# MATRIX$_X$® 7.0

## GETTING STARTED GUIDE

WINDOWS VERSION

WindRiver®

**Corporate Headquarters**
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): 800/545-WIND
telephone: 510/748-4100
facsimile: 510/749-2010

For additional contact information, please visit the Wind River URL:

**http://www.windriver.com**

For information on how to contact Customer Support, please visit the following URL:

**http://www.windriver.com/support**

*MATRIX$_X$ Getting Started Guide, Windows Version 7.0*
Edition 1
16 Nov 00
Part #: 000-0119-006

# Contents

# 1
# *Introduction: The MATRIX$_X$ Product Family*

This chapter contains an overview of the MATRIX$_X$ Product Family. It concludes with a section on the organization and use of this document.

## *MATRIX$_X$ Product Family*

The MATRIX$_X$ Product Family includes the following:

**Xmath —** The mathematics and system analysis environment of the MATRIX$_X$ Product Family. See *1.1 Xmath*, p.3.

**SystemBuild —** A graphical programming environment that uses a hierarchically structured block diagramming paradigm for modeling and simulation of linear and nonlinear dynamic systems. See *1.2 SystemBuild*, p.3.

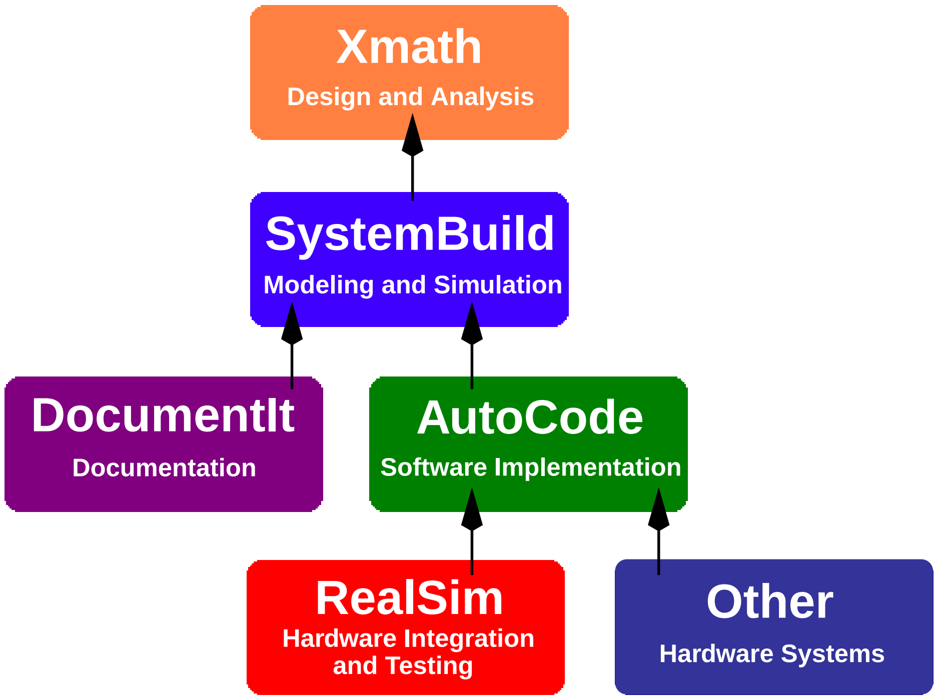
**AutoCode —** Template technology used to process SystemBuild model files to produce C or Ada code. See *1.3 AutoCode*, p.4.

**DocumentIt —** TPL template technology (similar to AutoCode) used to capture information from SystemBuild model files and then format it to create documentation. See *1.4 DocumentIt*, p.4.

**RealSim —** Real-time software and hardware combination that enables you to do real-time simulations of feedback control system models designed in SystemBuild. See *1.5 RealSim*, p.4.

Figure 1-1 shows product dependencies of the MATRIX$_X$ Product Family.

Figure 1-1 **MATRIX$_X$ Product Family Overview with Dependencies**



The MATRIX$_X$ Product Family core software must be installed according to the *MATRIX$_X$ System Administrator's Guide, Windows Version.*

- SystemBuild users must have Xmath.
- AutoCode and DocumentIt users need SystemBuild and Xmath.
- RealSim users must have AutoCode, SystemBuild, and Xmath.

## 1.1 Xmath

The Xmath software environment facilitates system analysis and visualization. Xmath contains over 700 predefined functions and commands, interactive color graphics, and a programmable graphical user interface (PGUI). The MathScript scripting language simplifies command and function programming. Object-oriented design provides convenient data management and speeds program execution. The structure and capabilities of Xmath are discussed in the *Xmath User's Guide*, while the Xmath online Help provides easy access to Xmath commands and functions.

- Xmath commands support basic operations such as creating, plotting, saving, and loading data, and accessing online Help. *3.1 Introduction to Xmath*, p.31 describes the capabilities of Xmath and its modules.
- Xmath commands provide access to SystemBuild and its related products. Xmath handles data for SystemBuild and all other products in the MATRIX$_X$ Product Family.

## 1.2 SystemBuild

SystemBuild visual modeling and simulation software lets you model many kinds of systems, from control loops to complex vehicle applications. You can use SystemBuild to prepare models that can be simulated with the SystemBuild simulator. Built-in simulation tools let you interactively verify, test, and modify system models.

To create a model, you can use all of the SystemBuild standard and optional features. BetterState blocks facilitate the integration of hierarchical state transition models. The optional Interactive Animation (IA) module or the Altia Design module adds the ability to control your model interactively during simulation. With IA, the icons are put in one or more picture files (**.pic**) while Altia images are stored in design files (**.dsn**).

For additional information, see *4. SystemBuild*.

## 1.3 AutoCode

AutoCode is an automatic code generator for SystemBuild models. The AutoCode software processes SystemBuild model files you create and outputs compilable ANSI C or Ada code.

The output code can be compiled to produce a stand-alone real-time executable program suitable for running in a test-bed environment or for use in an embedded real-time system. Advanced template programming language (TPL) template technology provides a powerful programming capability to tailor nearly any part of the generated code to specialized needs. For additional information, see *5. AutoCode*.

## 1.4 DocumentIt

DocumentIt is an automated documentation generator for SystemBuild models. This module integrates documentation with SystemBuild design activity for easier and more accurate manuals and reports. Templates are included for FrameMaker, Microsoft Word, and WordPerfect markup formats. Using TPL, you can capture and tailor any part of the generated document for special documentation standards or other needs.

For additional information, see *6. DocumentIt*.

## 1.5 RealSim

The RealSim controller lets you do real-time simulations of feedback control system models designed in SystemBuild. In this way, you can see how a prototype will perform in the “real” world before actually building the prototype.

The RealSim environment lets you run models developed in SystemBuild in real time, connecting to real hardware for real-time simulation, rapid prototyping, and hardware-in-the-loop modeling. Run-time graphical user interfaces can be built to let you monitor values and change setpoints in the application running on a real-time computer, These are done in the same manner as interactive simulations in SystemBuild. In addition to the software tools, real-time computers with analog and digital I/O are available to complete the RealSim environment.

For additional information, see *7. RealSim*.

## 1.6 Using This Manual

This manual acquaints you with the MATRIX$_X$ Product Family software. It provides an introduction to each software product and includes tutorials to assist you in learning key tasks.

### Organization

This manual is organized as follows:

- *1. Introduction: The MATRIX$_X$ Product Family* introduces each software product in the MATRIX$_X$ Product Family.
- *2. MATRIX$_X$ Publications, Online Help, and Customer Support* lists the MATRIX$_X$ Product Family publications available, describes how to use online help, explains the conventions used in MATRIX$_X$ online and printed books, and provides contact information for MATRIX$_X$ customer support.
- *3. Xmath* provides an overview of Xmath and the Xmath modules, and contains a tutorial.
- *4. SystemBuild* provides an overview of SystemBuild and the SystemBuild modules, and gives a tutorial that includes use of a BetterState statechart.
- *5. AutoCode* provides an overview of the AutoCode code generator.
- *6. DocumentIt* provides an overview of the DocumentIt document generator.
- *7. RealSim* provides an overview of the RealSim real-time simulator.

## Conventions

This publication makes use of the following types of conventions: font, format, symbol, mouse, and note. These conventions are detailed in *2. MATRIX$_X$ Publications, Online Help, and Customer Support*.

# 2 MATRIX$_X$ Publications, Online Help, and Customer Support

This chapter provides publication conventions and instructions for using MATRIX$_X$ online books and Help. It also contains an annotated list of the online books, and concludes with directions for obtaining release information and customer support.

## 2.1 Online and Printed Book Conventions

The MATRIX$_X$ online and printed books use several types of conventions: font, format, symbols, mouse, and levels of notes. These conventions are discussed in the sections that follow.

### 2.1.1 Font Conventions

This sentence is set in the default text font, Palatino, which is used for general text. Use of fonts and styles other than the standard text default is summarized in Table 2-1:

Table 2-1 **Font Conventions**

| Fonts and Styles | Use |
|---|---|
| Palatino | General text. Palatino is the default text font. |
| **bold Palatino** | **Bold Palatino** is used for command and function names, filenames, directory paths, and environment variables.<br><br>Xmath commands (for example, **SAVE**, **LOAD**, **SET**) are shown in uppercase, while Xmath functions are set in lowercase (for example, **random**, **plot**, **kronecker**) |
| *italic Palatino* | *Italic Palatino* is used for emphasis, first instances of terms defined in the glossary, publication titles, and chapter, section, and topic headings. |
| `Courier` | `Courier` is used for system output, code examples, prompt responses, and syntax examples. |
| **`bold Courier`** | **`Bold Courier`** is used for user input (anything you are expected to type and enter). |
| *`italic Courier`*<br>***`italic bold Courier`*** | Italic is also used in conjunction with *`Courier`* or ***`bold Courier`*** to denote placeholders in syntax examples or generic examples. |
| Helvetica | Helvetica is used for window and dialog names, menu selections, and named items in a window or dialog. Dialog messages and keyboard keys are also set in this font. |

### 2.1.2 Format Conventions

Xmath output appears in `Courier` directly below the **`bold Courier`** input (see Example 2-1). If the output is extremely large, continuation marks (... or :) are used to indicate continuation, or replace missing parts.

Example 2-1 **Xmath Sample Input and Output**

```
x=random(2,6)

x (a rectangular matrix) =

  0.827908    0.926234    0.566721    0.571164 ...
  0.559594    0.124934    0.727922    0.267777 ...

x'

ans (a rectangular matrix) =

  0.827908     0.559594
      :            :
      :            :
  0.0568928    0.988541
```

If the input is long, continuing lines of input are indented as shown in Example 2-2.

