke the Compiler environment.
2. Add all the source files listed in on page 268 to the project or make file.
3. Under project options, specify the project type as a Windows Dynamic Link Library (.DLL).
4. Under compiler preprocessor options, specify \VISSIM30\VSDK as the include directory.
5. Build the project.

Using the constraint block with a custom implicit solver

To indicate the number that VisSim uses to sort the block when presented as a vector in a user-written solver, enter it in the ID box of the Constraint Properties dialog box. VisSim does not require the ID to be unique or contiguous; it sorts them in sequential order. The default is 0.


Setup constraint block

Setup constraint block

ID 0

Creating custom global optimizers

You can write a global optimizer as a .DLL file. VisSim recognizes a user-written global optimizer when it is named VOPT.DLL and resides in your current directory. VOPT.DLL should also contain an exported function in the following format:

```
int FAR EXPORT USER_OPT_FUNC(DOUBLE *unknownVec, int
   unknownCount, int costCount, int globalConstraintCount);
```

Optimize has a prototype declared in VSUSER.H.

Before you initiate global optimization, make sure VOPT.DLL is in your current directory and the User Solver parameter in the dialog box for the Simulate menu's Optimization Setup command is activated.

Source files for building a custom global optimizer

The following table lists the source files for building a global optimizer. These files are installed in \VISSIM30\VSOLVER and \VISSIM30\VSDK.

Source file	Description
VOPT.C	A C source file for a sample global optimizer. The heart of the optimizer is the vissimRequest() function that you call to obtain the inputs to the `cost` blocks and to supply values to the outputs of the `parameterUnknown` blocks. Using vissimRequest(), you can write a wide variety of optimization algorithms. For more information, see the description below.
VOPT.DEF	A definition file that contains linker commands to build a .DLL file from object code.
VOPT.MAK	A make file that contains rules for automatically building a .DLL file.
VSUSER.H	A C language header file that contains function prototypes and command definitions for the vissimRequest() call.
IMPSIM.LIB	VisSim import library that describes the address of vissimRequest().

Using vissimRequest() in a custom global optimizer

The vissimRequest() function is a general-purpose function for making requests to VisSim. A user-written global optimizer uses vissimRequest() to read and write global optimization information in a block diagram. The general format of vissimRequest() is:

```
long FAR vissimRequest(long req, long arg2, long arg3 )
```

The first argument (long req) is a message code describing the action for VisSim to take. The list of message codes is defined in the file named VSUSER.H, which is installed in \VISSIM30\VSOLVER. The message codes that pertain to writing a global optimizer are as follows:

Message code	Description
VR_GET_GLOBAL_COST	Writes a vector of current `cost` block input values into a vector pointed at by arg2.
VR_GET_GLOBAL_CONSTRAINTS	Writes a vector of current `globalConstraint` block input values into a vector pointed at by arg2.
VR_GET_GLOBAL_CONSTRAINT_BOUNDS	Writes a vector of `globalConstraint` block low bounds into a vector pointed at by arg2, and a vector of `globalConstraint` block high bounds into a vector pointed at by arg3.

Message code	Description
VR_GET_GLOBAL_OPT_INFO	Gets information related to the global optimization settings in the dialog box for the Optimization Setup command. The information provided is a copy of the current optimization state; modifying it will not change VisSim's state. Arg2 should contain a pointer to an OPT_INFO structure, defined in VSUSER.H, which will be filled in by the vissimRequest() call. Arg3 should contain the size of this structure (sizeof(OPT_INFO)) to allow for version compatibility checking.
VR_GET_GLOBAL_UNKNOWNS	Writes a vector of current `parameterUnknown` block output values into the vector pointed at by arg2. Ordering of the elements vector can be determined by the value of the ID parameter for the `parameterUnknown` block. VisSim sorts in sequential order, from low to high.
VR_GET_GLOBAL_UNKNOWNS_INPUT	Writes a vector of current `parameterUnknown` block input values into the vector pointed at by arg2. Ordering of the elements vector can be determined by the value of the ID parameter for the `parameterUnknown` block. VisSim sorts in sequential order, from low to high.
VR_GET_GLOBAL_UNKNOWN_BOUNDS	Writes a vector of `parameterUnknown` block low bounds into a vector pointed at by arg2, and a vector of `parameterUnknown` block high bounds into a vector pointed at by arg3.
VR_GET_VERSION	Returns the current version of VisSim.
VR_GET_VISSIM_STATE	Gets information related to the global state of VisSim. The information provided is a copy of the current internal state; modifying it will not change VisSim's state. Arg2 should contain a pointer to a SIM_INFO structure, defined in VSUSER.H, which will be filled in by the vissimRequest() function. Arg3 should contain the size of this structure (sizeof(SIM_INFO)) to allow for version compatibility checking.
VR_RESET_XFERS	For internal use only.
VR_RUN_SIMULATION	Starts a simulation run.
VR_SET_GLOBAL_UNKNOWNS	Sets current `parameterUnknown` block output values from arg2.

Building a custom global optimizer

Most languages have a Project Build facility that automates the process of building a .DLL file. The following procedure guides you through the process of building a project in general terms. Refer to the documentation for the application language you're using for specific instructions.

▶ **To build a custom global optimizer**

1. Invoke the Compiler environment.
2. Add all the source files listed in on page 271 to the project or make file.
3. Under project options, specify the project type as a Windows DLL.
4. Under compiler options, specify the following:
   - Memory Model to be Large.
   - Windows Prolog/Epilog to be Real Mode_far Functions.
5. Build the project.

Appendix B

Extending the Block Set

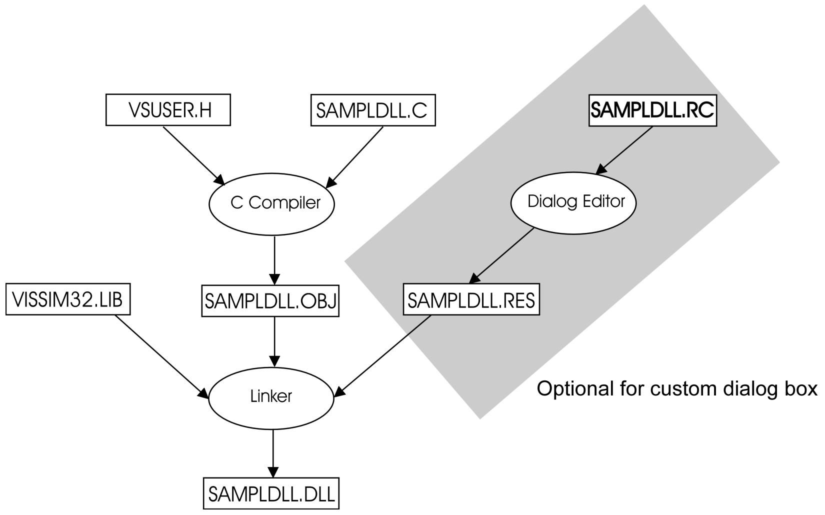
This appendix covers the following information:

- Writing DLLs
- Building DLLs
- Calling conventions
- Simulation level functions
- Block level functions
- Exported functions
- Debugging DLLs
- Binding DLLs to `userFunction` blocks
- Adding user-written blocks to the Blocks menu

The big picture

VisSim provides an Application Programming Interface (API) that allows you to extend the standard block set by creating Dynamic Link Library (DLL) files and binding them to `userFunction` blocks. A DLL file is like a regular executable file with the exception that it cannot start execution on its own. A DLL function can be called just like functions that are part of a normal executable file.

The following diagram shows how files are processed to create VisSim DLLs. This diagram steps you through the process of creating a DLL from a C source file; however, you can also create DLLs in Fortran and Pascal.



The main steps in the creation of VisSim DLLs are:

1. Create or edit an existing C, Fortran, or Pascal source file.
2. Create a project DLL for your compiler.
3. Execute a build operation, which compiles your source code into an object file.
4. Link the object file with VISSIM32.LIB to produce a DLL.

Criteria for writing DLLs

You can write DLLs in any language, provided the language has the following capabilities:

- 64-bit floating point array parameters
- Pointers to 16-bit integers
- `_stdcall` calling conventions (default for Microsoft Fortran and Delphi Pascal)

Example DLLs written in C, Fortran, and Pascal are distributed with VisSim and reside in subdirectories under the \VISSIM30\VSDK directory.

Building a DLL

Most languages have a Project Build facility that automates that process of building a DLL. The following procedure guides you through the process of building a project in general terms. Refer to the documentation for the application language you’re using for specific instructions.

1. Invoke the Compiler environment.

2. Add the following files to the project file:
   - All the source files
   - All the resource files
   - VISSIM32.LIB
3. Under project options, specify the project type as a Windows Dynamic Link Library (.DLL).
4. Under compiler preprocessor options, specify \VISSIM30\VSDK as the include directory.
5. Build the project.

How VisSim talks to a DLL

There are three types of functions used for communication between VisSim and your DLL:

- Simulation level functions
- Block level functions
- VisSim exported functions

Simulation level functions and block level functions are DLL functions VisSim can call via a `userFunction` block. Simulation level functions are called once per simulation-wide event. Block level functions, on the other hand, are called once for each `userFunction` block. If you have no `userFunction` blocks, no calls are made; if you have 100 `userFunction` blocks, they are called 100 times.

Exported functions are VisSim functions that you can call from a DLL. These functions allow a DLL to request information from VisSim, as well as instruct VisSim to perform specific actions.

Calling conventions

The following information pertains to the DLL functions described in the next several sections:

- Functions are shown in C syntax. For Fortran and Pascal syntax, look in the sample source files in subdirectories under \VISSIM30\VSDK.
- All arguments are pointers to data types. Since Fortran passes variables by reference, a normally declared Fortran variable can be passed as an argument.

Simulation level functions

There are two simulation level functions:

- **vsmInit().** Called at VisSim start-up. You use this function to insert one or more blocks in the Blocks menu.
- **vsmEvent().** Called at simulation-wide events, such as simulation start, end of time step, and simulation end.

Initialization function - vsmInit()

This DLL function is called by VisSim to allow the DLL to perform initialization, particularly to insert `userFunction` blocks into the Blocks menu.

```
EXPORT32 int EXPORT PASCAL vsmInit()
```

To have vsmInit() be called when VisSim starts up, you must tell VisSim the path to your DLL, as described on page 288.

System-wide event function - vsmEvent()

This function is called by VisSim to notify the DLL of interesting simulation-wide events.

```
LPSTR PASCAL vsmEvent(int msg, int wParam, long *arg);
```

Message	Function
VSE_POST_SIM_END	VisSim has stopped, either from user stop or time expiration.
VSE_POST_SIM_START	VisSim has initialized all blocks and is about to start.
VSE_PRE_SIM_START	VisSim is about to start a simulation but has not initialized any blocks.
VSE_SIM_RESTART	VisSim is restarting due to auto-restart.
VSE_TIME_STEP	Simulation has completed a time step.
WM_COMMNOTIFY	Comm port data is ready.
WM_DESTROY	VisSim is exiting.
WM_VSM_WINDOW_HANDLE	Handle to VisSim main window. **arg:** Window handle

Block level functions

In order to interface smoothly with VisSim, VisSim can call seven Pascal-style functions that share the base DLL function name and have an event code suffix corresponding to a VisSim event. You should supply a function for each event that you want your DLL code to handle. The additional functions are described below:

Function name	**Purpose**	**When it's called**
userBlock()	Block time step	Each simulation time step
*userBlock*Event()	Block event handler	On occurrence of a block related event
userBlock PA()	Block parameter allocation	Block creation
userBlock PC()	Block parameter change	Right button click
userBlock PI()	Block parameter initialization	Immediately after *userBlock*PA()
userBlock SE()	Block simulation end	Simulation end time
userBlock SS()	Block simulation start	Simulation start time

The term *userBlock* is a placeholder for your DLL base function name. You specify the DLL base function name when you bind the DLL to a `userFunction` block. For more information, see page 287.