Example 2-2 **Sample Convention for Handling Longer Lines of Code**

```
Sys=system(makepoly([1,-1.63,5.5],"s"),
    makepoly([1,2.7,5.6,13.5,8.1],"s"))

Sys (a transfer function) =

          2
         s - 1.63s + 5.5
  -----------------------------
   4      3      2
  s + 2.7s + 5.6s + 13.5s + 8.1

initial integrator outputs
  0
  0
  0
  0
```

```
Input Names
-----------
Input 1

Output Names
------------
Output 1

System is continuous
```

## 2.1.3 Symbol Conventions

Symbols used in this manual include those shown in Table 2-2:

Table 2-2 **Symbol Conventions**

| Symbol | Use |
|---|---|
| % | UNIX® operating system prompt for C shell. Xmath input shows no prompt (as you will usually be typing in the Xmath Commands window command area). |
| $ | UNIX operating system prompt for Bourne and Korn shells. |
| { } | Braces denote optional arguments or keywords in Xmath syntax. For example: `[out1,out2]=fun(in1,in2,{in3,keywords})` |
| [ ] | Brackets indicate that the enclosed information is optional. The brackets are generally not typed when the information is entered. |
| \| | A vertical bar separating two text items indicates that either item can be entered as a value. |
| → | Hierarchical menu selections are indicated with arrows: In the Xmath main menu select File→Load to load a model or demo. The arrow is also used to specify hierarchical structure in the online Help Topics Hierarchy pane, and in the topics index: See the MATRIX$_X$ online Help *plot→linestyles* topic. |

### 2.1.4 Mouse Conventions

This document assumes you have a standard, right-handed 2- or 3-button mouse. From left to right, the buttons are referred to as MB1, MB2, and MB3 for a right-hand mouse definition; these buttons are right to left for a left-hand mouse definition. For workstations with a two button mouse, MB1 is the left button and usually the right button behaves as MB3.

All instructions assume MB1 unless otherwise noted. Some common mouse instructions are shown in Table 2-3:

Table 2-3 **Common Mouse Instructions**

| Instruction | Use |
|---|---|
| click | Press, then quickly release MB1. |
| double-click | Rapidly click MB1 twice. |
| drag | Hold down MB1 while moving the mouse; release the button when the desired result is obtained. |

The following mouse-click combinations are useful for selecting text:

- To select a word, point anywhere within the desired word and double–click.
- To select an entire line, point anywhere on the line and triple-click.
- To select all text in an Xmath window area, move the cursor into the area and quadruple-click.

### 2.1.5 Note, Caution, and Warning Conventions

Within the text of this manual, you may find notes, cautions and warnings:

**NOTE:** Note indicates information that emphasizes or supplements important points of the main text. A note may supply information that applies only in special cases, or details that apply only to specific releases of the product.

**CAUTION:** Caution advises that failure to take or avoid a specific action could result in loss of data.

**WARNING:** Warning advises that failure to take or avoid a specific action could result in physical damage to the user or hardware.

## 2.2 Using Online Books

The MATRIX$_X$ CD-ROM contains software, documentation in PDF format, and a copy of Adobe® Acrobat® Reader with Search. With Acrobat Reader you can view, search, and print any document on the MATRIX$_X$ CD-ROM.

Using Online Books describes viewing, printing, and searching the online PDF files.

If you do not already have Acrobat Reader with Search installed, see the *MATRIX$_X$ System Administrator's Guide, Windows Version* for installation details. The Search feature provides a full-text search across all documents on the CD-ROM. If you don't have the Search capability, we recommend that you install the version provided on the MATRIX$_X$ CD-ROM.

To determine whether your copy of Acrobat Reader includes Search, start it, and examine the toolbar (select Window→Show Tool Bar if the toolbar isn't visible). If Search is installed, the toolbar ends with the four search buttons shown in *Find and Search in PDF*, p.14.

### 2.2.1 Viewing, Printing, and Searching PDF Files

To view the documentation, launch Adobe Acrobat Reader with Search (version 3.0 or higher); then open the file **welcome.pdf**. If you do not have Acrobat on your system, see the *MATRIX$_X$ System Administrator's Guide, Windows Version* for installation details.

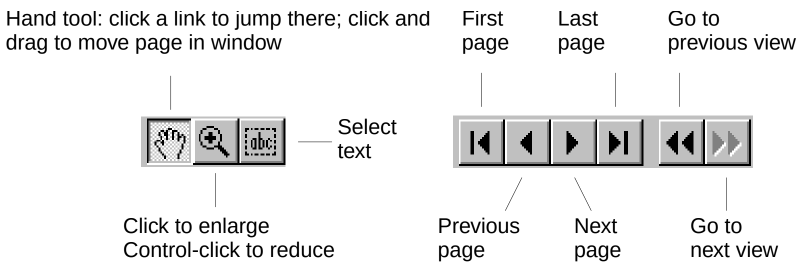
The **welcome.pdf** file is an overall table of contents for PDF documentation on the CD-ROM. Click any document title to open that document.

#### Using Acrobat Reader

Each document has a "bookmarks" pane displayed on the left.

All bookmarks and blue text are hypertext links. To follow a link, be sure the hand tool is selected; then click the bookmark or blue text.

Use the following Acrobat toolbar buttons for browsing and navigation:


Hand tool: click a link to jump there; click and drag to move page in window
First page
Last page
Go to previous view
Select text
Click to enlarge Control-click to reduce
Previous page
Next page
Go to next view

Select Help → Reader Guide for a detailed description of all Acrobat capabilities.

Bookmarks can contain the following links:

- Welcome: Link to the Welcome screen.
- Document Title: Link to the cover of the current document.
- Contents: Link to the Table of Contents for the current document.
- Chapter and Section Bookmarks: Links to chapters and sections in the document.
- A right-pointing triangle in front of a bookmark means there are sub-bookmarks; click the triangle to expand to lower-level bookmarks.
- Index: Link to the Index of the current document if one exists.

### *Pasting Text Into Other Applications*

To copy examples or text from a PDF document into an application, first click Tools → Select Text, or click the abc button. Select the text, and then use the usual technique on your platform to copy and paste the text into the target application.

### *ATTENTION: Copy and Paste Known Problem*

In some cases, example text can contain special typeset characters, such as a non-breaking space, or special left- or right-facing delimiters (“_”, ‘_’), etc., that will not be properly parsed when pasted into an application. If you receive an error message, retype the special characters, and the input will be processed. The following string is an example:

```
plot(a,b, { xlab = “label_string” })
```

If the application does not understand the paired double quotes (“_”) you need to retype them so the input contains straight quotes:

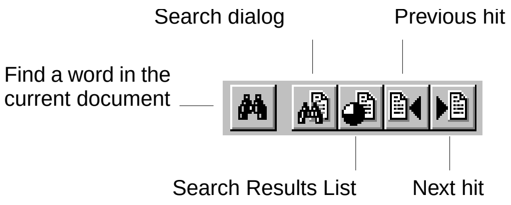
```
plot(a,b, { xlab = "label_string" })
```

## Printing Documents

To print a file, select File → Print and then specify the desired pages. Note that the PDF page numbers appearing at the bottom of the Acrobat screen count every page, including the cover, the table of contents, and so forth. Be sure to use these page numbers (rather than the document page numbers) when printing a range of pages.

**NOTE:** You must print PDF files on a PostScript printer.

## Find and Search in PDF



The Acrobat Reader Find feature locates words or word phrases in the current PDF document.

To use Find, click the plain binoculars on the toolbar, or select Tools → Find.

To make a full-text search over all documents on the CD-ROM, use the Search tool. Click the binoculars overlaid on a document on the toolbar, or select Tools → Search. In the Search dialog, enter the word or phrase you want to find, select options as desired, and then click Search. Search displays a Search Results List dialog of documents containing the term. Click any document in the list to open it. All instances of the term on the first page on which it occurs are highlighted.

To find the next or previous occurrence, click the Next or Previous hit buttons on the Acrobat toolbar. To return to the Search Results List dialog, click on the Search Results List button.

The Search command includes powerful features for expanding a search using automatic word-stemming, a thesaurus to find synonyms, and a “sounds like” feature. You can also use wild cards in terms, control case matching, and include Boolean connectives.

For further details on Search, select Help → Plug-In Help → Using Acrobat Search to read the online guide.

### ATTENTION: Known Search Index Problem

The Search feature uses a search index file. When you open **welcome.pdf** or any other document, Acrobat automatically “attaches” the required index file.

Acrobat sometimes attaches the index file more than once. Then, the Search Results List dialog contains the same document multiple times. The workaround is as follows:

1. Raise the Search dialog; then click the Indexes button.
2. Remove *all* indexes by selecting each index in turn and clicking the Remove button. When you remove the final index, you are warned that you won’t have any indexes; click OK. Close the dialog.
3. Exit and restart Acrobat; then re-open the **welcome.pdf** file.

## 2.3 MATRIX$_X$ Installation Guides

The following documents provide instructions for installing the MATRIX$_X$ Product Family software:

***System Administrator's Guide, Windows Version*** — Describes proper setup of a PC running Windows 98 or Windows NT® for the installation of MATRIX$_X$ software products.

***FLEXlm End User Manual*** — Describes FLEXlm from the end-user perspective. It explains how to use the command-line tools that are part of the standard FLEXlm distribution.

***RealSim AC-1000 Controller Installation Guide*** — Describes how to install RealSim software for an AC-1000 controller on both the host and target computers; describes the new features incorporated into RealSim 6.X, provides instructions for updating existing projects to RealSim 6.X, and provides some configuration notes.

***RealSim PC Controller Installation Guide*** — Describes how to install RealSim software for PC controllers, such as AC-104 and PCI_Pro, on both the host and target computers; provides instructions for updating existing projects to RealSim 6.X and some configuration notes.

## 2.4 MATRIX$_X$ Getting Started Guide and Master Index

The following MATRIX$_X$ documents provide help for getting started with basic tasks and for finding the information you need.

***MATRIX$_X$ Getting Started Guide, Windows Version*** — Describes the MATRIX$_X$ Product family and provides an introduction to basic tasks and tutorials for using MATRIX$_X$ software on the ***Windows*** platform.

***Core Documentation Master Index*** — The MATRIX$_X$ Core Documentation suite includes the *MATRIX$_X$ Getting Started Guide*, and all Xmath, SystemBuild, Autocode, and DocumentIt books. This document indexes all of the Core Documentation suite except *Interactive System Identification Module, Part 2*, and Xμ *Module.*

## 2.5 Xmath Books

Xmath software is documented in the *Xmath User's Guide* (formerly *Xmath Basics*) and in manuals for each optional Xmath module.

***Xmath User's Guide*** — Describes Xmath structure and concepts. It provides a tutorial, covers basic features for general Xmath use, and describes advanced Xmath features such as creating a GUI, creating your own MathScript commands, functions, or objects, and linking external programs.

***Control Design Module*** — Explains the use of the Control Design Module including Linear system representation, building system connections, system analysis, classical feedback analysis, and state-space design. It describes each function in the Control Design Module.

***Interactive Control Design Module*** — Describes how to use the Interactive Control Design Module (ICDM), which is a tool for interactive design of continuous-time, single-input, linear time-invariant controllers. ICDM uses the Xmath programmable graphical user interface (PGUI or GUI).

***Interactive System Identification Module, Part 1*** — Describes the Interactive System Identification Module (ISID), which includes system identification, model reduction, and signal analysis tools for identification of linear, discrete time, and multivariable systems.

***Interactive System Identification Module, Part 2*** — Focuses on a special interactive graphical interface for ISID commands that further simplifies system identification. Various graphical comparison tools allow you to try different identification and validation methods. This interface also supplies plots useful for system identification with the touch of a button.

***Model Reduction Module*** — Describes the model reduction module (MRM), a collection of tools for reducing the order of systems.

***Optimization Module*** — Describes nonlinear, quadratic, and linear optimization functions.

***Robust Control Module*** — Describes the robust control module (RCM), a collection of analysis and synthesis tools that assist in the design of robust control systems.

***X*µ *Module*** — Describes Xmath functions used for modeling, analysis, and synthesis of linear robust control systems.

## 2.6 SystemBuild Books

The SystemBuild manuals consist of the *SystemBuild User's Guide* and a number of other manuals for SystemBuild blocks and modules.

***SystemBuild User's Guide*** — Describes how to use SystemBuild, the graphical modeling and simulation environment, to construct a model for a dynamic system. SystemBuild lets you create custom building blocks, hierarchically organize model subsystems into SuperBlocks, and run system simulations based on the models.

***Aerospace Models*** — Describes libraries of SystemBuild models that were written for the aerospace industry.

***BetterState User's Guide*** — Describes how to use BetterStateChart blocks for modeling complex event, state, and transition information.

***BlockScript User's Guide*** — Describes how to write instructions in the BlockScript language. BlockScript is used in SystemBuild with both BlockScript blocks and BetterStateChart blocks.

***FuzzyLogic Block*** — Describes how to use the SystemBuild Fuzzy Logic Block to obtain fuzzy logic control methodology within SystemBuild for simulation and/or code generation. The Fuzzy Logic Block allows users to implement fuzzy logic decision structures of arbitrary complexity within a standardized block-diagram control-logic structure.

***HyperBuild User's Guide*** — Describes how to decrease the computer simulation time of medium and large SystemBuild models. The bigger and more complex the SystemBuild model, the more significant the increase in simulation speed. HyperBuild achieves this improvement by converting a SystemBuild block diagram into highly optimized C code (called HyperCode) that executes much faster in the simulation engine, which normally interprets the model data. HyperBuild can be used to generate code for continuous SuperBlocks only.

***Interactive Animation User's Guide*** — Describes how to create and link Interactive Animation pictures to your model and how to use these pictures for model control and results display at run time. This module is usually considered part of SystemBuild and is now supplied with the standard RealSim package.

***Neural Network Module*** — Describes how the Neural Network Module (NNM) provides users the capability to define, parameterize, and include neural networks as SuperBlocks in a SystemBuild block diagram. Adding neural network technology to the fully integrated block diagram language of SystemBuild includes the capability to simulate your neural network models and to generate embedded code for them via AutoCode.

***State Transition Diagram Block*** — Describes the State Transition Diagram (STD) block. This separately licensed block can be obtained from the SystemBuild Palette Browser SuperBlocks menu. The STD block is an interface between a finite state machine and a SuperBlock diagram. In SystemBuild, each state in a finite state machine is graphically rendered as a bubble rather than a block; the STD editor is used to create bubble diagrams.

## 2.7 AutoCode and DocumentIt Books

***AutoCode User's Guide*** — Describes how to use AutoCode to generate code from a SystemBuild block diagram.

***AutoCode Reference*** — Supplements the *AutoCode User's Guide* and provides additional reference information.

***DocumentIt User's Guide*** — Describes how to use DocumentIt to generate design documentation from a SystemBuild block diagram.

***pCODE Template*** — Describes the template that is used with AutoCode to generate code for the pSOSystem real-time operating system.

***Template Programming Language*** — Describes how to write templates using the Template Programming Language (TPL) for AutoCode and DocumentIt.

## 2.8 RealSim Books

***RealSim User's Guide*** — Details the building of a model and its execution on a RealSim controller.

***RealSim Command Reference*** — Provides command descriptions for using RealSim hardware and software. This includes commands entered from the host, from the controller console, and from Xmath.

***RealSim AC-1000 Controller System Reference*** — Describes the RealSim PC-based architecture for an AC-1000 controller.

***RealSim PC Controller System Reference*** — Describes the RealSim PC-based architecture for an AC-104 or a PCI Pro controller.

***RealSim PC Controller Device Driver Kit Reference*** — Describes how to write a device driver for the RealSim PC model AC-104 controller.

## 2.9 Using Online Help

MATRIX$_X$ online Help is implemented as an HTML fileset linked to MATRIX$_X$ using the Netscape NetHelp Help engine. For fully functional MATRIX$_X$ online Help, you must use Netscape Navigator (HTML browser). Other browsers can be used to view the online Help fileset, but they do not interface with MATRIX$_X$ to provide context-sensitive Help.

MATRIX$_X$ online Help is compatible with Netscape Navigator Versions 3.01 through the version shipped with your MATRIX$_X$ distribution. The version we ship is the most recent English International version available across all MATRIX$_X$ platforms at the time of an initial MATRIX$_X$ release.

The following topics are covered here:

- *Starting the Online Help*
- *Using the MATRIXx Help Window*
- *Navigating Between Topics*
- *Finding Specific Help Topics*
- *Using Help Examples*
- *Using Context-Sensitive Help*
- *Using MATRIX$_X$ Help with Different Versions of Navigator*

### 2.9.1 Starting the Online Help

You can raise the general MATRIX$_X$ Help window in these ways:

- From the command area of the Xmath Commands window, type **help**. If you know the name of the command, function, or topic, specify it after the Help command; for example, **help sba**.
- From the Xmath Commands window, select Help→Topics.
- If Netscape is not running, Xmath launches Netscape and opens the MATRIX$_X$ Help window. On Windows platforms the window is launched directly. On UNIX platforms, Netscape is launched, and then the Help window is spawned from the Netscape session.
- You can also launch the MATRIX$_X$ online Help window independent of Xmath, assuming the MATRIX$_X$ environment variables are properly set and Netscape is on your path. From the operating system command line, type:

```
mtxhelp
```

Note, on Windows platforms, the online Help has its own shortcut on the Start menu.

### *Multiple Navigators*

If you have a Netscape Navigator already running, Xmath attempts to launch the Help window in that version; if the version is not 3.01 or greater, you receive an error message.

- On UNIX platforms it is wise to **close Netscape and/or any other color intensive X resources before starting MATRIX$_X$**. MATRIX$_X$ applications open successfully, but if necessary colors are allocated by other applications, the color mapping your system attempts can result in a poor working environment. The same is true on Windows platforms, but to a lesser extent.
- On Windows platforms Nethelp opens the MATRIX$_X$ Help using *the last version of Navigator called*, not necessarily the Navigator shipped with MATRIX$_X$. You get an error message if the version is not 3.01 or higher.

The MATRIX$_X$ Help consists of simple HTML files with hypertext links so that behavior is consistent among platforms and Navigator versions. Netscape functionality has changed from version to version, however. See Using MATRIXx Help with Different Versions of the Navigator for hints on using different versions with MATRIX$_X$ Help.

### *Common Startup Questions*

#### *Why Does this Help Topic Look Funny?*

In MATRIX$_X$ Help, only text for examples and syntax (where returns must be preserved) have a predetermined font (Courier). For body text Netscape uses default fonts, or whatever you have selected in the Netscape File menu Preferences dialog. The Netscape default font is Times. In rare cases, your machine may not have this font loaded, or it may not have the font in the size you have selected, resulting in pages with letters mysteriously missing. Try choosing another font size, or another font.

- It is best if you have Netscape and other color-intensive applications closed before you start MATRIX$_X$. This allows your application to get the colors it needs. It also means that Netscape may not be able to grab the standard colors the online Help uses. This means hypertext links may not be blue, and so forth. In general, the bad colors do not impair the functionality. For the best results:

a. Close all applications.

b. Start MATRIX$_X$ (but do not launch the Help yet).

c. Start Netscape, and then start online Help from the Xmath Commands window.

#### *Why Can't I Use Explorer?*

Our files are generic HTML 3.2, so you can use Explorer to view them; however, you must install Netscape if you want Context-Sensitive Help to work. If you use Explorer, some graphical problems may occur, as we only test the Navigator, and the level of HTML support does vary among browsers.

#### *Where Is this File?*

The topic's filename and relative location are shown at the bottom of every file, just above the copyright, e.g., **$XMATH/help/masterIX.doc.html**. You can find the value of the environment variable **$XMATH** from the command area of the Xmath Commands window. For UNIX systems, type **oscmd("env");** for Windows systems, type **oscmd("set")**. Locate **XMATH** among the environment variables displayed.

## *2.9.2 Using the MATRIXx Help Window*

### *Help Window Layout*

The MATRIX$_X$ Help window includes elements common to many browsers:

- *Frames*
- *Buttons*

### Frames

The MATRIX$_X$ Help window uses three frames:

- The left frame contains the topics hierarchy. All blue text entries are links to MATRIXx Help topics. Topics usually contain lists of pertinent functions and commands.
- The lower right frame displays the current topic. For example, click a subject in the topics hierarchy, and it is displayed in the topic frame.
- The upper right frame displays the letters of the alphabet, and the Symbols topic. These are entry points into the online Help index. For example, click D to display an alphabetized list of topics that start with D.

You can use scrollbars or the Bottom and Top buttons to navigate within frames. To change the width or height of a frame, click the dividing line between two frames and drag in the direction you want the frame to enlarge/decrease. Alternatively, change the size of the entire window using a method appropriate to your window manager.

### Buttons

The Help window frame has Backwards, Forward, and Exit buttons. The backwards and forwards arrows move you to the last link visited, or forward in the viewing history to a link you've previously visited. You can also go forward and back from the Netscape Quick Access menu. Hold the right mouse button down (anywhere within the Help window) to raise this menu.

## 2.9.3 Navigating Between Topics

By default, blue text in MATRIX$_X$ Help can be used to jump to a related topic. To jump, click the text.

The Prev and Next buttons, when shown, take you to the previous/next file in the fileset. Because topics are cross-linked, this is not necessarily the next file in the listing that the topic hierarchy shows, because topics are cross-linked.