You can have any number of user-written blocks in a single DLL file.

All definitions are kept in the VSUSER.H file. This file should be included in every user DLL.

Time step function - *userBlock*()

The *userBlock*() function is called at each time step to calculate simulation values. It is the only function a DLL is required to export.

The *userBlock*() function may be called an arbitrary number of times during a time step interval. The number of calls depends on the integration method and whether the VBF_HAS_STATE flag is active for a block. Note that the outputs are not preserved from call to call. Therefore this function must write to its outputs at each call.

```
void PASCAL userBlock(double param[], double inSig[], double
    outSig[])
```

The inSig array is filled by VisSim with the values presented to the input connector tabs on the `userFunction` block. Store the result values in the outSig array. VisSim presents the outSig values to the corresponding output connector tabs on the `userFunction` block.

Event handler function - *userBlock*Event()

The *userBlock*Event() function is called at block events, such as block repaint, end of time step, and simulation end.

```
LPSTR PASCAL userBlockEvent(HWND h, int msg, WPARAM wp,
   LPARAM lp)
```

This function lets you save and restore mixed data types. Note that the arguments are the same as Windows or Windows NT event functions.

VisSim calls your function with the following messages:

Message	Description
WM_VSM_ADD_CONNECTOR	Signals a user-request for a connector to be added to a block. **wp:** Current count **lp:** 1 if input 0 if output
WM_VSM_BLOCK_INFO	Used for writing a custom optimizer.
WM_VSM_BLOCK_PLACED	Signals that a block has been placed in the diagram. lp: Block handle
WM_VSM_BLOCK_SETUP	This event is generated only if there is no *userBlock*PC() function defined for the block. This event is generated when you click the right mouse button on the block. **wp:** Internal ID **lp:** Block handle
WM_VSM_CHECKPOINT_STATE	Signals a user-request to checkpoint system states; that is, set the checkpoint buffer to the current state value.
WM_VSM_CONNECTOR_NAME	Returns string of connector label name. Null if no label is desired. **wp:** Port number (negative if output) **lp:** Block handle

Message	Description
WM_VSM_CREATE	Is called when a user block is created. You can return flags to customize block treatment. (For more information, see “Return flags for WM_VSM_CREATE,” below.) **wp:** 0 if create from file 1 if create from menu 2 if create from clipboard **lp:** Block handle
WM_VSM_DEL_CONNECTOR	Signals a user-request for a connector to be deleted from a block. **wp:** Current count **lp:** 1 if input 0 if output
WM_VSM_DESTROY	Is called when user block is destroyed. **lp:** Block handle
WM_VSM_FILE_CLOSE	Reserved.
WM_VSM_FILE_READ	Signals that block and all its parameters have been read in from file. **lp:** Block handle
WM_VSM_GET_BLOCK_BITMAP	Reserved.
WM_VSM_GET_BLOCK_NAME	Provides a block with a custom name. Return a null terminated string that contains the new name. The name may contain newline characters.
WM_VSM_GET_PARAM_DESC	Provides a data descriptor for saving and restoring data. Return a text string that describes your data by following these guidelines:

Format character	Data Type
i	2-byte integer
I	4-byte integer
f	4-byte float
F	8-byte float
c	Single-byte character

All of the above format characters can take an optional count suffix, which is enclosed in square brackets. For example to save two 8-byte floats and a 32-character string in C, use the following string notation:

“F [2] c[32]”

In Fortran, the string notation is:

‘F [2] c[32]’ c

Message	Description
WM_VSM_INFO	Reserved.
WM_VSM_RETAIN_STATE_ RESTART	Restarts with retained states. **lp:** Block handle
WM_VSM_SIM_RESTART	Signals restart due to continue or single step. **lp:** Block handle
WM_VSM_SNAP_STATE	Signals a user-request to snap system states; that is, set the initial condition to the current state value.
WM_VSM_STOP_SIM	Signals that VisSim has stopped, either by user or time expiration. **wp:** 1 if single step **lp:** Pointer to parameter vector
WM_VSM_SUPPRESS_WARN_ UNCONNECT	Checks to suppress the unconnected input warning message. Return 1 if suppression is desired. **wp:** Port number (negative if output) **lp:** Block handle

Return flags for WM_VSM_CREATE

Flag	Description
VBF_HAS_STATE	Tells VisSim that block has state and can break an algebraic loop. VisSim calls this block once, before all other blocks, to present an initial condition; then the block is called during normal diagram execution.
VBF_USE_SIGNAL_DESCRIPTORS	Block input and output vector are vector of type SIGNAL (not double).
VBF_ALLOC_VEC_OUTPUT	Causes VisSim to automatically allocate output matrix for matrix input blocks.
VBF_EXECUTE_ALWAYS	Causes VisSim to execute the block regardless of graph connectivity.
VBF_ALLOW_VEC_CHAMELIONS	Causes connectors to accept scalar or vector connections.
VBF_STRAIGHT_WIRES	Causes wires to be drawn as straight lines rather than auto-routing.
VBF_MENU_ONLY	This is a menu item only and no block is created; however, the *userBlock*Event() function is called. This flag is useful for menu select to dialog.

Parameter allocation function - *userBlock*PA()

The *userBlock*PA() function is called when you first enter a DLL file/function pair in the dialog box for the `userFunction` block, or when a diagram is first read into VisSim. **Note:** This call is no longer required if block menu insertion is used.

```
long PASCAL userBlockPA(short *ppCount)
```

This function returns the parameter storage requirements, in bytes, for the `userFunction` block, and the number of prompted parameters. If you want VisSim to prompt for parameter values, set ppCount to the desired number. The maximum value for ppCount is 12. You need to allocate eight bytes for each prompted parameter. You can request additional storage for a function's private use. This additional storage can be accessed as array elements of the parameter vector after the first ppCount elements.

When this function is not supplied, no parameter storage is allocated.

Parameter change function - *userBlock*PC()

The *userBlock*PC() function is called when you click the right mouse over the `userFunction` block.

```
char * PASCAL userBlockPC(double * param)
```

This function lets you change block parameters for the `userFunction` block. If you want to create a dialog box to browse and set parameter values, you can do so and return a NULL pointer. If you want VisSim to browse and set parameter values for you, you should return a pointer to a NULL terminated string. The string should contain semicolons to separate each parameter prompt. You may have up to 12 parameters using this default method of parameter setting.

Parameter initialization function - *userBlock*PI()

The *userBlock*PI() function is called at block creation time, either from the menu or a file, for parameter initialization. It lets you provide initial values for parameters.

```
void PASCAL userBlockPI(double * param)
```

This function is called immediately after the parameter allocation function *userBlock*PA().

Simulation end function - *userBlock*SE()()

The *userBlock*SE() function is called just after a simulation ends to perform post simulation processing.

```
void PASCAL userBlockSE(double param[], long * runCount)
```

Simulation start function - *userBlock*SS()

The *userBlock*SS() function is called just prior to the start of a simulation to perform initialization processing necessary for a simulation run.

```
void PASCAL userBlockSS(double param[], long * runCount)
```

Exported functions

The seven exported functions are described below.

General information request - vissimRequest()

The vissimRequest() function provides a general, extensible request mechanism for obtaining information from VisSim.

```
EXPORT32 long vissimRequest(long req, arg2, arg3);
```

The first argument (long req) is a message code describing the action for VisSim to take. The message codes that pertain to writing a custom block are as follows:

Message	Description
VR_DISABLE_BLOCK_TYPE	Removes block matching name string from Blocks menu. **arg1:** LPSTR name string
VR_EXECUTE	Runs diagram but doesn’t change time (no integration).
VR_GET_BLOCK_PARAMS	Returns block parameter pointer. **arg1:** Block handle
VR_GET_CLEAR_BLOCK_ERR	Clears red error state on block. Uses currently executing block if block handle in arg1 is null. **arg1:** Block handle (opt.)
VR_GET_CONSTRAINTS	Gets constraint values **arg1:** Double * constraint
VR_GET_GLOBAL_CONSTRAINT_ BOUNDS	Gets global constraint bounds. **arg1:** Double * upper bound **arg2:** Double * lower bound
VR_GET_GLOBAL_CONSTRAINTS	Gets global constraints. **arg1:** Double *
VR_GET_GLOBAL COST	Gets global cost. **arg1:** Double * global cost
VR_GET_GLOBAL_OPT_INFO	Gets global optimization settings. **arg1:** OPT_INFO
VR_GET_GLOBAL_UNKNOWN_ BOUNDS	Gets global unknown bounds. **arg1:** Double * upper bound **arg2:** Double * lower bound
VR_GET_GLOBAL_UNKNOWNS	Gets global unknowns. **arg1:** Double *
VR_GET_GLOBAL_UNKNOWNS_ INPUT	Gets global unknown input. **arg1:** Double * global unknown initial condition

Message	Description
VR_GET_SOLVER_INFO	Gets settings from Implicit Solver tab in the Simulation Properties dialog box. **arg1:** Pointer to Implicit Solver
VR_GET_STARTUP_DIR	Returns VisSim directory string.
VR_GET_SUB_VERSION	Returns version letter suffix.
VR_GET_UNKNOWNS	Gets unknown values. **arg1:** Double * unknowns
VR_GET_UNKNOWNS_INPUT	Get unknown inputs. **arg1:** Double *
VR_GET_VERSION	Returns major version in high byte and minor version in low byte. **arg1:** SIM_INFO pointer
VR_GET_VISSIM_STATE	Gets current simulation state and copy to pointer in arg1.
VR_GET_WINDOW_HANDLE	Returns VisSim main window handle. Useful for model dialog creation.
VR_REALLOC_USER_PARAM	Reallocates parameter vector and returns newly reallocated pointer. **arg1:** Block handle **arg2:** New parameter size
VR_RESET_XFERS	For internal use only.
VR_SET_BLOCK_CONNECTOR_COUNT	Sets the connector count on the block. **arg1:** Block handle **arg2:** input # = upper word output # = lower word
VR_SET_BLOCK_MENU	Adds user-defined block to Blocks menu. **arg1:** Pointer to initialized USER_MENU_ITEM vector
VR_SET_CONNECTOR_CHAR	Sets indicator character on block connector. **arg1:** Character to set
VR_SET_CONNECTOR_LABEL	For internal use only.
VR_SET_GLOBAL_UNKNOWNS	Sets global unknowns from supplied vector. **arg1:** Double * global unknown
VR_SET_UNKNOWNS	Sets unknown values from user-supplied vector. **arg1:** Double * unknown
VR_SNAP_STATES	Causes VisSim to use current integrator/delay state as initial condition.

Get current simulation time - getSimTime()

This function stores the current simulation time in the double precision float variable pointed to by simTime.

```
EXPORT32 void PASCAL getSimTime( DOUBLE *simTime);
```

Get current simulation time step - getSimTimeStep()

This function stores the current simulation time step in the double precision float variable pointed to by simTimeStep.

```
EXPORT32 void PASCAL EXPORT getSimTimeStep( DOUBLE
   *simTimeStep);
```

Print debug message - debMsg ()

This function prints a dialog box containing a debugging message. Because you can't perform normal screen I/O under Windows or Windows NT, VisSim provides debMsg to display information pertaining to the variables for your userFunction block's function. The format is identical to the C printf() function. Since this function allows an arbitrary number of arguments, it must be called using the C language convention. To call it from Fortran or Turbo Pascal, for example, you must declare it as C language code. VisSim displays the output string in a standard dialog box that contains a Retry, Ignore, and Abort button. Press Retry or Ignore to continue the simulation. Press Abort to cancel the simulation.

```
EXPORT32 int CDECL EXPORT debMsg P((char LPSTR fmt , ... ));
```

Request simulation end - stopSimulation()

This function requests that VisSim stop a simulation.

```
EXPORT32 void PASCAL EXPORT stopSimulation( int stopVal);
```

If stopVal is 1, the current simulation run is stopped. If you have activated the Auto Restart parameter under the Range tab in dialog box for the Simulate > Simulation Properties command, VisSim starts the next simulation run. If stopVal is 2, all simulation runs are stopped.