### Topic Groupings

Some topic categories have been grouped into single large topics:

- AutoCode
- Model Reduction Module
- Optimization Module
- Programmable GUI
- Robust Control Module
- RealSim
- RVE
- Simulation
- SystemBuild Utilities

You can access these topics using index and cross reference jumps in the normal way, however:

- Top and Bottom links lead to the first and last files in the category, not the top or bottom of the individual Help.
- Scroll up or down to navigate through the fileset topics; Prev and Next are not available.
- When you print a combined fileset, the entire topic category is printed.

## 2.9.4 Finding Specific Help Topics

- If you know the name of the topic you want to view, go to the command area of the Xmath Commands window and type **help**, followed by the name of a command, function, or block. Abbreviation is supported as long as enough characters are supplied to guarantee a unique response. For example,

```
help plot     # raise the help on plot
help algeb    # raise the help on the AlgebraicExpression block
```

- Go to the topics hierarchy and select a general topic; follow links through the topic hierarchy listings until you find a topic of interest.
- Use the online Help Index. Click a letter of the alphabet in the upper right Frame to display all entries that start with that letter. The Prev and Next buttons to jump to the next alphabetized category.

### *2.9.5 Using Help Examples*

There is a special convention for MATRIX$_X$ commands and functions. If a command or function name at the top of the Help category is blue, there is a link from the topic name directly to the Examples portion of the Help. In some cases, examples are distributed through the topic. This is usually done to provide related discussion, or keyword category grouping.

Command or function Help usually include Help examples in the form of Xmath or SBA commands and functions. To test the Help examples, use your window manager's copy and paste conventions to copy the example text into the command area of the Xmath Commands window, and then press Return. Usually this involves highlighting some text, using mouse-clicks or menu options to copy the text, and then pasting the text to the command area of the Xmath Commands window.

**NOTE:** Loss of highlighting is a frequent side-effect of color map conflicts. Netscape's default highlight color in most environments is a pale yellow. If this color is not available it may seem that highlighting is "broken" when you attempt to highlight text on a white page. In most cases you can assume the highlighting is taking place and carry out your copy and paste operation successfully.

Some Help examples consist of Xmath command and function definitions used in conjunction with calls issued from the command area. To test examples where Xmath commands and functions are defined:

- Use a text editor to create a new Xmath command or function file.
- Copy and save the example command or function definition script to the file.
- Name the file **commandName.msc** or **functionName.msf** and save it to a folder included in the lookup path.
- Execute the commands that call the newly defined command or function, by copying them into the Xmath command area.

### *2.9.6 Using Context-Sensitive Help*

The MATRIX$_X$ online Help facility is context sensitive. Clicking the Help button or the ? toolbar button from a specific window or dialog launches Netscape Navigator to provide you with information specific to that topic.

For example, to get Help on the SuperBlock Editor, go to the Editor and select Help→Topics, or click the ? toolbar button. The Help for the SuperBlock Editor appears. To get Help on a given block, open its block dialog (select the block, and then press Return), and click the Help button. The Help for the active block is displayed.

## 2.9.7 Using MATRIX$_X$ Help with Different Versions of Navigator

If you are not familiar with HTML browsers you should read the Netscape Navigator online documentation. The MATRIX$_X$ online Help deals exclusively with the Navigator. If you have Communicator, which includes the Navigator as part of a tool suite, only the Navigator is relevant to MATRIX$_X$.

For more information on Netscape products, see Netscape's home page at **http://home.netscape.com**.

### 4.X Navigator Commands

A standalone Navigator (sans Communicator) is shipped with MATRIXx 6.1 or higher.

- In Navigator 4.X, the NetHelp application disables all normal browser menus and toolbar buttons in a Netscape window. If you want to print or manipulate a MATRIX$_X$ Help topic, you must first send the file to a new window. To do this, place your cursor in the frame containing the desired text, and then use the right mouse button to raise the Navigator Shortcut menu; select Open Frame in New Window. The new window has all the standard Navigator options enabled. You can now print the Help file, or even save the HTML source and edit it locally (this is useful when you are writing online Help for a MathScript program and you'd like it to look the same as MATRIX$_X$ Help).
- To alter the appearance of a Netscape window (for example the font sizes or styles), use Edit→Preferences. Note that changes you make in one window affect all others.
- For more information on Netscape products open a new window and select Help→Contents. The Netscape NetHelp window appears.

### *Navigator 3.X Commands*

Although Wind River does not ship Navigator 3.X, the MATRIX$_X$ Help interface supports it. To get more information on using Netscape 3.X, read the Netscape Navigator HandBook. This document is available from the Help menu on the Netscape window frame. By default, you can't see this menu from the MATRIX$_X$ Help window because NetHelp turns off the toolbars for the standard browser.

- On Windows platforms, the menu bar cannot be raised. However, the most useful commands can be triggered with the Ctrl key:

| Keystrokes | Action |
|---|---|
| Ctrl+f | Open the Find dialog. |
| Ctrl+p | Open the Print dialog. |
| Ctrl+n | Open a New browser. By default nothing is displayed in the new window. To view a specific Help file, find its location at the bottom of MATRIX$_X$ Help topic file (just above the copyright). In the new browser, select File→Open and supply the Help topic path, where $XMATH is expanded to the location of your Xmath installation.<br><br>You can now print the Help file, or even save the HTML source and edit it locally (this is useful when you are writing online Help for a MathScript program and you'd like it to look the same as MATRIX$_X$ Help). |

## 2.10 MATRIX$_X$ Release Information

For current MATRIX$_X$ release information, see the MATRIX$_X$ 7.0 *Release Notes:*

- online books: click Release Notes on the document CD Welcome page.
- online Help: select the topic *Release Info→Release Notes*

## 2.11 MATRIX$_X$ Customer Support

For up-to-date information on how to obtain customer support for MATRIX$_X$, visit the Wind River web site at the following URL:

**http://www.windriver.com/support**

To contact customer support for MATRIX$_X$:

- Send E-mail to **support@windriver.com.**
- Phone **1-800-USA-4WRS (800-872-4977).**

# 3 Xmath

Xmath provides tools for mathematical analysis. You can create, store, plot, and explore data in Xmath. You can define your own functions, commands, and objects, and also link in externally compiled C or Fortran code. Xmath is the controlling environment for SystemBuild and related products. This chapter gives an overview of Xmath functionality.

## 3.1 Introduction to Xmath

The following sections introduce the Xmath tools and capabilities:

- Data Handling
- Numerical Analysis
- MathScript

### 3.1.1 Data Handling

MathScript, the language of Xmath, allows you to define and manipulate data in the form of numbers, objects, graphs, and text. Xmath provides a graphical user interface to facilitate data management. You can save, load, import, and export data.

### 3.1.2 Numerical Analysis

Xmath provides an extensive library of commands and functions, including mathematical functions and filter design functions. Xmath also provides two plotting facilities, one with an interactive graphics display, and the other integrated with a programmable GUI facility.

Optional Xmath modules contain commands and functions to address special uses. The modules are documented in online Help and each has an online manual. Discussions of theory and examples are provided in the manuals. See *2. MATRIX$_X$ Publications, Online Help, and Customer Support* for a summary of available documentation.

### 3.1.3 MathScript

Xmath's programming language, MathScript, allows users to alter or extend Xmath's functionality. An interactive debugger and a full complement of checking utilities simplify developing scripts to define functions, commands, and objects.

Xmath has an object-oriented structure that makes it unique among numerical analysis tools. This enables efficient numerical handling, including the overloading of operators, and more. Xmath's hierarchical objects greatly reduce the amount of user programming devoted to checking data characteristics.

Xmath includes a fully programmable graphical user interface (PGUI or GUI). This programmable GUI allows you to create and manipulate windows, dialogs, and other user interface tools. Any user can develop convenient user interfaces. You can find instructions for using and building GUI applications in the Xmath online Help topic: MathScript Programming, Programmable GUI.

MathScript supports calling external routines from within Xmath, or you can call Xmath from your own C programs. The Linked External (LNX) facility uses an interprocess communication (IPC) mechanism for communication between your external routine, which runs as a separate process, and Xmath. You can modify and recompile your routine without exiting Xmath, so that you can use and debug external programs in the same session. The User-Callable Interface (UCI) allows a C program to invoke Xmath as a computational engine. You can invoke Xmath from your C program and pass it values or expressions to evaluate and retrieve results, perform calculations, or plot values. For information on how to create LNXs and UCIs, see the *Xmath User's Guide*.

## 3.2 Getting Started in Xmath

This section assumes that Xmath has been properly installed and configured. See the *MATRIX$_X$ System Administrator's Guide, Windows Version* for installation details.

**NOTE:** Many of the operations described in this guide can be accomplished by alternative methods and shortcuts. To simplify the presentation, only one method is specified in most cases.

### 3.2.1 Directories Defined by Environment Variables

The MATRIX$_X$ product line is installed in a directory known as ISIHOME. The installation process modifies Xmath startup scripts and provides the location of ISIHOME as an environment variable (**%ISIHOME%**) that is known *only* within the MATRIX$_X$ environment. Three additional environment variables, also known only within the MATRIX$_X$ environment, define three subdirectories of ISIHOME: **%XMATH%**, **%CASE%**, and **%SYSBLD%**.

The MATRIX$_X$ environment variables are recognized only in the Xmath command area. If you need to use them elsewhere (for example, in the operating system), you must specify the full pathname. In such cases, we indicate the file location with italics: *ISIHOME*, *XMATH*, *SYSBLD*, and *CASE*. If you do not know this pathname, you can determine it by typing the following command within the Xmath command area:

```
oscmd("echo %variable%");
```

where **%variable%** is **%ISIHOME%**, **%XMATH%**, **%CASE%**, or **%SYSBLD%**.

**NOTE:** The environment variables discussed within this section are subject to change, and therefore, should not be used in scripts.

### 3.2.2 Setting Your Display Colors

Xmath plots require that the Windows display driver be set to display a minimum of 256 colors.

Default colors for your display windows, borders, and other screen components are established through the Appearance tab in the Settings→Control Panel→Display selection, just as in other Windows applications.

### 3.2.3 Starting Xmath

To start and run Xmath, select Start→Programs→MATRIX$_X$→Xmath.

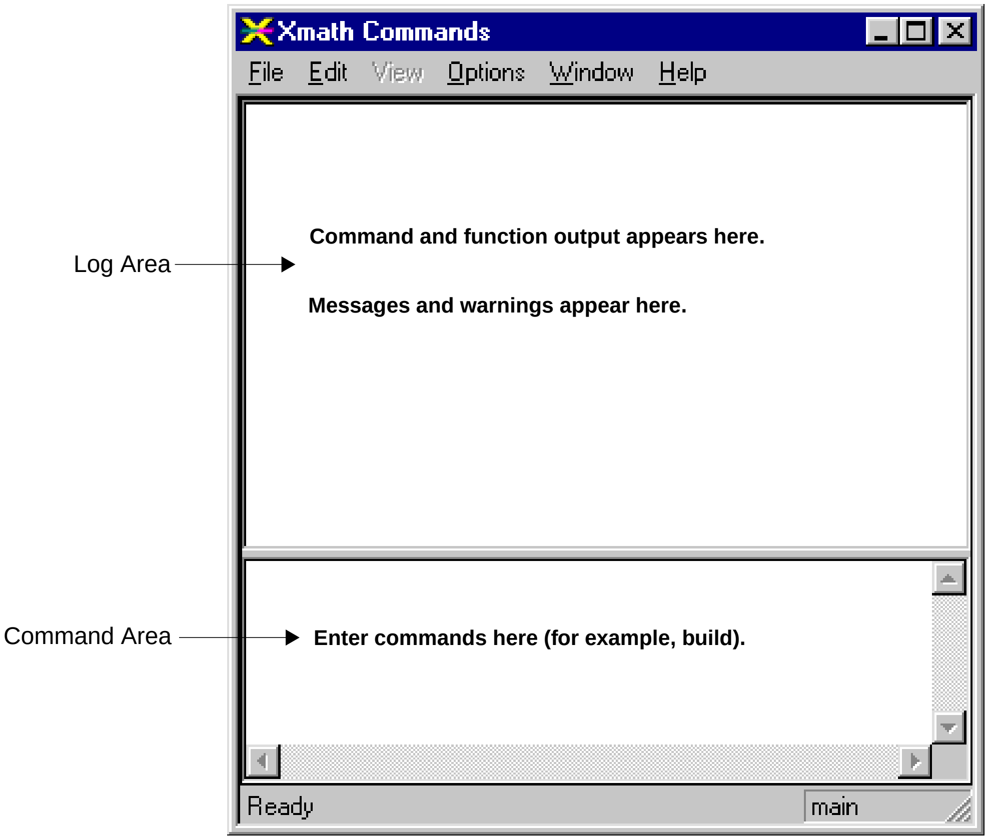
Xmath starts and the Xmath Commands window is displayed as shown in Figure 3-1.

### 3.2.4 The Xmath Commands Window

When you invoke Xmath, the Xmath Commands window is displayed (see Figure 3-1).

You type input in the command area. Output, environment status, and error messages are displayed in the log area above.

Figure 3-1 **Xmath Commands Window**



**Command and function output appears here.**

**Messages and warnings appear here.**

**Enter commands here (for example, build).**

### Menu Choices

Menu choices available in the Xmath Commands window are:

- The Edit menu displays commands for editing the Xmath command area. Conventional windows selections are Undo, Cut, Copy, and Paste. Xmath adds Clear Log, Clear Command, and Send Command.
- The View menu is reserved for expansion.
- The Options menu includes a Font menu item that allows you to set any TrueType font installed on your Windows system to be used in your Xmath displays. It also has a Format menu item for selecting the format of numeric values displayed in the Xmath log window.

**NOTE:** Xmath provides no capability for saving a font selection between sessions.

- The Window menu lets you bring up the Graphics or Palette window or invoke SystemBuild.

### Command Window Execution

The Commands window has two command modes: *single-line* and *multiline*.

The default mode is single-line. After typing a MathScript instruction, you press the Return key, and the instruction is executed by Xmath.

The key sequence Shift-Return toggles the command mode. In multiline mode, the Return key adds a new line rather than sending the instruction to Xmath.

For example:

| | |
|---|---|
| `for i=1:10` | Press Shift-Return at the end of the first line to switch to multiline command mode. Press Return to add a new line. |
| `i?` | PressReturn at the end of the second line to add a new line. |
| `endfor` | Press Shift-Return at the end of the last line to switch to single-line command mode. Press Return to send the multiline for loop to Xmath. |

**NOTE:** The above text is not valid for cutting and pasting from online format into Xmath.

### 3.2.5 Running Demos

For a tutorial of Xmath's basic features, see "Xmath Jumpstart" in *Xmath User's Guide*. For an online demo, click in the command area, and then type:

```
demo
```

You can choose from several example scripts. As a script executes, explanatory text is displayed in the logarea; a Pause dialog pauses the script to give you time to read the text or view a plot. Move the Xmath Pause dialog so that it does not obscure the Commands window.

### 3.2.6 Accessing Online Help

To access online Help from Xmath, select Help →Topic from the Xmath Commands window. See *2.9 Using Online Help*, p.21, for detailed instructions on using the online Help feature.

### 3.2.7 Stopping Xmath

- To exit from Xmath, select File→Exit from the Commands window menu bar. Xmath prompts you to save the workspace.
- To stop an Xmath operation, type Ctrl-Break. Note that Ctrl-Break cannot interrupt a process in communication with the operating system (**load**, **save**); this includes creating and displaying windows.
- To abort Xmath if the program stops responding (for example, after a system error), enter Ctrl-Alt-Delete to display the Task Manager; then select Xmath Commands and click the End Task button.

## 3.3 Performing Sample Xmath Tasks

This section introduces some basic and advanced Xmath features, including MathScript. If you have never used a mathematical analysis package, become familiar with the demos described in *3.2.5 Running Demos*, p.37, before continuing.

In the sections below, instructions that you enter in the commands area are shown in **bold Courier**. A description of the instruction, if applicable, and related Help topics are found following the Xmath comment symbol (#), to the right of each input.

Try some of the examples below. You do not need to type comments. If you are accessing this document online, the instructions can be copied and pasted to the command area for execution.

**NOTE:** Command and function names can be shortened to a unique opening substring (as few as four characters).

If Xmath is not running, see *3.2.3 Starting Xmath*, p.34.

### 3.3.1 Creating Data

The first step in mathematical analysis is usually creating some data. Enter the following MathScript statements to create and save the variables (comments are for your information):

```
                 # See punctuation.
a=[1,2,2^2,3^3]  # Define a variable. See vector and operators.
b=1:.1:5         # See regular vector.
c=sin(b)         # Call a function. See functions.
```

Xmath provides the ability to save a graph as data (to a variable):

```
graph1=plot(c,{title="Creating the Graph Object graph1."})
```

For more information on graph objects see the online Help topic “Graph Object” under Xmath→Plotting.

### 3.3.2 Getting to Know Objects

You have just created two types of numeric objects. Let’s identify each object.

```
whatis b          # See commands for command calling syntax.
whatis c          # See objects.
```

Look at the vector topic in the online Help. **vector** is a numeric class. Nonnumeric, or complex, objects are strings or combinations of strings and numeric objects. Polynomials fall into this category:

```
d=makepoly(a,"d")       # See makepoly.
e=polynomial(1:3,"d")   # See polynomial.
```

Xmath’s object structure allows you to build mathematical constructs in a natural way. Create a system as follows:

```
sys=system(d,e)         # See system and transfer function.
```

Some functions accept only a certain type object and return another type object. For example, **char( )** accepts an integer and returns a string:

```
str=char(65)
```

The **freq( )** function accepts a system and returns a parameter-dependent matrix (PDM). A PDM is a special object that stores matrices in relation to an independent parameter or domain. (In SystemBuild, simulation output is a PDM.) The independent parameter is typically time or frequency.

Let’s see how PDMs look.

```
f=freq(sys,b)?
    g=freq(sys,{fmin=1,fmax=length(f),npts=length(f)})?
```

To create **f**, we specified a vector of frequencies; this became the domain. To create **g**, we let **freq( )** calculate the frequencies for the domain. Let’s compare the two:

```
graph1=plot(f,{rows=2})?
graph2=plot(g,{row=2})?
```

For more on PDMs, see pdm and PDM object in online Help. For more on the **plot( )** function, see the *Xmath User's Guide* and the *plot* topic in the MATRIX$_X$ online Help.

### 3.3.3 Saving, Loading, and Printing Data

To list the variables you have created so far, type

```
who
```

Note the sizes (see who( ) in online Help for an explanation).

To save everything you have created, type

```
save
```

Xmath saves all data to a file with the default name **save.xmd** in the current working directory. (You may want to specify a filename because **save.xmd** will be overwritten by the next **save** command.) The first of the following two commands saves your variables to a file, and the second uses a wildcard to save a subset of variables to a different file.

```
save "try.xmd"
save "try_2.xmd" g* sys
```

See “save” and “wildcards” in the Xmath online Help.

Type the following command to display your working directory:

```
show directory
```

You can use the Xmath operating system command **oscmd** to list the files you saved:

```
oscmd("dir try*.*")
```

The operating system should find both **try.xmd** and **try_2.xmd**. If it does, you can delete what you have created in Xmath:

```
delete *
```

Retrieve the second file you saved and use the function **who( )** to list the variables that you have:

```
load "try_2"
who
```

### *Graphics*

Let's use the variable **sys** again:

```
nyquist(sys)?
```

The function **nyquist( )** creates a plot; however, the output of **nyquist( )** is not the graphics object. To save the contents of the Graphics window, use one of the following methods:

- In the Xmath Graphics window, select File→Bind to Variable, and then specify the variable name **graph3**
- From the Xmath Commands window command line, type:

  ```
  graph3=plot()
  ```

You now have three graph objects: **graph1**, **graph2**, and **graph3**. You can display them in a manner analogous to other variables:

```
graph1
graph2
graph3
```

### *Printing Graphs*

To print the graph currently displayed in the Graphics window, use one of the following methods:

- In the Xmath Graphics window, select File→Print and fill in the resulting dialog.
- In the Xmath Commands window, enter

  ```
  hardcopy {color=0}
  ```

  The setting **color=0** ensures that you receive a black and white rather than a color plot, which is the default.

**NOTE:** To use the hardcopy command to print directly, the environment variable **%XMATH_PRINT%** must be defined. Open Control Panel→System and examine the environment variables. If you need further help, see the *Xmath User's Guide*.

Use **hardcopy** to save your graphics to a PostScript™ **(.ps)** file and then submit the file to the printer with a standard command. For example:

```
hardcopy graph3, file="graph3.ps", {color=0}
```

From a Command Prompt window, type:

```
copy file.ps path_to_printer
```

## 3.4 MathScript

MathScript, the language of Xmath, defines statements, constructs, punctuation, functions (MSFs), commands (MSCs), and objects (MSOs). You can use MathScript to create your own functions and commands. Open a text editor, and create a file named **cdown.msf** (**.msf** corresponds to MathScript function) with contents as shown in Example 3-1.