Flag error - setBlockErr()

This function requests that VisSim flag the currently executing block in red. All nested blocks will be flagged in red as well. To clear a flagged block, click the right mouse button on the block.

```
EXPORT32 void PASCAL EXPORT setBlockErr();
```

Add menu item - setUserBlockMenu()

This function adds a block (or menu item) to the VisSim menus.

```
EXPORT32 void EXPORT setUserBlockMenu P((USER_MENU_ITEM * ));
```

This functions recognizes one argument, which is a pointer to an array of structures. The structures define the menu name, DLL name, number of inputs, number of outputs, number of parameters, and help string. The format of the structure is as follows:

```
typedef struct {
  char * menuName;
  char * funcName;
  int inputCount;
  int outputCount;
  int paramCount;
  char * helpText;
} USER_MENU_ITEM;
```

You need one structure for every block (or menu item).

In addition, the last element in the array of structures that is passed back in must be {0}.

Debugging hints

The following guidelines will make it easier to debug your DLLs:

- MSVC lets you set a breakpoint in your DLL before running VisSim.
- Set VISSIM32.EXE as the calling program by choosing Build > Settings > Debug. This starts VisSim automatically when you press F5 (or choose Debug > Go).
- Use conditional breakpoints to get control near the problem area.
- Single-step with values in watch window to find problems.
- If your program hangs, press CTRL+ALT+PRTSCRN (or choose Debug > Break) to return control to the debugger. This works best under Windows 95 and Windows NT.
- On Debug > Break or a General Protection Fault, use View > Stack Trace to find the location of the offending instruction.
- Floating point errors, if continued, often result in General Protection Fault that point to source line of floating point error.

Binding a DLL to a userFunction block

When you bind a DLL to a userFunction block, VisSim calls the DLL each time the block is executed.

▶ **To bind a DLL to a userFunction**

1. Insert a `userFunction` block in your diagram.
2. Choose Edit > Block Properties.

3. Point to the `userFunction` block and click the mouse.
4. Do the following:
   - In the DLL File Name box, enter the name of the DLL file containing the user function.
   - In the Base Function Name box, enter the base name of the function.
5. Click on the OK button, or press ENTER.

Adding a user-written block to the Blocks menu

This allows VisSim to invoke the DLL at VisSim start-up, and allows the DLL to insert blocks into the Block menu by calling setUserBlockMenu().

▶ **To add a user-written block to the Blocks menu**

1. In VisSim, choose Edit > Preferences.
2. Click on the Addons tab.
3. Double-click on the ellipsis (…) and type in the path to the DLL function, or click on the … button to locate the DLL function.
4. Click on the OK button, or press ENTER.

Appendix C

Toolbox and Components Libraries

VisSim provides a wide range of toolbox functions and diagram components to further enhance your modeling and simulation capabilities. Because they are in .VSM file format, you can easily incorporate them into your diagrams using the File menu's Add command or the `embed` block.

Toolboxes

The toolbox libraries supplied with VisSim include functions for controls, electro-mechanical design, Padé approximations, and signal generation. VisSim also provides a toolbox library of miscellaneous functions (called Tools).

Controls toolbox library (\VISSIM30\TOOLBOX\CONTROLS)

Toolbox function	Description
DERIV_A.VSM	Continuous derivative model
DERIV_D.VSM	Discrete derivative function
DIFFR.VSM	Discrete difference model
HYST_CTL.VSM	Hysteresis controller
HYSTER.VSM	Hysteresis function
LAG.VSM	General first order unity gain all pole low pass filter
LEAD.VSM	General first order lead unity gain compensator
P_CTL.VSM	Proportional (P) controller
PI_CTL.VSM	Proportional Integral (PI) controller
PID_CTL.VSM	Proportional Integral Derivative (PID) controller

Toolbox function	Description
RATE_LIM.VSM	Rate limited controller
RFB_CTL.VSM	Rate feedback controller
TF1_CONT.VSM	Continuous first order transfer function
TF1_DISC.VSM	Discrete first order transfer function
TF2_CONT.VSM	Continuous second order transfer function
TF2_DISC.VSM	Discrete second order transfer function
TRIM_INT.VSM	Trimmed integrator - finds initial state for zero derivative
ZINTBR.VSM	Digital integrator (backward rectangular)
ZINTFR.VSM	Digital integrator (forward rectangular)
ZINTTR.VSM	Digital integrator (trapezoidal)

Electro-mechanical toolbox library (\VISSIM30\TOOLBOX\ELECHMECH)

Toolbox Function	Description
A2D.VSM	Analog-to-digital converter model with settable *dT* and bit length
ACDQ_MOT.VSM	Three-phase AC motor model utilizing DQ coordinate transformation
D2A.VSM	Digital-to-analog converter model with settable *dT* and bit length
DC_MOT.VSM	Armature controlled DC motor
ENCODER.VSM	Encoder model
MUX4.VSM	Four-channel multiplexer
PWM.VSM	Pulse wave modulation model

Padé toolbox library (\VISSIM30\TOOLBOX\PADE)

Toolbox Function	Description
PADE1.VSM	First order Padé approximation to time delay
PADE2.VSM	Second order Padé approximation to time delay
PADE3.VSM	Third order Padé approximation to time delay
PADE4.VSM	Fourth order Padé approximation to time delay

Signal generation toolbox library (\VISSIM30\TOOLBOX\SIGGEN)

Toolbox Function	Description
3_PHASE.VSM	Three-phase sinusoidal signal generator
CAL_TIME.VSM	Simulation time in day, hour, minutes
DT.VSM	Simulation time in seconds
SAWTOOTH.VSM	Generates a sawtooth wave form
SQR_WAVE.VSM	Generates a square wave form
TRIANGLE.VSM	Generates a triangular wave form

Tools toolbox library (\VISSIM30\TOOLBOX\TOOLS)

Toolbox Function	Description
AVG_VAL.VSM	Average (mean) value estimator for periodic signals
COUNTER.VSM	Pulse counter
MAG_PHAS.VSM	Computes the magnitude ratio and phase margin between two input signals
MAX_VAL.VSM	Detects the high peak value every cycle of a periodic wave form
MIN_VAL.VSM	Detects the peak low value every cycle of a periodic wave form
PERIOD.VSM	Wavelength estimator for a periodic signal
PH_DIFF.VSM	Phase difference estimator
RMS.VSM	Computes the root mean square (RMS) value of a signal
SWEEP.VSM	Provides parameter sweep settings
VEC_ANLY.VSM	Amplitude - phase vector display

Components

The components libraries supplied with Professional VisSim include DSP, dynamical, electro-mechanical, electric, hydraulic, process control, thermal, and turbine components.

DSP library (\VISSIM30\COMPNENT\DSP)

Component	Description
CONVOLXI.VSM	Analytical and numerical solutions for an impulse response system
KFILT.VSM	Filter for estimating particle coordinates and velocity components
WAVELETS.VSM	Two-parameter wavelet generation

Dynamical system library (\VISSIM30\COMPNENT\DYNSYS)

Component	Description
ANTENNA.VSM	Position control of flimsy antenna
REEL.VSM	Control of wire speed on a motor-controlled take-up reel
ROBEAM3.VSM	Reduced-order steady-state beam model

Electro-mechanical library (\VISSIM30\COMPNENT\ELECHMECH)

Component	Description
2DCMOTS.VSM	Two motors connected by a flexible belt
CRANE.VSM	Movement of a crane payload
HOIST.VSM	One mass nonlinear hoistway
STEPPER.VSM	Stepper motor for Variable Reluctance or Permanent Magnet types

Electrical library (\VISSIM30\COMPNENT\ELECTRIC)

Component	Description
POWERSUP.VSM	Two-diode, full-wave rectified DC power supply

Hydraulic libraries (\VISSIM30\COMPNENT\HYDRAULIC…)

\ACTUATOR library

Component	Description
HYDMOTOR.VSM	Hydraulic motor
TWNCHMAC.VSM	Double-sided actuator

\INCLUDE library

Component	Description
GENDEFS.VSM	General definitions for hydraulic library

\MECHLOAD library

Component	Description
LINEAR.VSM	Linear mechanical load
ROTATNAL.VSM	Rotational mechanical load

\MINORLOS library

Component	Description
BEND.VSM	Fluid flow through a pipe bend
SUDCONTR.VSM	Sudden contraction of fluid due to an exit from a large chamber into a pipe
SUDEXPNS.VSM	Sudden expansion of fluid due to exit into a large chamber

\MISC library

Component	Description
MASSWLIM.VSM	Mass with limits
VOLUME.VSM	Capacitance volume effects

\ORIFICE library

Component	Description
ORIFICE.VSM	Flow through an orifice

\PIPE library

Component	Description
CONDUIT.VSM	Pressure gradient evaluation for laminar and turbulent flow through conduits

\POWRLOSS library

Component	Description
POWRLOSS.VSM	Power loss and temperature rise in fluid

\PUMPS library

Component	Description
POSDSPMP.VSM	Positive displacement pump
PRSCMPMP.VSM	Pressure compensated pump.

\SPLTMERG library

Component	Description
1ORFSPLT.VSM	Split one fluid stream into two (orifice at the exit on one of the outlet ports)
3WAYSPLT.VSM	Splits one fluid stream into three
MERGE.VSM	Merges two fluid streams into one
MERGE3LN.VSM	Merges three fluid streams into one and includes an orifice on the exit
MERGEALG.VSM	Joins two streams algebraically without introducing a pressure state
MRGALG3I.VSM	Joins three streams algebraically without introducing a pressure state
PLNMMERG.VSM	Merges two fluid streams into one and the downstream boundary condition is the flow rate
PLNMRG3L.VSM	Merges three fluid streams into one and the downstream boundary condition is the flow rate
PRSTRAN.VSM	Pressure transients in hydraulic conduits
SPLTWORF.VSM	Splits one fluid stream into two (orifice at the exit of each outlet port)

\VALVES library

Component	Description
REGLVALV.VSM	Pressure regulating valve
RELFVALV.VSM	Pressure relief valve

Process control library (\VISSIM30\COMPNENT\PROCESS)

Component	Description
BEER.VSM	Beer brewing model
CSTR.VSM	Simple continuous stirred tank reactor model
DISTIL.VSM	Binary distillation column
NISOTH.VSM	Non-isothermal continuous stirred tank reactor model

Thermal control library (\VISSIM30\COMPNENT\THERMAL)

Component	Description
HEATEXCH.VSM	Heat exchanger model

Turbine library (\VISSIM30\COMPNENT\TURBINE)

Component	Description
GT2.VSM	Twin spool gas turbine model

Appendix D

Sample Block Diagrams

VisSim comes with numerous block diagrams that cover a broad spectrum of applications and illustrate many of VisSim's fundamental design and simulation features. The sample block diagrams reside in the subdirectories under \VISSIM30\APPEXAMPL. This appendix describes the sample block diagrams provided by VisSim.