Example 3-1 **cdown.msf**

```
#{
     cdown counts from the integer input down to 1
     and displays the square root of each count.
     cdown outputs a vector of the square roots
}#
function [out]=cdown(c)

if is(c,{integermin=1}) then
    out=[];
    display "************"
    for i=[c:-1:1]
      display "SQRT(" + string(i) + ") = " + string(sqrt(i))
      out=[out,sqrt(i)];
    endfor
else
    error("cdown accepts positive integers only","C",c)
endif

endfunction
```

Save your file in the current directory for Xmath (or any directory in the lookup path), and return to Xmath. To see your current lookup path:

```
show path
```

To add a new directory to the path:

```
set path "directory path specification"
```

Try calling **cdown( )** with valid and invalid inputs:

```
cdown(5)

cdown(-5)

cdown("what, me worry?")
```

## 3.5 The Xmath Debugger

The Xmath debugger helps you to debug MathScripts you write (MSFs, MSCs, and MSOs). You can control the Xmath debugger interactively from the command area in the Commands window.

### 3.5.1 Starting the Debugger

Debug mode starts under three circumstances:

- A call to **debug** is made with a script that is set up for debugging—that is, you execute the debug command:

  **debug** *script_name*

  The debugger opens automatically on the first executable line in the script.
- A script contains a syntax error (for example, an error in punctuation, such as a missing brace: **plot(a,{xlab="A missing brace")**.
- A script contains a run-time error. A run-time error occurs when an instruction is impossible to process. The following statement would cause a run-time error because the operation + does not accept an integer and a string:

  ```
  x=5 + "hello"
  ```

Normally, when an error is detected in a script, Xmath automatically displays the error in the debugger window and sets the interpreter to debugging mode. To prevent the interpreter from going into debugging mode, execute the command:

**set debugonerror off**

### 3.5.2 Using the Debugger

In the command window, let's start the debugger by typing:

```
debug cdown
```

The debugger sets a break at the first line of executable code—in this case, line 6. Now that a break point is set, let's try the debugger:

```
cdown(2)
```

Look at the difference in the status bar at the bottom of the Xmath window. You are now in debug mode, and the function that you are debugging shows on the right side. You can step through the code and examine local or global variables. Type **next** to continue until you reach the first line of the **for** loop. To watch the variable **i**, type:

```
set watch i
```

Continue to type **next**. Note that you travel through the for loop two times, and the debugger notifies you when **i** is incremented.

You can examine variables local to the function. In the command area, type:

```
who
i?
```

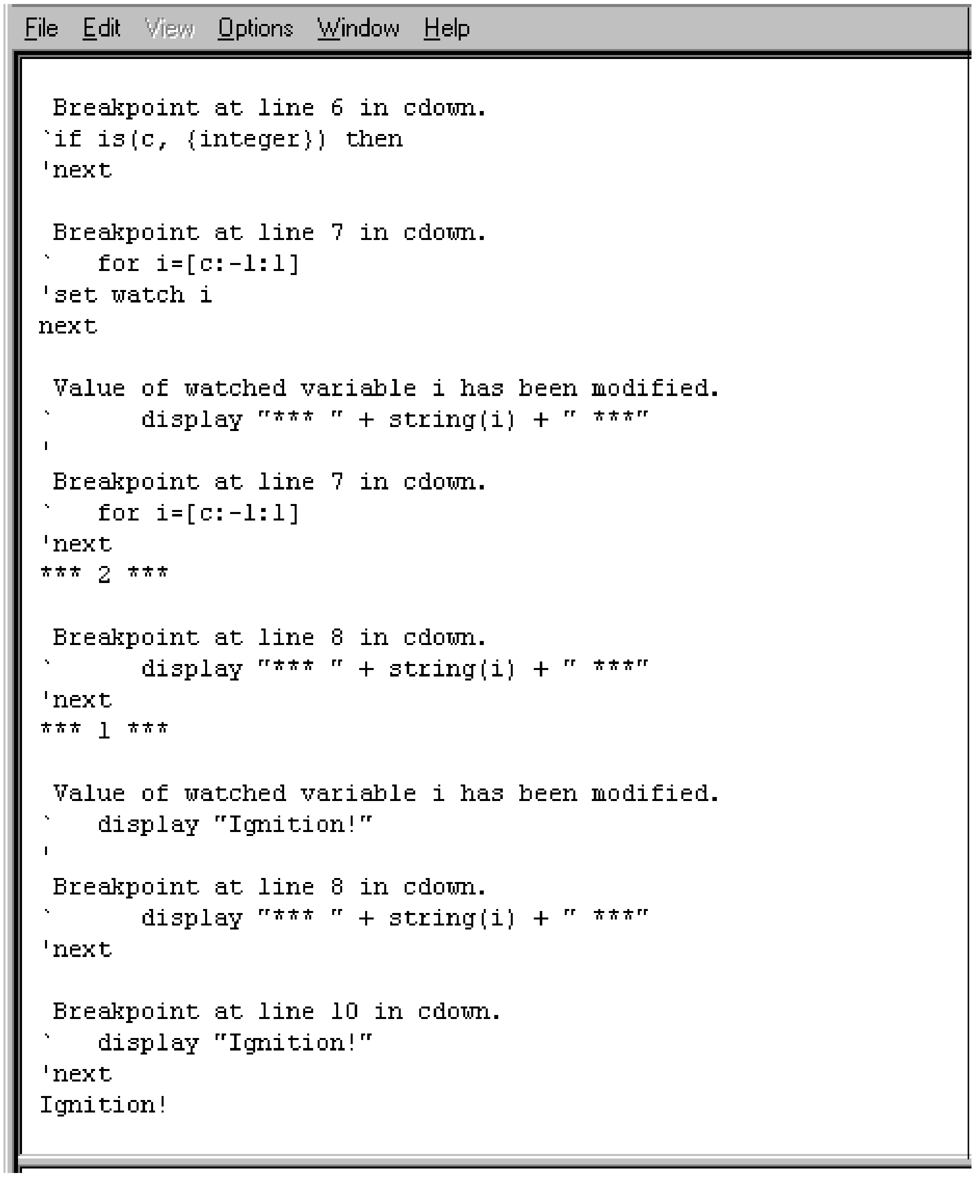
When you fall out of the loop, type **next**, or **go** to run the function through to the end. Figure 3-2 shows a debugging session similar to what you have done.

For additional information, see the "MathScript Programming→MathScript Debugger" online Help topic.

### 3.5.3 Exiting the Debugger

When you reach the end of the MathScript, you automatically exit debug mode. Type **abort** in the Commands window to exit debug mode before completing the script.

Figure 3-2 **Xmath Debugger Session**



```
 Breakpoint at line 6 in cdown.
`if is(c, {integer}) then
'next

 Breakpoint at line 7 in cdown.
`   for i=[c:-1:1]
'set watch i
next

 Value of watched variable i has been modified.
`      display "*** " + string(i) + " ***"
'
 Breakpoint at line 7 in cdown.
`   for i=[c:-1:1]
'next
*** 2 ***

 Breakpoint at line 8 in cdown.
`      display "*** " + string(i) + " ***"
'next
*** 1 ***

 Value of watched variable i has been modified.
`   display "Ignition!"
'
 Breakpoint at line 8 in cdown.
`      display "*** " + string(i) + " ***"
'next

 Breakpoint at line 10 in cdown.
`   display "Ignition!"
'next
Ignition!
```

### 3.5.4 Correcting Errors During Debugging

When you are in the process of developing a MathScript, you can open your file in an ASCII text editor and fix problems that the debugger locates. After you save your file, you can restart the script, and start debugging again. The debugger identifies the locations of errors by means of program line numbers; however, one limitation of some editors is that they do not support line numbers. You can use the editor's find feature to locate the error by copying the line containing the error from the debugger to the search field. To avoid this inconvenience, you can use an ASCII editing program that supports line numbering.

## 3.6 Xmath Plotting

Xmath provides a choice of three basic plotting functions:

- The **plot( )** function provides an easy to learn syntax for 2d and 3d plotting in an interactive graphics window. For a quick, interactive look at your data, and for 3d plotting, the **plot( )** function is a good choice.
- The **uiPlot( )** function provides full featured 2d plotting integrated with an extensive programmable GUI facility. If you want more control over the formatting of your 2d graphics, or the ability to integrate plots with your own interactive Xmath PGUI tools, then **uiPlot( )** has the power you need.
- The **plot2d( )** function provides quick access to advanced formatting features of the uiPlot function, while avoiding the cost of constructing a programmable GUI tool. Use **plot2d( )** to obtain highly-customized 2d graphics without writing a PGUI tool.

## 3.7 Exploring Additional Topics

There are many more topics to explore in Xmath. For additional information, see the MATRIX$_X$ online Help and the *Xmath User's Guide*.

# 4
# SystemBuild

SystemBuild is a graphical programming environment that uses a block diagram paradigm with hierarchical structuring for modeling and simulation of linear and nonlinear dynamic systems. You can use the SuperBlock editor to build block diagram models, and then test them with SystemBuild Simulator and additional analysis tools. This chapter presents an overview of SystemBuild, as well as a tutorial to guide you through some of the most common SystemBuild tasks. The tutorial includes the use of BetterState with SystemBuild.

Additional information about SystemBuild is available:

- The *SystemBuild User's Guide* details use of the SystemBuild SuperBlock Editor and the SystemBuild Simulator. It also contains a comprehensive guide to terms, concepts, and keyboard and mouse actions, as well as several chapters on special topics.
- The extensive SystemBuild block library and other technical reference topics are documented in the MATRIX$_X$ online Help.
- *2.6 SystemBuild Books*, p.18 lists additional SystemBuild publications.

## 4.1 Introduction to SystemBuild

This section introduces fundamental SystemBuild concepts, tools and functions.

Table 4-1 provides definitions for key terms used throughout this guide and in other SystemBuild documentation.