This subdirectory	Contains this type of diagram
\AEROSPAC	Aerospace diagrams
\ANIMATE	Animation diagrams
\BIOPHYS	Biophysical diagrams
\BUSINESS	Business diagrams
\CHEMENG	Chemical engineering diagrams
\CTRL_DES	Control design diagrams
\DYN_SYS	Dynamic system diagrams
\ELECTRO	Electro-mechanical diagrams
\ENVIRON	Environmental diagrams
\FIXPTDSP	Fixed Point DSP diagrams
\MMI	Man-machine interface diagrams
\MOTION	Motion control diagrams
\OPTIMIZE	Optimization diagrams
\POWER	Power system and component diagrams

This subdirectory	Contains this type of diagram
\PROCESS	Process control diagrams
\SIG_PROC	Signal processing diagrams
\STATCHRT	Logic diagrams and state machines

Aerospace block diagrams (\VISSIM30\APPEXAMPL\AEROSPAC)

Block diagram	Description
6DOF.VSM	Six degree of freedom simulation

Animation block diagrams (\VISSIM30\APPEXAMPL\ANIMATE)

Block diagram	Description
2LINK.VSM	Loose model of a two-link pendulum
PHYSBE.VSM	Physiological Simulation Benchmark Experiment that shows the bloodflow in the human body
ROCKET.VSM	Simulation of the trajectory of a ballistic missile

Biophysical block diagrams (\VISSIM30\APPEXAMPL\BIOPHYS)

Block diagram	Description
BIOREACT.VSM	Bio-reactor showing cell culture growth in a nutrient solution
PHYSBE.VSM	Physiological Simulation Benchmark Experiment that shows the bloodflow in the human body

Business block diagrams (\VISSIM30\APPEXAMPL\BUSINESS)

Block diagram	Description
CASHFLOW.VSM	Track cash flow within a manufacturing organization
WAGECHAO.VSM	Affect of changing interest rate on employment

Chemical engineering block diagrams (\VISSIM30\APPEXAMPL\CHEMENG)

Block diagram	Description
AMMONAB.VSM	Steady state absorption column where ammonia is recovered from an ammonia-air gas mixture by absorption into water, using a counter-current-packed column
BATCHD.VSM	*n*th order homogeneous liquid phase reaction in a batch tank reactor
BATEX.VSM	Single solution batch extraction
BATSEQ.VSM	Complex batch reaction sequence (can be used to study various reaction kinetics of interest simply by varying the rate constants)
BEAD.VSM	Diffusion and reaction in a spherical bead
BSTILL.VSM	Binary batch distillation column
CASTOR.VSM	Batch decomposition of acetylated castor oil to drying oil
CHAOS.VSM	Chaotic oscillatory behavior
CONSTILL.VSM	Continuous binary distillation column
CSTR.VSM	System of three continuous stirred tank reactors with first order isothermal reaction
CSTRCOM.VSM	Isothermal CSTR with complex reaction
DISRE.VSM	Dynamic behavior of an non-ideal isothermal tubular reactor to predict the variation of concentration with respect to both axial distance along the reactor and flow time
DISRET.VSM	Dispersion model of DISRE.VSM is extended for non-isothermal reactions to include the dispersion of heat from a first order reaction
DRY.VSM	Drying of solids by diffusion
EQEX.VSM	Simple equilibrium stage extractor
EXMULTI.VSM	Continuous equilibrium multistage extractor
GPJIF.VSM	Solution of partial differential equations
HEATEX.VSM	Shell and tube heat exchanger
HMT.VSM	Semi-batch manufacture of hexamethylenetriamine.
HOPFBIF.VSM	Hopf bifurcation
LORENZ.VSM	Random differential equation behavior
MCSTILL.VSM	Continuous multicomponent distillation column

Block diagram	Description
NOSTR.VSM	Non-ideal stirred tank reactor
ROD.VSM	Radiation from metal rod
TANKBLD.VSM	Liquid stream blending problem
TANKHYD.VSM	Two interacting tank reservoirs
THERM.VSM	First order, exothermic reaction in a continuous stirred-tank reactor, equipped with jacket cooling
TUBE.VSM	Tubular reactor, steady state design for an nth order reaction
TUBEMIX.VSM	Non-ideal tube-tank mixing system
TUBTANK.VSM	Comparison of steady state conversions for both continuous tank and tubular reactors for nth order reaction kinetics
TWOTANK.VSM	Two tank level control, where the level control of tank 2 is based on the regulation of the inlet flow to the tank 1

Control design block diagrams (\VISSIM30\APPEXAMPL\CTRL_DES)

Block diagram	Description
PIDTUNE.VSM	PID control gain optimization
PLL_CTRL.VSM	Phase-locked loop servo that models a simple controller
RL_DES.VSM	Root locus design example
V_DERPOL.VSM	Van der Pol’s nonlinear dynamical system

Dynamical systems block diagrams (\VISSIM30\APPEXAMPL\DYN_SYS)

Block diagram	Description
BOUNCE.VSM	Bouncing ball and the dynamic exchange of data
CORNU.VSM	Cornu's spiral
LORENZ.VSM	Chaotic system based on the Lorenz attractor
MANUFACT.VSM	Manufacturing and product distribution organization
METER.VSM	Illustrates uses of the meter block
RAYLEIGH.VSM	Rayleigh equation for bubble growth in super heated liquids
ROCKET.VSM	Rocket dynamics
SOMBRERO.VSM	Parameter sweep for Sombrero function
SPRING.VSM	Simple, second-order, damped harmonic system
STUKBLOK.VSM	Motion with static coulomb stiction

Electro-mechanical block diagrams (\VISSIM30\APPEXAMPL\ELECTRO)

Block diagram	Description
ACMOTOR.VSM	Three-phase AC motor system that plots motor speed and torque against time
DCMOTOR.VSM	GAE 12 amp micro-torque motor response curve under mild loading conditions
DOORSYS.VSM	Digitally controlled electro-mechanical door system
FLEXLOAD.VSM	Response of a DC motor and gearbox with deadband to a flexible load
FW_RECT.VSM	Full wave rectifier test case
SM_1PH.VSM	Stepper motor system

Environmental block diagrams (\VISSIM30\APPLEXAMPL\ENVIRON)

Block diagram	Description
BIOREACT.VSM	Bio-reactor showing cell culture growth in a nutrient solution
ROOMCTRL.VSM	HVAC model of a single-room cooling system with on/off thermostat

Fixed-point DSP block diagrams (\VISSIM30\APPLEXAMPL\FIXPTDSP)

Block diagram	Description
FILTDESN.VSM	Unit delay filter implementation
SCALE.VSM	Simulation of numerical overflow in fixed-point DSP

Man-machine interface block diagrams (\VISSIM30\MMI)

Block diagram	Description
MORE_PLT.VSM	Moore control panel
PIDPLATE.VSM	PID control panel

Motion control block diagrams (\VISSIM30\MOTION)

Block diagram	Description
PACDEMO.VSM	Pacific Scientific motion control demo
QUADCODE.VSM	Quadrature encoding

Optimization block diagrams (\VISSIM30\APPEXAMPL\OPTIMIZE)

Block diagram	**Description**
2POINT.VSM	Classic two-point value problem using cost and `parameterUnknown` blocks
LC_FIND.VSM	Illustrates the use of `constraint` and `unknown` blocks to determine initial conditions of integrators
PIDTUNEZ.VSM	Optimization of a second order plant with a first order controller using `cost` and `parameterUnknown` blocks
ROOTS.VSM	Illustrates the use of `constraint` and `unknown` blocks to find one of the roots of a quadratic equation

Power block diagrams (\VISSIM30\APPEXAMPL\POWER)

Block diagram	**Description**
INVERTER.VSM	Inverter
PSV_TRBN.VSM	Gas turbine simulation

Process control block diagrams (\VISSIM30\APPEXAMPL\PROCESS)

Block diagram	**Description**
CSTRS.VSM	Simulation of a set of three isothermal continuous stirred tank reactors in a series
POWPLANT.VSM	Power plant simulation
ROOMCTRL.VSM	Room temperature controller
VALVE.VSM	Simulates a mechanical valve with finite actuation time

Signal processing block diagrams (\VISSIM30\APPEXAMPL\SIG_PROC)

Block diagram	Description
ANL_PLL.VSM	Analog phase-locked loop system
DIG_PLL.VSM	Digital phase-locked loop system
FILTTEST.VSM	Illustrates filter design capabilities
F_SERIES.VSM	Fourier series problem
PWM_EX.VSM	Pulse width modulation of a sinusoidal signal

Logic diagrams and state machines (\VISSIM30\APPEXAMPL\STATCHRT)

Block diagram	Description
LOG_EX1.VSM	Counter and reset counter implementation
LOGICBLK.VSM	Logic and timing blocks
ONESHOT.VSM	Simulation of a “one shot”
PIDPAPER.VSM	PID controller logical simulation
STATEM1.VSM	State machine demo

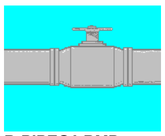
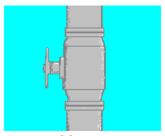
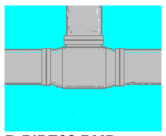
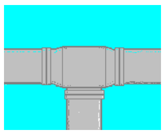
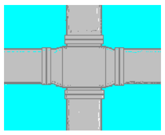
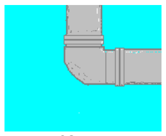
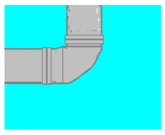
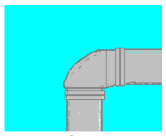
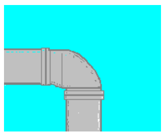
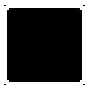
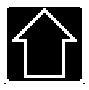
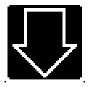
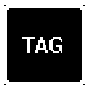
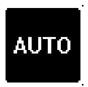
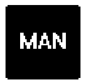
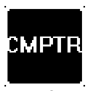
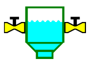
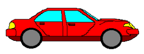
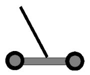
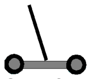
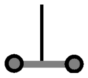
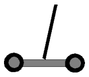
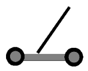
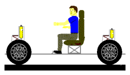
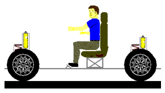
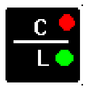
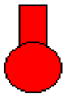
Appendix E

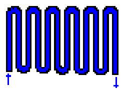
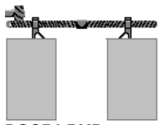
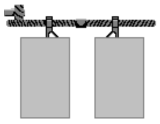
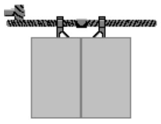
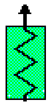
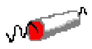
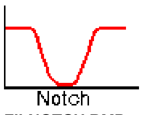
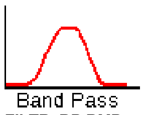
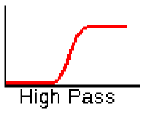
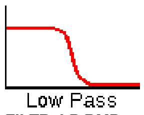
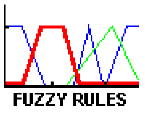
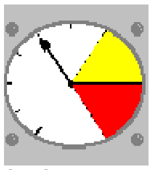
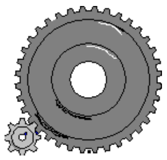
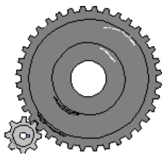
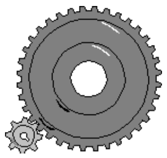
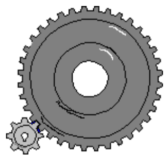
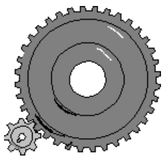
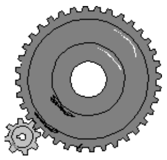
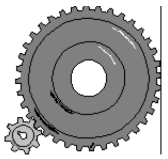
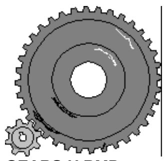
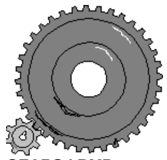
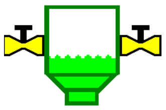
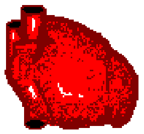
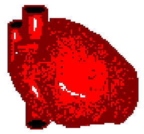
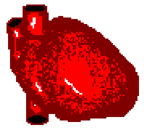
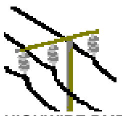
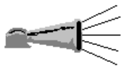
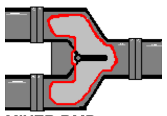
Working with Bitmaps

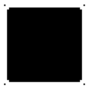
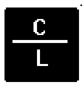
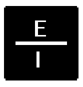
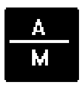
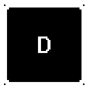
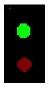
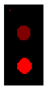
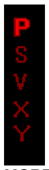
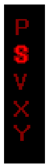
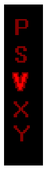
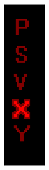
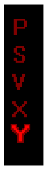
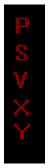
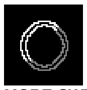
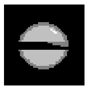
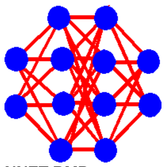
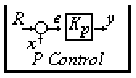
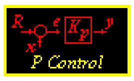
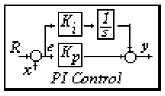
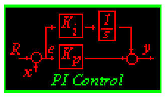
Pictures, or graphical images, can be configured on many VisSim blocks to enhance the visual representation of a block diagram. They can also be used to create animated simulations.