Table 4-1 **Definition of Key SystemBuild Terms**

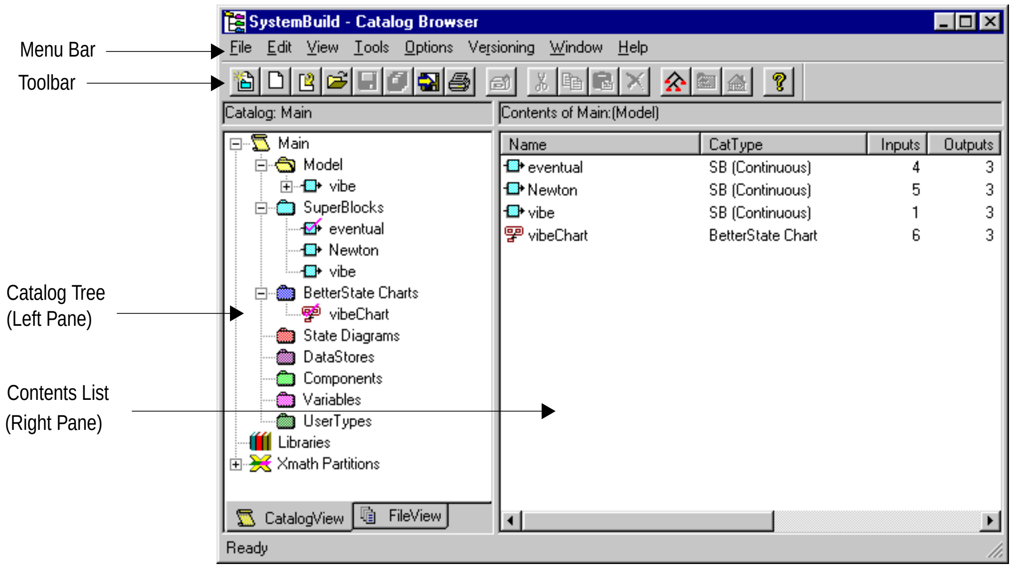
| **Term** | **Definition** |
|---|---|
| SuperBlock | A basic hierarchical object in SystemBuild, which serves as a container for blocks and defines the environment in which they operate. |
| Block | A basic functional element of SystemBuild. A set of blocks are used to make a block diagram model of a controller or a real-time system. |
| Internal Connection | Signals and data are passed between blocks using connections that appear as lines in the diagram within the Editor window. Internal connections pass data between blocks within the same SuperBlock. |
| External Connection | Connections between the SuperBlocks of a model and between the SuperBlocks and the outside world. |
| BetterState Chart | A construct that maintains state information, controls state transitions based on events and conditions, and produces actions associated with states and transitions. |

### *4.1.1 Catalog Browser*

The Catalog Browser is used to manage your SystemBuild models. You use the Catalog Browser to save and load model catalogs composed of SuperBlocks and BetterState Charts. It can also be used to view currently loaded SuperBlocks and BetterState Charts, create new SuperBlocks and BetterState Charts, and to select SuperBlocks and BetterState Charts for editing, as well as other functions.

The Catalog Browser, shown in Figure 4-1, contains a menu bar and a tool bar with buttons that are shortcuts to menu operations. The main portion of the Catalog Browser is divided into two *panes*. The left pane displays a hierarchical catalog tree of different types of objects (for example, SuperBlocks). The hierarchy provides compartmentalization of models and allows you to build and visualize extremely large models; it also provides for reuse of elements of a diagram. The right pane contains the contents of the catalog object selected in the left pane.

Figure 4-1 **Catalog Browser**



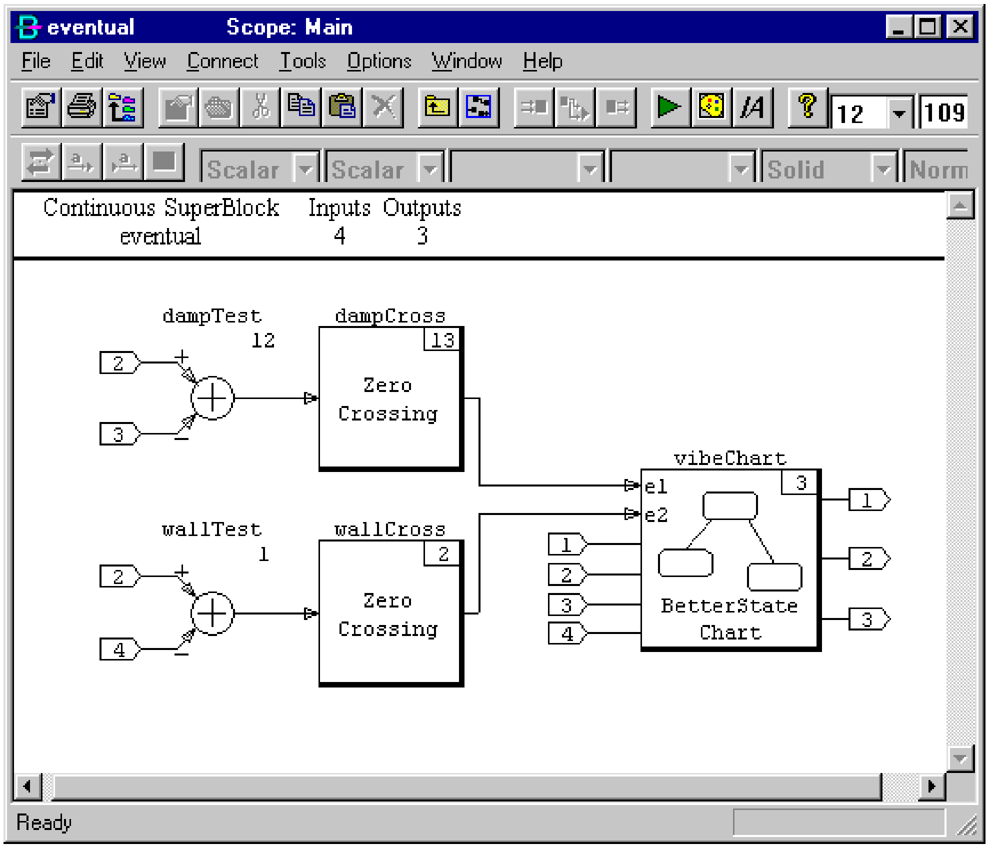
| Name | CatType | Inputs | Outputs |
|---|---|---|---|
| eventual | SB (Continuous) | 4 | 3 |
| Newton | SB (Continuous) | 5 | 3 |
| vibe | SB (Continuous) | 1 | 3 |
| vibeChart | BetterState Chart | 6 | 3 |

## 4.1.2 SuperBlock Editor

The SuperBlock Editor (also known as the Editor window or Editor) offers a user-friendly graphical modeling environment, which allows you to construct continuous-time, discrete-time, and hybrid systems of arbitrary complexity. You use the Editor window (see Figure 4-2) to edit the contents of your model.

SystemBuild supports the use of up to 20 Editor windows at once. Each Editor window can display the contents of one SuperBlock. The Window menu available on both the Catalog Browser and the SuperBlock Editor facilitates switching between the Catalog Browser to select a SuperBlock for editing, and an Editor window to do the editing.

Figure 4-2 **SystemBuild Editor**

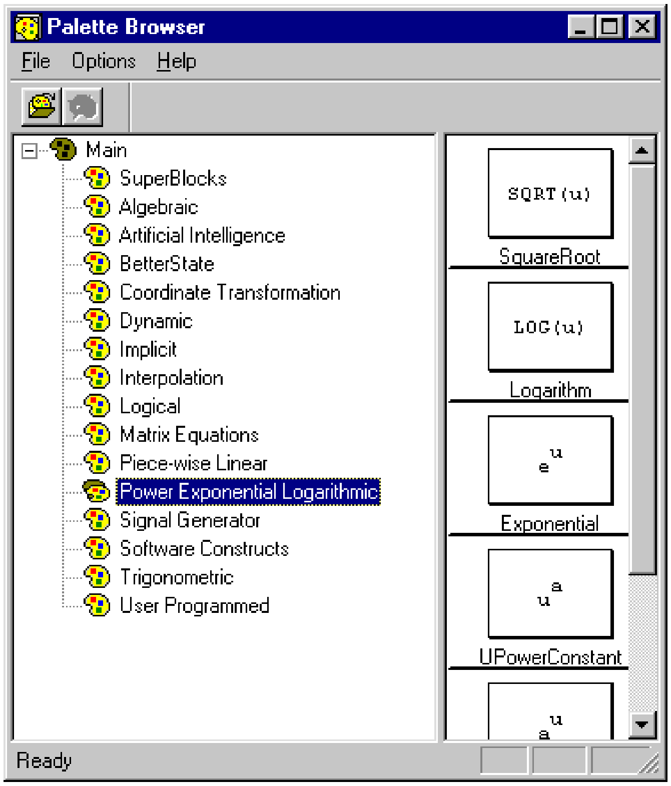


Continuous SuperBlock eventual Inputs 4 Outputs 3

### 4.1.3 SystemBuild Palette Browser

The SystemBuild Palette Browser (see Figure 4-3) provides a choice of over 80 block types, including dynamic systems, algebraic and logical functions, signal generators, piecewise linear functions, trigonometric and exponential functions, and user-programmable blocks. The palette feature is customizable. The *SystemBuild User's Guide* describes several methods for adding custom blocks and custom palettes.

Figure 4-3 **Palette Browser**



### 4.1.4 SystemBuild Simulator

The SystemBuild Simulator facilitates simulating your block diagram model under user-defined conditions. The Simulator provides flexibility in algorithms for integration, data input methods, model timing, and other areas. Both interactive and command based simulation interfaces are provided.

Simulation in an interactive mode lets you interact with the model, and monitor outputs of your blocks during simulation. You can debug your models with interactive capabilities such as block stepping or time stepping.

You can change the values of some block parameters during simulation by using the Run-Time Variable Editor (RVE).

### *4.1.5 Two- and Three-Button Pointing Devices*

Workstation users of SystemBuild, accustomed to the three-button pointing device, or mouse, may find themselves on a Windows NT/95 machine with only a two-button mouse. The Microsoft Windows convention for the two-button mouse is that any operation that you perform using the middle mouse button on the workstation can be performed on a PC by using the right mouse button and the Ctrl key. To connect blocks, one of the most common functions of SystemBuild, for example, click the right mouse button in each block with the Ctrl key pressed.

### *4.1.6 Specifying an ASCII Text Editor*

You may need a text editor for entering text in some block properties dialogs. Each tab that requires text has a drop-down combo box that allows you to select a text editor.

To customize the editor selections available:

Edit the ***SYSBLD*\etc\user.ini** file.

See the *SystemBuild User's Guide* for details.

To change the default editor in Windows operating systems, from Windows Explorer, select Control Panel→System. In the System dialog, select the Environment tab. At the bottom, type **EDIT_COMMENT** in the Variable text field; type the path and filename of your editor in the Value text field. Alternatively, you can type the following command in a DOS Command window:

**set EDIT_COMMAND=*Editor_name***

You must restart Xmath before the new editor becomes available.

If no text editor is specified by the **%EDIT_COMMENT%** environment variable, the default is Notepad, which is a simple, menu-driven ASCII text editor available on every Microsoft Windows system.

### 4.1.7 SystemBuild Optional Modules

This section describes optional modules available for SystemBuild.

#### Fuzzy Logic Block

The Fuzzy Logic Block module lets you design and implement fuzzy logic real-time applications that are fully supported by SystemBuild, AutoCode, RealSim, and DocumentIt.

#### Neural Network Module

The Neural Network Module lets you define, parameterize, and include neural networks as SuperBlocks in a SystemBuild block diagram. Adding neural network technology to the fully integrated block diagram language of SystemBuild includes the capability to simulate your neural network models and to generate embedded code for them via AutoCode. The module supports both training (offline) and learning (real-time) modes of operation.

#### State Transition Diagram Block

State transition diagrams (STD) offer the capability to design and implement finite state machines. A mathematically rigorous implementation of finite state machines is supported by simulation, AutoCode code generation, and DocumentIt. RealSim also supports STDs.