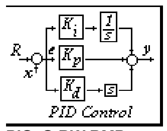
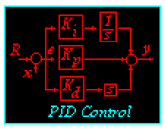
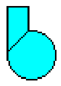
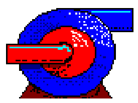
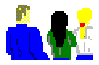
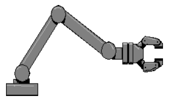
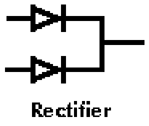
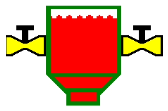
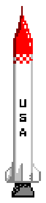
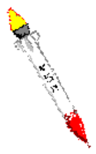
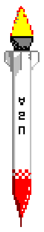
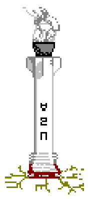
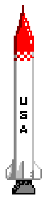
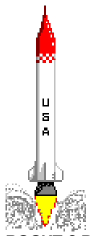
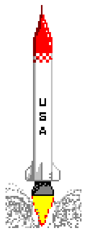
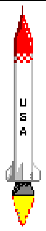
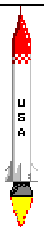
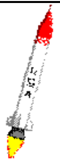
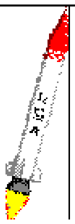
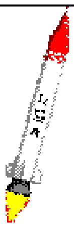
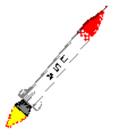
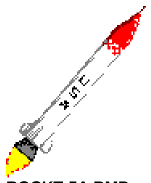
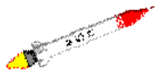
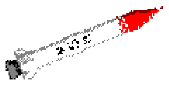
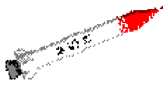
VisSim provides a Bitmap library containing a wide range of motion and process control pictures. The library is installed in \VISSIM30\BITMAPS\DIAGRAM. If the library does not contain a picture you need, you can always create one using any of the numerous drawing and icon creation packages that run under Windows. When you create pictures, follow these simple guidelines:

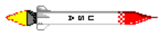
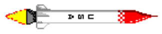
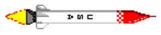
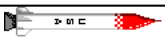
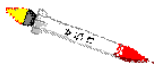
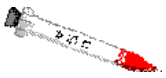
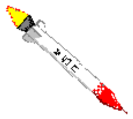
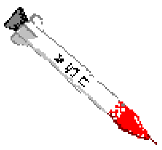
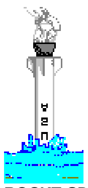
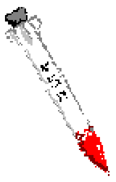
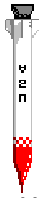
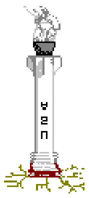
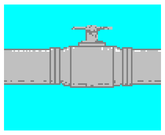
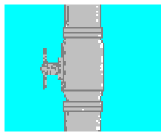
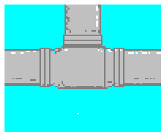
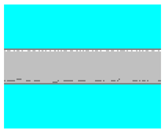
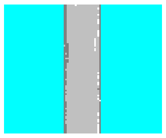
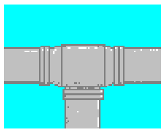
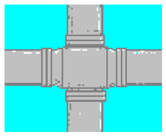
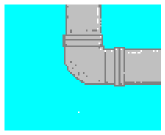
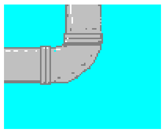
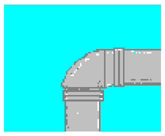
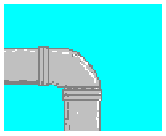
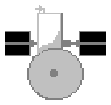
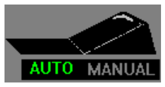
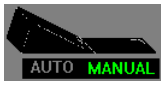
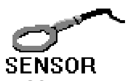
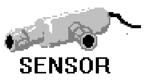
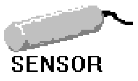
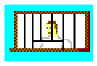
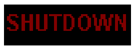
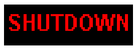
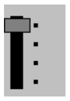
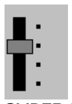
- Save the pictures in .BMP file format
- To avoid screen clutter, create pictures no larger than ¾" by ¾"

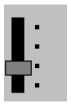
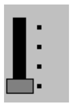
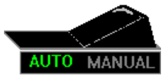
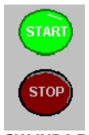
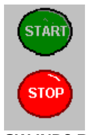
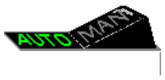
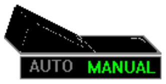
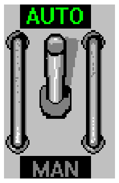
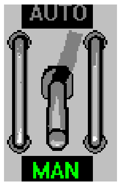
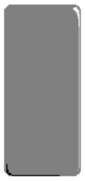
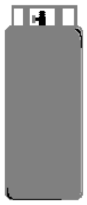
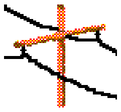
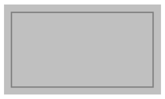
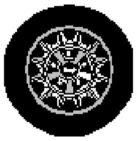
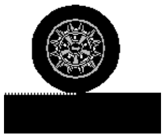
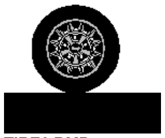
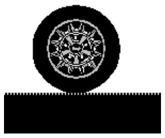
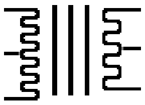
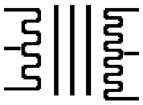
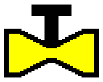
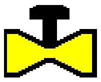
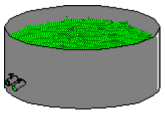
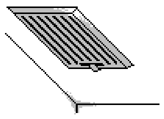
3D-BUT1.BMP	3D-BUT2.BMP	 B-PIPEO1.BMP	 B-PIPE02.BMP	 B-PIPE03.BMP
B-PIPE04.BMP	B-PIPE05.BMP	 B-PIPE06.BMP	 B-PIPE07.BMP	 B-PIPE08.BMP
 B-PIPE09.BMP	 B-PIPE10.BMP	 B-PIPE11.BMP	 BAIL-00.BMP	 BAIL-01.BMP
 BAIL-02.BMP	BAIL-O3.BMP	 BAIL-04.BMP	 BAIL-05.BMP	 BAIL-O6.BMP
 BAIL-07.BMP	BAIL-08.BMP	BAIL-L00.BMP	BAIL-L01.BMP	BAIL-L02.BMP
BAIL-PAN.BMP	BFLY.BMP	 BLU-TANK.BMP	 CAR.BMP	 CART-1.BMP
 CART-2.BMP	 CART-3.BMP	 CART-4.BMP	 CART-5.BMP	 CHAS-A.BMP
 CHAS-B.BMP	CHASI.BMP	 CLRDGN.BMP	CO-AXIAL.BMP	 COMPRESS.BMP

 COOLCOIL.BMP	 DANGER.BMP	 DISH-R.BMP	 DISH.BMP	 DOOR1.BMP
 DOOR2.BMP	 DOOR3.BMP	 EAR.BMP	 EVAP.BMP	 FAIL.BMP
 FIBEROPT.BMP	 FILNOTCH.BMP	 FILTR_BP.BMP	 FILTR_HP.BMP	 FILTR_LP.BMP
 FUZZY.BMP	 GAUGE.BMP	 GEARS-A.BMP	 GEARS-B.BMP	 GEARS-C.BMP
 GEARS-D.BMP	 GEARS-E.BMP	 GEARS-F.BMP	 GEARS-G.BMP	 GEARS-H.BMP
 GEARS-I.BMP	 GRN-TANK.BMP	 HEART-A.BMP	 HEART-B.BMP	 HEART-C.BMP
 HEATCOIL.BMP	 HIGHWIRE.BMP	 HORN1.BMP	 HORN2.BMP	 MIXER.BMP

 MOORE-00.BMP	 MOORE-01.BMP	 MOORE-02.BMP	 MOORE-03.BMP	 MOORE-04.BMP
MOORE-05.BMP	 MORE-L01.BMP	 MORE-L02.BMP	 MORE-M-P.BMP	 MORE-M-S.BMP
 MORE-M-V.BMP	 MORE-M-X.BMP	 MORE-M-Y.BMP	 MORE-M00.BMP	MORE-PAN.BMP
 MORE-SWT.BMP	MORE-U00.BMP	MORE-U01.BMP	MORE-U02.BMP	MORE-U03.BMP
MORE-U04.BMP	MORE-U05.BMP	MORE-U06.BMP	MORE-U07.BMP	MORE-U08.BMP
 MORE-U09.BMP	 NNET.BMP	NORM.BMP	 P_CTL_BW.BMP	 P_CTL.BMP
PANEL-BG.BMP	PANEL.BMP	PASS.BMP	 PLC_BW.BMP	 PI_CTL.BMP

 PIC_C-BW.BMP	 PIC_CTL.BMP	 PUMP-LQD.BMP	 PUMP.BMP	 Q.BMP
 R-ARM.BMP	 RECTIFI.BMP	 RED-TANK.BMP	 ROCKT-1.BMP	 ROCKT-10.BMP
 ROCKT-11.BMP	 ROCKT-12.BMP	 ROCKT-A1.BMP	 ROCKT-2.BMP	 ROCKT-2A.BMP
 ROCKT-3.BMP	 ROCKT-3A.BMP	 ROCKT-4.BMP	 ROCKT-4A.BMP	 ROCKT-4B.BMP
 ROCKT-5.BMP	 ROCKT-5A.BMP	 ROCKT-6.BMP	 ROCKT-6A.BMP	 ROCKT-6B.BMP

 ROCKT-7.BMP	 ROCKT-7A.BMP	 ROCKT-78.BMP	 ROCKT-7C.BMP	 ROCKT-8.BMP
 ROCKT-8A.BMP	 ROCKT-9.BMP	 ROCKT-9A.BMP	 ROCKT-SD.BMP	 ROCKT10A.BMP
	 ROCKT11A.BMP	 ROCKT12A.BMP	 S-PIPE01.BMP	 S-PIPE02.BMP
 S-PIPE03.BMP	 S-PIPE04.BMP	 S-PIPE05.BMP	 S-PIPE06.BMP	 S-PIPE07.BMP
 S-PIPE08.BMP	 S-PIPE09.BMP	 S-PIPE10.BMP	 S-PIPE11.BMP	 SAT.BMP
 SD-AUTO.BMP	 SD-MAN.BMP	 SENSOR-H.BMP	 SENSOR-L.BMP	 SENSOR.BMP
 SERVER.BMP	 SHUTDWN0.BMP	 SHUTDWN2.BMP	 SLIDER-1.BMP	 SLIDER-2.BMP

 SLIDER-3.BMP	 SLIDER-4.BMP	 SW-AUTO.BMP	 SW-IND1.BMP	 SW-IND2.BMP
 SW-LON.BMP	 SW-MAN.BMP	 SW-TOGG1.BMP	 SW-TOGG2.BMP	 TANK-BEZ.BMP
 TANK-SM.BMP	 TELEPOLE.BMP	 THERM-BG.BMP	 TIRE-ONL.BMP	 TIRE.BMP
 TIRE1.BMP	 TIRE2.BMP	 TRANS-M.BMP	 TRANS.BMP	 VALVE-H.BMP
 VALVE.BMP	 VAT.BMP	 VENT.BMP		

Appendix F

VisSim Viewer

VisSim Viewer is a run-time version of VisSim that allows you to distribute your block diagram models to end users not licensed to use VisSim. VisSim Viewer provides end users with all the features and capabilities of VisSim, with the following exceptions:

- It will not let end users create new block diagrams
- It will not let end users add, delete, or reposition block, or change wiring paths in the diagrams they view

Distributing VisSim Viewer and your block diagrams to end users

As a licensed user of Professional VisSim, you are granted a royalty-free right to reproduce and distribute up to 100 copies of VisSim Viewer, as described under the terms of the VisSim Viewer Distribution License Agreement, at the end of this appendix. Please take a minute to review this agreement.

▶ **To distribute VisSim Viewer**

1. Copy VSVIEWER.EXE from your VisSim main directory to another disk.
2. Copy your diagram files to the copied VisSim Viewer disk or another disk.
3. Include printed copies of the following sections of this appendix with the VisSim Viewer: "VisSim Viewer documentation," "Installing and starting VisSim Viewer," and "VisSim Viewer End User License Agreement."

VisSim Viewer documentation

The *VisSim Viewer User's Guide* has been formatted as a Microsoft Write file named VIEWER.WRI and is included on the VisSim Viewer disk. End users should make a printed copy of this file as soon as they install VisSim Viewer on their computers. VIEWER.WRI contains information on:

- Installing, starting, and quitting VisSim Viewer
- Basic windowing techniques
- Using online help

It also provides tutorial lessons that cover:

- Loading, viewing, simulating, and optimizing a block diagram
- Observing simulation results using plot and stripChart blocks
- Printing block diagrams, plots, and strip charts
- Copying block diagrams to other applications
- Saving files

Installing and starting VisSim Viewer

Before installing VisSim Viewer, end users should check that their computers meet the following minimum configuration:

- Microsoft Windows version 3.1 or higher
- EGA or higher resolution monitor
- At least 1 MB hard disk space
- At least 1 MB RAM with 256K configured as extended memory

▶ **To install VisSim Viewer**

1. Start Windows.
2. Insert the VisSim Viewer disk into the floppy drive.
3. Do one of the following:

For this platform	Do this
Windows 95 or Windows NT	Select Start and choose the Run command.
Windows 3.1	Select File from the Program Manager menu bar and choose the Run command.

4. In the Command Line box, type a:install or b:install, and click on OK.

At the completion of the installation, Install builds a VisSim group window with a VisSim Viewer icon inside it. To start VisSim Viewer, double-click on the VisSim Viewer icon.

VisSim Viewer End User License Agreement

This is a legal agreement between you, the end user, and Visual Solutions, Incorporated ("VSI"). BY OPENING THIS SEALED DISK PACKAGE AND USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNOPENED DISK PACKAGE AND THE ACCOMPANYING ITEMS TO THE PLACE YOU OBTAINED THEM FOR A FULL REFUND. THIS AGREEMENT IS SEPARATE FROM ANY AGREEMENT BETWEEN YOU AND THE SUPPLIER OF ANY ACCOMPANYING .VSM OR OTHER FILES.

1. **OWNERSHIP OF THE SOFTWAR**E. The enclosed VSI software program ("SOFTWARE") and the accompanying written materials are owned by VSI or its suppliers and are protected by United States copyright laws and international treaty provisions and all other applicable national laws. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g., a book or musical recording) except that if the SOFTWARE is not copy protected, you may either (a) make one copy of the SOFTWARE solely for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk, provided you keep the original solely for backup or archival purposes. You may not copy the written materials accompanying the SOFTWARE.

2. **GRANT OF LICENSE.** VSI grants to you the right to use one copy of the enclosed SOFTWARE on a single computer. The SOFTWARE is in "use" on a computer when it is loaded into temporary memory (i.e., RAM) or installed into permanent memory (e.g., hard disk, CD-ROM, or other storage device) of that computer. You may network the SOFTWARE, provided you have a separate license for each computer at which the SOFTWARE is used.

3. **OTHER RESTRICTIONS.** You may not rent or lease the SOFTWARE, but you may transfer the SOFTWARE and accompanying written materials on a permanent basis, provided you retain no copies and the recipient agrees to the terms of this Agreement. You may not reverse engineer, decompile, or disassemble the SOFTWARE. If the SOFTWARE is an update or has been updated, any transfer must include the update and all prior versions.

4. **LIMITED WARRANTY.** VSI warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of your receipt of the SOFTWARE. Any implied warranties on the SOFTWARE are limited to 90 days. Some states do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. VSI makes no warranties concerning .VSM files or other software supplied by other companies.

5. **CUSTOMER REMEDIES** VSI'S ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT VSI'S CHOICE, EITHER (A) RETURN OF THE PRICE PAID OR (B) REPLACEMENT OF THE SOFTWARE THAT DOES NOT MEET VSI'S LIMITED WARRANTY AND WHICH IS RETURNED TO VSI WITH A COPY OF YOUR RECEIPT. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or 30 days, whichever is longer. These remedies are not available outside the United States of America.

6. **NO OTHER WARRANTIES** VSI DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, WITH RESPECT TO THE SOFTWARE AND THE ACCOMPANYING WRITTEN MATERIALS. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

7. This Limited Warranty is void if failure of the SOFTWARE has resulted from modification, accident, abuse, or misapplication.

8. **NO LIABILITY FOR CONSEQUENTIAL DAMAGES** IN NO EVENT SHALL VSI OR ITS SUPPLIERS BE LIABLE FOR ANY OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THIS VSI PRODUCT, EVEN IF VSI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, VSI'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THIS SOFTWARE. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you.

9. This Agreement is governed by the laws of the State of Massachusetts, U.S.A.

10. Should you have any questions concerning this Agreement, or if you desire to contact VSI for any reason, please write: Visual Solutions, Inc., 487 Groton Rd., Westford, Massachusetts 01886.

11. **U.S. GOVERNMENT RESTRICTED RIGHTS** The SOFTWARE and accompanying written materials are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data And Computer Software clause at 52.227-7013. Contractor/manufacturer is Visual Solutions, Inc./487 Groton Rd./Westford, MA 01886.

VisSim Viewer Distribution Agreement

This is a legal agreement between you, a licensed owner of Professional VisSim or the VisSim Viewer software package and Visual Solutions. Visual Solutions grants you a royalty-free right to reproduce and distribute 100 copies of VisSim Viewer provided that you: (a) distribute VisSim Viewer only in conjunction with and as a part of your block diagram(s); (b) do not require the use of VisSim companion products (See "Additional Grant of License"); (c) do not use Visual Solutions' name, logo, or trademarks to market your block diagram(s); (d) legally acknowledge Visual Solutions' copyrights and trademarks where applicable; and (e) agree to indemnify, hold harmless, and defend Visual Solutions and its suppliers from and against any claims or lawsuits, including attorney fees that arise or result from the use or distribution of your block diagram(s).

Additional Grant of License: If, to use your block diagram(s), VisSim Viewer end users require the use of one or more VisSim companion products, call your Visual Solutions Sales Representative for product re-sale terms and restrictions.

Index

!= (not equal to) block, 152
$firstPass variable, 132
$lastPass variable, 132
$runCount variable, 132
$timeStart, 132
$timeStep variable, 132
$timeStop, 132
* (multiply) block, 142
/ (divide) block, 145
< block, 147
<= block, 148
== (equal to) block, 149
> block, 153
>= block, 155
1/S block, 192, 202, 231
1/X (inverse) block, 156
1/Z block (unitDelay), 51, 52, 257

A

abs block, 157
acos block, 159
adaptive Bulirsh-Stoer algorithm, 38
adaptive Runge Kutta 5th algorithm, 38
Add (File), 128
Add Connector (Edit), 22
add menu item function, 286
adding block diagrams, 128
aerospace block diagrams, 298
alarms, audio and visual, 199
algebraic loops, 58, 84
aligning blocks, 29
analog filter prototypes, 101–2
and block, 160
animate block, 161
    applying pictures, 79
    creating animation, 79
    example, 77
animation, 77–82
    faceplates, 165
    hit testing, 168
animation block diagrams, 298
Animation blocks
    animate, 78–81, 161
    lineDraw, 81–82, 205
Annotation blocks
    bezel, 165
    comment, 172
    date, 178
    index, 190
    label, 198
    scalarToVec, 237
    variable, 128–32, 260
    vecToScalar, 260
    wirePositioner, 261
Arithmetic blocks
    * (multiply), 142
    / (divide), 145
    1/X (inverse), 156
    abs, 157
    convert, 173
    gain, 186
    pow, 224
    sign, 238
    summingJunction, 245
    unitConversion, 256
    -X (negate), 143
arithmetic expressions, entering, 18
asin block, 161
atan2 block, 162
auto-panning, 4

B

backward Euler (Stiff) algorithm, 38
band specifications, 105
bar graphs, 75–77
bessel block, 163
Bessel filters, 101, 102
bezel block, 165
biophysical block diagrams, 298
bitmap library, 305
block diagrams
    adding, 128
    aerospace, 298
    animation, 298
    annotating, 172, 178, 198
    biophysical, 298

block diagrams (*continued*)
- business, 298
- chemical engineering, 299
- control design, 300
- copying to other applications, 29
- creating, 7
- DSP, 301
- dynamical systems, 300
- electro-mechanical, 301
- embedding, 127
- environmental, 301
- logic, 303
- motion, 301
- opening, 10
- optimization, 302
- power, 302
- previewing, 11
- printing, 11–13
- process control, 302
- protecting, 135–36
- samples, 298–301
- saving, 11
- setting up, 7
- signal processing, 303
- state machines, 303
- statistics, 134
- viewing, 266

Block Labels (View), 267

block level functions
- *userBlock*(), 279
- *userBlock*Event(), 280
- *userBlock*PA(), 283
- *userBlock*PC(), 283
- *userBlock*PI(), 283
- *userBlock*SE(), 283
- *userBlock*SS(), 283

block properties
- arithmetic expressions, 18
- C expressions, 18
- numeric data, 18
- precision control, 18–19
- setting up, 17–18

Block Properties (Edit), 17

blocks
- != (not equal to), 152
- * (multiply), 142
- / (divide), 145
- <, 147
- <=, 148
- == (equal to), 149
- >, 153
- >=, 155
- 1/X (inverse), 156
- abs, 157
- acos, 159
- aligning, 29
- and, 160
- animate, 78–81, 161
- asin, 161
- atan2, 162
- bessel, 163
- bezel, 165
- buffer, 166
- button, 168
- case, 169
- changing text attributes, 267
- comment, 172
- compound, 123
- connector tabs. *See* connector tabs
- const, 172
- constraint, 83, 173
- convert, 173
- copying, 26–29
- cos, 173
- cosh, 174
- cost, 89, 174
- crossDetect, 175
- custom, 16
- date, 178
- DDE, 119–22, 178
- DDEreceive, 115–17, 178
- DDEsend, 117–19, 178
- deadband, 179
- deleting, 32
- derivative, 179
- display, 180
- dotProduct, 181
- embed, 127, 138, 181

blocks (*continued*)
- error, 181
- exp, 182
- export, 111–14, 182
- expression, 182
- extensibility, 260, 275–88
- fft, 184
- finding, 30–32
- flipping, 29
- gain, 186
- gaussian, 188
- globalConstraint, 188
- histogram, 73–75, 188
- ifft, 189
- import, 109–12, 190
- index, 190
- inserting, 15
- int, 192
- integrator, 51
- integrator (1/S), 192
- invert, 198
- label, 198
- light, 199
- limit, 201
- limitedIntegrator (1/S), 202
- lineDraw, 81–82, 205
- ln, 206
- log10, 205
- map, 206
- max, 215
- merge, 217
- meter, 75–77, 218
- min, 218
- moving, 26–29
- multiply, 220
- neuralNet, 221
- not, 221
- or, 222
- parabola, 223
- parameterUnknown, 89, 224
- plot, 59–67, 224
- pow, 224
- PRBS, 226
- properties. *See* block properties
- pulseTrain, 226

blocks (*continued*)
- quantize, 227
- ramp, 229
- realTime, 229
- relay, 230
- replacing, 30–32
- resetIntegrator (1/S), 231
- rt-DataIn, 234
- rt-DataOut, 234
- sampleHold, 235
- scalarToVec, 237
- selecting, 25
- setting up. *See* block properties
- sign, 238
- sin, 239
- sinh, 240
- sinusoid, 240
- slider, 241
- sqrt, 242
- stateSpace, 51, 54, 242
- step, 244
- stop, 244
- stripChart, 67–73, 245
- summingJunction, 245
- tan, 247
- tanh, 248
- timeDelay, 249
- transferFunction, 48, 51, 54, 99, 100
- transferFunction, 252
- transpose, 255
- types, 16
- uniform, 256
- unitConversion, 256
- unitDelay, 98
- unitDelay (1/Z), 51, 52, 257
- unknown, 83, 259
- userFunction, 260, 275–88
- user-written, 260, 275–88
- variable, 260
- vecToScalar, 260
- vsum, 261
- wirePositioner, 261
- wiring. *See* wiring blocks
- -X (negate), 143
- xor, 261

Blocks menu. *See also* individual blocks
- adding blocks to, 288

Boolean blocks
- !=, 152
- <, 147
- <=, 148
- ==, 149
- >, 153
- >=, 155
- and, 160
- not, 221
- or, 222
- xor, 261

buffer block, 166
built-in variables, 132
business block diagrams, 298
Butterworth filters, 101, 102
button block, 168

C

C expressions, entering, 18
case block, 169
Chebyshev filters, 101, 102
check boxes, 6
checkpointing simulations, 39
checkpointing system states, 193, 203, 232, 257, 280
chemical engineering block diagrams, 299
choosing commands, 4
Clear (Edit), 32
Clear Errors (Edit), 57
Close button, 2
closed-loop system, 45–48
coloring
- plotting background, 267
- screen display, 267
- text, 267
- window, 267
- wires, 267

Colors (View), 21, 267
command buttons, 6
comment block, 172
- retaining character format, 267

component libraries
- DSP, 292

component libraries (*continued*)
- dynamical systems, 292
- electrical systems, 292
- electro-mechanical systems, 292
- hydraulic systems, 293–94
- process control, 295
- thermal control, 295
- turbine, 295

compound blocks
- basics, 133
- coloring, 267
- configuring pictures on, 125
- connector labels, 126, 267
- connector tabs, 124
- containing global variables, 131
- creating, 124, 127
- dissolving, 126
- drilling, 124
- embedding, 127
- hiding, 125
- protecting, 136–37
- selecting, 25

Connector Labels (View), 126, 267
connector tabs, 19
- adding and removing, 22, 280
- coloring, 266
- connection class, 23
- in display mode, 24
- in presentation mode, 24
- labels, 126, 267
- manipulating, 22
- on compound blocks, 124
- probing, 56
- restricting access, 24
- symbols, 22, 24
- unconnected inputs, 23

const block, 172
constraint block, 83, 173, 174
Continue (Simulate), 55
continuous simulations, 44–51
- spring-damper arm, 45–48
- spring-damper arm with external force example, 50–51

continuous time transfer function, 48–49, 50–51
control design block diagrams, 300

Control Panel, 54
Control Panel (View), 55
controls toolbox functions, 289
convert block, 173
Copy (Edit), 28
copying
    block diagrams, 29
    character formatting, 267
copying blocks
    drap-and-drop editing, 27
    rules, 27
    with Clipboard, 28
cos block, 173
cosh block, 174
cost block, 89
cost function, 89
Create Compound Block (Edit), 124, 127
crossDetect block, 175
Cut (Edit), 28
cutting and pasting blocks
    rules, 27
    with Clipboard, 28
    with drag-and-drop, 27

D

data I/O, 182
Data Types (View), 266
date block, 178
.DAT files, exporting, 182
DDE
    creating links, 114
    editing a link address, 122
    linking from another app, 115–17
    linking to another app, 117–19
    two-way links, 119–22
DDE block, 119–22, 178
DDE blocks
    DDE, 119–22, 178
    DDEreceive, 115–17, 178
    DDEsend, 117–19, 178
DDEreceive block, 115–17, 178
DDEsend block, 117–19, 178
deadband block, 179
definition and below scope, 130, 131
degMsg(), 286
deleting blocks, 32
derivative block, 179
Diagram Information (File), 134, 135
diagram scope, 130, 131
diagram tree
    expand and collapse, 4
    jump to location, 4
    resizing, 4
dialog boxes, 6–7
    missing, 5
difference equations, 52
differential equations, 45–48
    converting to integration, 44
differentiators, 104
digital filters. *See* filter design
discrete MIMO systems, 52
discrete time simulations, 51–53
    entering difference equations, 52
    multi-rate, 52
display block, 180
display boxes, 7
Display Mode (View), 22, 266
Dissolve Compound Block (Edit), 126
divide (/) block, 145
DLL functions, 260
    adding to Blocks menu, 288
    building with Project Build, 276
    calling, 287
    criteria, 276
    debugging, 287
    VisSim calling conventions, 277
dotProduct block, 181
drop-down list boxes, 6
DSP block diagrams, 301
DSP components, 292
Dynamic Link Library functions. *See* DLL functions
dynamical system components, 292
dynamical systems block diagrams, 300

E

electrical system components, 292
electro-mechanical block diagrams, 301
electro-mechanical system components, 292
electro-mechanical toolbox functions, 290

embed block, 127, 181
  editing, 128
  protecting, 138–39
  reconnecting, 128
environmental block diagrams, 301
equal to (==) block, 149
error block, 181
error conditions, 57, 181
error conditions, resetting, 57, 181
Euler algorithm, 37
event handler function, 280
Exit (File), 13
exp block, 182
export block, 111–14, 182
exported functions
  debMsg(), 286
  getSimTime(), 286
  getSimTimeStep(), 286
  setBlockErr(), 286
  setUserBlockMenu(), 286
  stopSimulation(), 286
  vissimRequest(), 284
expression block, 182
extending the block set, 260, 275–88

F

feedback loops, 58, 84
fft block, 184
FFT plots, 62
FFT strip charts, 69
fields, 9
filter design
  comparison, 100
  FIR, 104–7
  frequency domain, 99
  IIR, 99, 100–103
  operations, 98
  time domain, 98–99
  with transferFunction block, 100–107
Find (Edit), 30–31
  Match Variable Definitions Only, 132
finding blocks, 30–31
FIR filters, 100, 104–7
  band specifications, 105
  continuous, 104
FIR filters (*continued*)
  discrete, 104
  generating, 106
  order, 105
  tapped delay implementation, 105
flag error function, 286
Fletcher Reeves global optimizer, 91
flexWires. *See* wiring blocks
  unbundling, 260
Flip Horizontal (Edit), 29
Fonts (View), 267
footers, 8
frequency domain filters, 99
frequency domain plots, 62
frequency domain strip charts, 69

G

gain block, 186
gauges, 75–77
gaussian block, 188
general information function, 284
get current simulation time function, 286
get current simulation time step function, 286
getSimTime(), 286
getSimTimeStep(), 286
global optimization
  avoiding system instability, 96
  example, 91
  tips, 96
global optimizers
  building, 273
  Fletcher Reeves, 91
  Polak-Ribiere, 91
  Powell, 91
  source files, 268
  vissimRequest function, 271
global variables, 131
globalConstraint block, 188
Go (Simulate), 55
greater than (>) block, 153
greater than or equal to (>=) block, 155
grids, snapping to, 29

H

headers, 8
Hilbert transfromers, 104
histogram block, 73–75, 188
  sizing, 74
hybrid simulations, 54
hydraulic system components, 293–94

I

ifft block, 189
IIR filters, 99, 100–103
  analog filter method, 101
  attenuation frequencies, 102
  attenuation levels, 102
  band pass specification, 102
  Bessel, 101, 102
  Butterworth, 101, 102
  Chebyshev, 101, 102
  cut-off frequencies, 102
  generating, 103
  Inverse Chebyshev, 101, 102
implicit equations
  convergence warnings, 85
  diagrams including, 302
  feedback loops, 84
  setting up, 83
  solving, 84
implicit solver
  building with nmake, 270
  error tolerance, 85
  maximum iteration count, 85
  Newton-Raphson, 85
  perturbation, 85
  relaxation, 85
  source files, 268
  user-defined, 85
  vissimRequest function, 268–69
import block, 109–12, 190
importing data
  map block, 206
  transferFunction block, 252
  with stateSpace block, 242
index block, 190
initial conditions, 17, 193, 203, 223, 235, 29, 232, 240, 249, 257
initialization function, 278
inserting blocks, 15
int block, 192
integration algorithms
  adaptive Bulirsh-Stoer, 38
  adaptive Runge Kutta 5th order, 38
  backward Euler (Stiff), 38
  choosing, 36–38
  Euler, 37
  input functions, 58
  integrator block, 192
  iteration count, 38
  limitedIntegrator block, 202
  resetIntegrator block, 231
  Runge Kutta 2d order, 37
  Runge Kutta 4th order, 38
  selecting, 35
  step sizes, 35, 38
  trapezoidal, 37
  truncation errors, 38
  unstable settings, 58
Integration blocks, 44
  integrator, 192
  limitedIntegrator, 202
  resetIntegrator, 231
integrator (1/S) block, 51, 192
inverse (1/X) block, 156
inverse Chebyshev filters, 101, 102
invert block, 198

L

label block, 198
landscape paper orientation, 8
less than (<) block, 147
less than or equal to (<=) block, 148
level scope, 130, 131
light block, 199
limit block, 201
limitedIntegrator (1/S) block, 202
Linear System blocks
  stateSpace, 242
  transferFunction, 48, 100, 252
linearization, 242, 252

linearization data
- .M file, 242, 252
- .MAT file, 243, 252

lineDraw block, 81–82, 205
ln block, 206
local variables, 131
log10 block, 205
logic diagrams, 303

M

map block, 206
.MAT files
- exporting, 182
- importing, 243, 252

MatLab
- exporting data to, 182
- importing data from, 243, 252

Matrix Operation blocks
- buffer, 166
- dotProduct, 181
- fft, 184
- ifft, 189
- invert, 198
- multiply, 220
- transpose, 255
- vsum, 261

max block, 215
Maximize button, 2
menu bar, 2
merge block, 217
meter block, 218
- coloring, 267
- properties, 76–77
- sizing, 75
- types, 75

.M files
- exporting, 182
- importing, 242, 252

min block, 218
Minimize button, 2
MMI
- audio and visual alarms, 199
- multiple bitmap animation, 199

motion control block diagrams, 301
moving blocks
- drag-and-drop, 27
- rules, 27
- with Clipboard, 28

multiply (*) block, 142
multiply (matrix) block, 220
multi-rate simulations, 52

N

negate (-X) block, 143
neuralNet block, 221
New (File), 7–9
Newton-Raphson solver, 85
noise
- random, 188
- uniform, 256

Nonlinear blocks
- case, 169
- crossDetect, 175
- deadband, 179
- int, 192
- limit, 201
- map, 206
- max, 215
- merge, 217
- min, 218
- quantize, 227
- relay, 230
- sampleHold, 235

not block, 221
not equal to (!=) block, 152

O

Open (File), 10
opening
- a new diagram, 7–9
- an existing diagram, 10
- fast opening, 10
- read-only access, 10

optimization. *See* global optimization; implicit equations; implicit solver; Optimization Properties (Simulate)
optimization block diagrams, 302

Optimization blocks
- constraint, 83, 173
- cost, 174
- globalConstraint, 188
- parameterUnknown, 224
- unknown, 83, 259

Optimization Properties (Simulate), 90
or block, 222

P

Pad*e* approximations toolbox functions, 290
page setup
- fields, 9
- fit to page, 8
- headers and footers, 8
- margins, 8
- orientation, 8, 13
- paper size, 8
- paper source, 8
- tiling, 8
- with Print command, 12

Page Setup (File), 7
panning, 4
parabola block, 223
parameter allocation function, 283
parameter change function, 283
parameter initialization function, 283
parameters. *See also* block properties
parameterUnknown block, 89, 224
passwords, 134
Paste (Edit), 28
Paste Link (Edit), 115
Paste Link (Edit), 119
path alias, 132
- creating, 132
- inserting, 133

plot block, 224
- coloring, 267
- sizing, 60
- zooming, 61

plot properties
- appearance, 67
- axis divisions, 67
- axis labels, 66
- data points, 64

plot properties (*continued*)
- decibel Y, 65
- external trigger, 63
- FFT, 62
- fixed bounds, 62, 66
- geometric markers, 63
- grid lines, 65
- line types, 64, 70
- log plots, 64
- max data points, 64
- multi XY traces, 64
- overplotting, 62
- reading coordinates, 65
- retracing, 67
- signal labels, 66
- time domain, 60
- time scaling, 66
- titles and subtitles, 66
- truncating FFT data, 62
- XY plots, 63

Polak-Ribiere global optimizer, 91
portrait paper orientation, 8
pow block, 224
Powell global optimizer, 91
power block diagrams, 302
PRBS block, 226
Preferences (Edit)
- Addons, 288
- Path Aliases, 132
- Preferences
  - Auto Connect Radius, 21
  - Color Compound Block, 267
  - Color Display, 267
  - High Precision Display, 18–19
  - Show Horizontal Scroll Bar, 264
  - Show Vertical Scroll Bar, 264
  - Snap To Grid, 11, 29
  - Training Mode Labels, 267
  - Use Rich Text Format, 172, 267

presentation mode
- activating, 266
- affect on connector tabs, 24

Presentation Mode (View), 266
previewing block diagrams, 11

Print (File)
- All, 12
- Copies, 12
- Current Level, 12
- Current Level and Below, 12
- Fit to Page, 12
- orientation, 13
- paper size, 13
- paper source, 13
- Print to File, 12
- printer, 12
- Tile Pages, 12

print debug message function, 286
Print Preview (File), 11
printing, 11
- fit to page, 8
- landscape mode, 13
- previewing, 11
- strip charts, 68
- tiling, 8

probing connector values, 56
process control block diagrams, 302
process control components, 295
protecting
- block diagrams, 10, 135–36
- compound blocks, 136–37
- embedded block diagrams, 138–39

pulseTrain block, 226

Q

quantize block, 227

R

radio buttons, 7
ramp block, 229
Random Generator blocks
- gaussian, 188
- PRBS, 226
- uniform, 256

random numbers, generating, 40
range control
- setting, 33

Real Time blocks
- rt-DataIn, 234
- rt-DataOut, 234

Real Time Config (File), 234
realTime block, 229
real-time simulations, 35
relay block, 230
Remez Multiple Exchange algorithm, 104
Remove Connector (Edit), 22
Rename Block (Edit), 125
Repaint Screen (Edit), 10
Replace (Edit), 31
replacing blocks, 31
request simulation end function, 286
Reset (Simulate), 56
resetIntegrator (1/S) block, 231
right mouse button, 18
rt-DataIn block, 234
rt-DataOut block, 234
Runge Kutta 2d order algorithm, 37
Runge Kutta 4th order algorithm, 38

S

sampleHold block, 235
Save (File), 11
Save As (File), 11
saving block diagrams, 11
saving system states, 39, 57
scalarToVec block, 237
scoping, 130
screen, coloring, 267
scroll bars, 4
scrolling lists, 7
selecting blocks
- area select, 26
- canceling selections, 26
- compound blocks, 25
- individually, 25
- toggle select, 26

setBlockErr(), 286
setUserBlockMenu(), 286
sign block, 238
signal, 303

Signal Consumer
- display, 180
- error, 181
- export, 111–14, 182
- histogram, 73–75, 188
- light, 199
- meter, 75–77, 218
- plot, 59–67, 224
- stop, 244
- stripChart, 67–73, 245

signal generation toolbox functions, 291
Signal Producer
Signal Producer blocks
- button, 168
- const, 172
- import block, 109–12, 190
- parabola, 223
- pulseTrain, 226
- ramp, 229
- realTime, 229
- sinusoid, 240
- slider, 241
- step, 244

simulation end function, 283
simulation level functions
- vsmEvent(), 278
- vsmInit(), 278

Simulation Properties (Simulate)
- Defaults, 41
- Integration Method
  - Adaptive Bulirsh-Stoer, 38
  - Adaptive Runge Kutta 5th order, 38
  - Backward Euler (Stiff), 38
  - Euler, 37
  - Max Iteration Count, 38
  - Max Truncation Error, 38
  - Min Step Size, 38
  - Runge Kutta 2nd order, 37
  - Runge Kutta 4th order, 38
  - Trapezoidal, 37
- Preferences
  - Check Connections, 39
  - Checkpoint State, 39
  - Frequency Units, 39, 103
  - Notify Simulation End, 40
- Preferences (*continued*)
  - Propagage Integer Types, 40
  - Raise Real-Time Priority, 40
  - Random Seed, 40
  - Warn Nonintegral Clock, 40
  - Warn Nonintegral Delay, 40
  - Warn Numeric Overflow, 40
- Range
  - Auto Restart, 34
  - End, 35
  - Retain State, 34
  - Run in Real Time, 35
  - Start, 35
  - Step Size, 35

simulation start function, 283
simulations
- animated, 199
- auto restart, 34
- basics, 43
- checkpointing, 39
- continuing, 55
- continuous, 44–51
- Control Panel, 54
- default settings, 41
- discrete, 51–53
- dynamic execution, 56
- end-of-sim message, 40
- error conditions, 57
- hybrid, 54
- integration algorithms, 35–38
- iterations, 38
- multi-rate, 52
- preferences, 38–40
- raising process priority, 40
- random numbers, 40
- range control, 33
- real-time, 35
- resetting, 56
- single-stepping, 56
- snapping system states, 57
- starting, 55
- step sizes, 35, 38
- stopping, 55, 244
- trimming a system, 57
- truncation errors, 38

simulations (*continued*)
  unconnected blocks, 58
sin block, 239
sinh block, 240
sinusoid block, 240
slider block, 241
Snap State (Simulate), 57
sound files
  exporting, 182
  playing, 182, 199
spring-damper arm example, 45–48
  modified, 50–51
sqrt block, 242
starting VisSim, 1, 263
state machine diagrams, 303
stateSpace block, 51, 54, 242
state-space matrices, 242, 252
status bar, 3
step block, 244
step sizes, 35, 38
Stop (Simulate), 55
stop block, 244
stopSimulation(), 286
strip chart properties
  appearance, 73
  axis divisions, 72
  axis labels, 71
  decibel Y, 70
  displayed time, 72
  external trigger, 70
  FFT, 69
  fixed bounds, 69, 72
  geometric markers, 69
  grid lines, 70
  Log Y, 70
  max data points, 70
  scroll back interval, 72
  signal labels, 71
  time domain, 68
  scaling, 72
  titles and subtitles, 71
strip charts. *See* stripChart block
stripChart block, 67–73, 245
  coloring, 267
  printing, 68

stripChart block (*continued*)
  properties. *See* strip chart properties
  sizing, 68
summingJunction block, 245
system states
  checkpointing, 39
  retaining, 34
  snapping, 57
system-wide event function, 278

T

tabs, 6
tan block, 247
tanh block, 248
text boxes, 6
thermal control components, 295
Time Delay blocks
  timeDelay, 249
  unitDelay (1/Z), 257
time domain filters
  with transferFunction block, 99
  with unitDelay block, 98
time domain plots, 60
time domain strip charts, 68
time step function, 279
timeDelay block, 249
title bar, 2
titles, block diagram, 10, 134
toolbar
  dimmed buttons, 3
  editng, 265
  hiding, 264
  moving, 2
Toolbar (Edit), 265–66
Toolbar (View), 2, 264
toolboxes
  controls, 289
  Electro-mechanical, 290
  Pad*e* approximations, 290
  signal generation, 291
  tools, 291
tools toolbox functions, 291
training mode labels, 267

Transcendental blocks
- acos, 159
- asin, 161
- atan2, 162
- bessel, 163
- cos, 173
- cosh, 174
- exp, 182
- ln, 206
- log10, 205
- sin, 239
- sinh, 240
- sqrt, 242
- tan, 247
- tanh, 248

transfer functions, 50–51
- continous time, 48–49
- transferFunction block, 252
- with polynomial constants, 48–49

transferFunction block, 48, 51, 54, 99, 100, 252
transpose block, 255
trapezoidal algorithm, 37
trimming a system, 57
turbine components, 295

U

Undo (Edit), 10
uniform block, 256
unitConversion block, 256
unitDelay block, 51, 52, 98, 257
unknown block, 83, 259
userBlock(), 279
*userBlock*Event(), 280
*userBlock*PA(), 283
*userBlock*PC(), 283
*userBlock*PI(), 283
*userBlock*SE(), 283
*userBlock*SS(), 283
userFunction block, 260. *See also* DLL functions
user-written blocks, 260. *See also* DLL functions

V

variable block
- built-in, 132

variable block (*continued*)
- creating, 129
- finding definitions, 132
- scoping, 130
- using, 129, 260

vector wires, 237, 260. *See* wiring blocks
vecToScalar block, 260
views
- block labels, 267
- diagram text, 267
- display mode, 266
- presentation mode, 266
- screen display, 267
- training mode labels, 267

VisSim
- customizing, 13, 266
- exiting, 13
- starting, 1, 263

VisSim Viewer, 313–16
- distributing, 313
- distribution license agreement, 316
- documentation, 313
- end user license agreement, 315
- installing, 314
- starting, 314

vissimRequest function, 268–69, 271–72, 284
vsmEvent(), 278
vsmInit(), 278
VSOLVER.C, 271
VSOLVER.DEF, 271
VSOLVER.MAK, 271
vsum block, 261
VSUSER.H, 271

W

.WAV files
- exporting, 182
- playing, 182, 199

windows, color, 267
wirePositioner block, 21, 261
wires
- bundling, 237
- coloring, 267
- positioning, 261

wiring blocks
- auto-connect, 21
- bundling flexWires, 20
- colors, 21
- creating, 20
- deleting, 22
- flexWires, 19
- hiding, 22
- positioning, 21
- rules of use, 20
- unbundling flexWires, 20
- unconnected blocks, 58
- vector wires, 19, 260
- wireless blocks, 260

X

-X (negate) block, 143
xor block, 261
XY plots, 63