## 4.2 Starting and Exiting SystemBuild

In this section you learn how to start and exit SystemBuild.

**NOTE:** Many of the operations described in this guide can be accomplished by alternative methods and shortcuts. To simplify the presentation, only one method is specified in most cases.

### Starting SystemBuild

To start SystemBuild:

1. If Xmath is not currently running, start Xmath as described in *3.2.3 Starting Xmath*, p.34.
2. Type the following in the command area of the Xmath Commands window:

   ```
   build
   ```

   After a short time, SystemBuild is loaded and the Catalog Browser window is displayed.

### Exiting SystemBuild

To exit SystemBuild:

Select File→Exit from the Catalog Browser.

SystemBuild asks if you want to save your work before exiting; if you answer yes, the Save As dialog appears.

## 4.3 Basic SystemBuild Tasks

This section describes tasks performed in basic SystemBuild use.

**NOTE:** Many of the operations described in this guide can be accomplished by alternative methods and shortcuts. To simplify the presentation, only one method is specified in most cases.

### 4.3.1 Creating a New SuperBlock

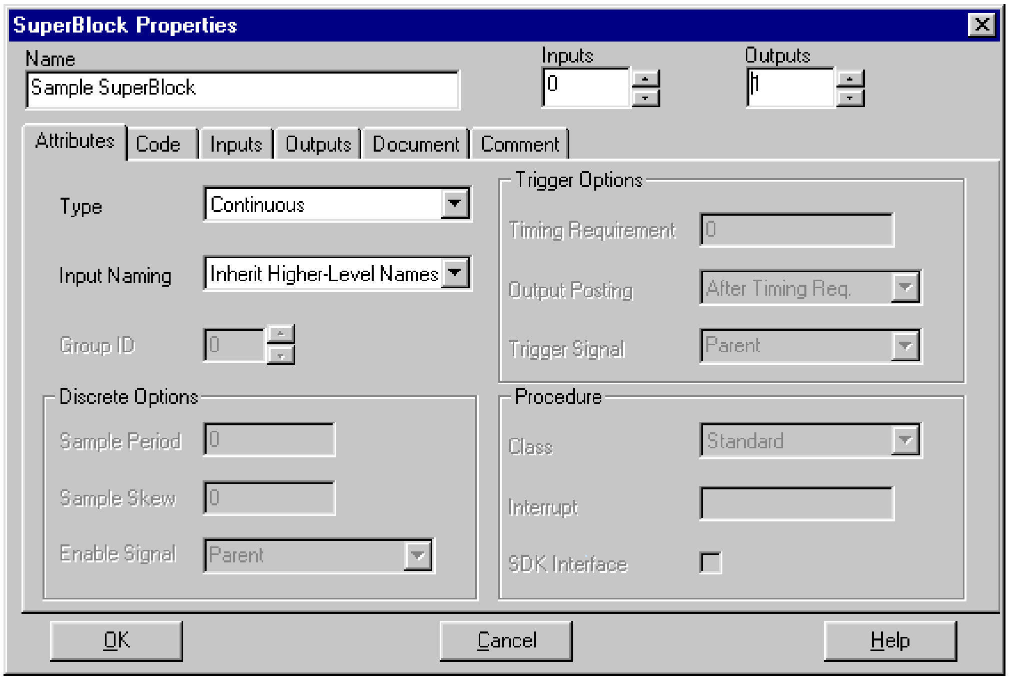
The Catalog Browser can be used to create a new top-level SuperBlock. If this window is not activated, click on the Catalog Browser's window frame to raise it to the top, or select Window→Catalog Browser from the Editor.

To create a new top-level SuperBlock and define its properties:

1. Select File→New→SuperBlock.

   The SuperBlock Properties dialog appears. Use this dialog to define properties of the SuperBlock, such as its name, type (continuous or discrete), and number of inputs and outputs.

Figure 4-4 **SuperBlock Properties Dialog for Creating a New SuperBlock**



SuperBlock Properties

Name: Sample SuperBlock
Inputs: 0
Outputs: 1

Attributes | Code | Inputs | Outputs | Document | Comment

Type: Continuous
Input Naming: Inherit Higher-Level Names
Group ID: 0

Trigger Options
Timing Requirement: 0
Output Posting: After Timing Req.
Trigger Signal: Parent

Discrete Options
Sample Period: 0
Sample Skew: 0
Enable Signal: Parent

Procedure
Class: Standard
Interrupt:
SDK Interface: ☐

2. With the SuperBlock Properties dialog (see Figure 4-4), perform the following steps:

   a. Click in the Name edit field, and type:

   ```
   Sample SuperBlock
   ```

   b. In the Outputs field, set the number of outputs to **1**.

   c. Click OK to verify creation of the SuperBlock.

   The SystemBuild Editor (or Editor window) now appears; it contains an Info Bar, which displays the SuperBlock name (Sample SuperBlock), type (Continuous), and other relevant information (0 inputs and 1 output) about the current SuperBlock.

## *4.3.2 Creating a New Block in a SuperBlock*

You create a new block in a SuperBlock by dragging it from the Palette Browser into the Editor window (see Figure 4-5).

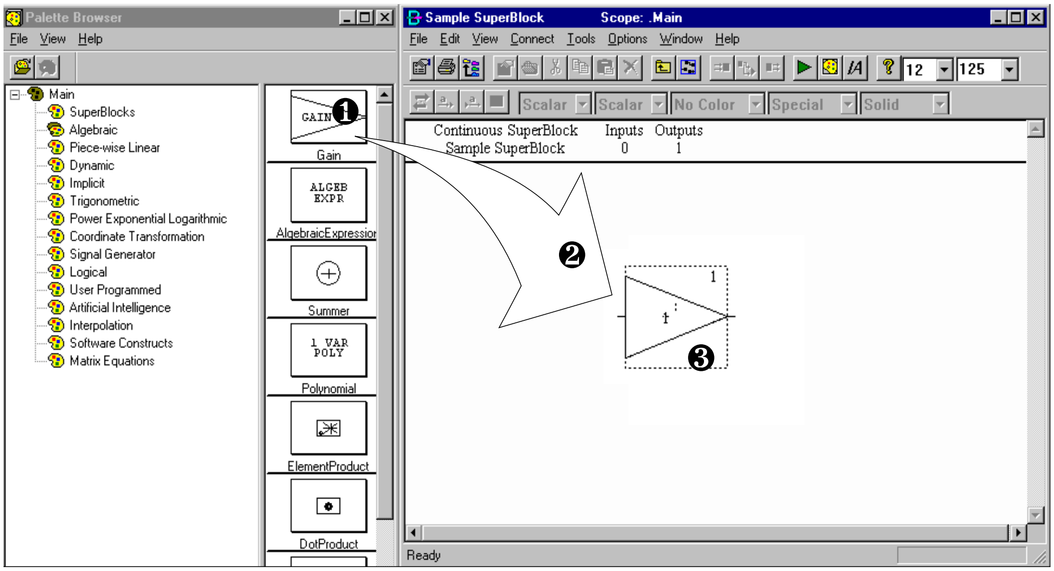
To create a new block in a SuperBlock:

1. With your SuperBlock displayed in the Editor window, select Window→Palette Browser to open the Palette Browser.

   Reposition your windows so that both the Palette Browser and the Editor are visible.

2. Click the Algebraic palette in the Palette Browser.
3. Move the mouse cursor over the Gain block icon. Press and hold down MB1. (❶)
4. While holding down MB1, drag the mouse cursor into the Editor window. (❷)
5. With the mouse cursor within the Editor window, release MB1 to complete the drag-drop operation. (❸)

Figure 4-5 **Creating a New Block Using the Palette Browser**



## 4.3.3 Loading a Model File

Loading a model file opens a previously saved SystemBuild diagram. Once loaded, you can then edit or simulate that model.

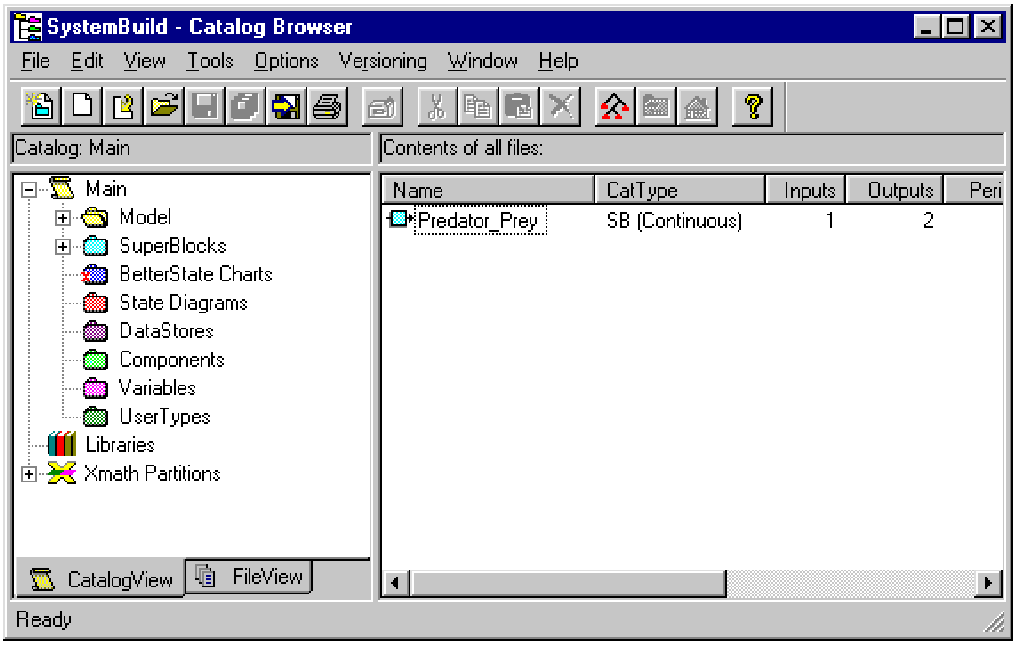
To load the **pred_prey** catalog file from the Xmath command area:

```
load "$SYSBLD\demo\predprey_demo\pred_prey.cat";
```

$SYSBLD is an environment variable that specifies your SystemBuild directory, defined automatically when you start Xmath. Environment variables are recognized only on the Xmath command line. SystemBuild catalogs can also be loaded from the Catalog Browser, but there you must specify the full pathname of the catalog directory.

After the load completes, the Catalog Browser lists the contents of the model (see Figure 4-6).

Figure 4-6 **Predator-Prey Model Loaded into Catalog Browser**



| Name | CatType | Inputs | Outputs | Peri |
|---|---|---|---|---|
| Predator_Prey | SB (Continuous) | 1 | 2 | |

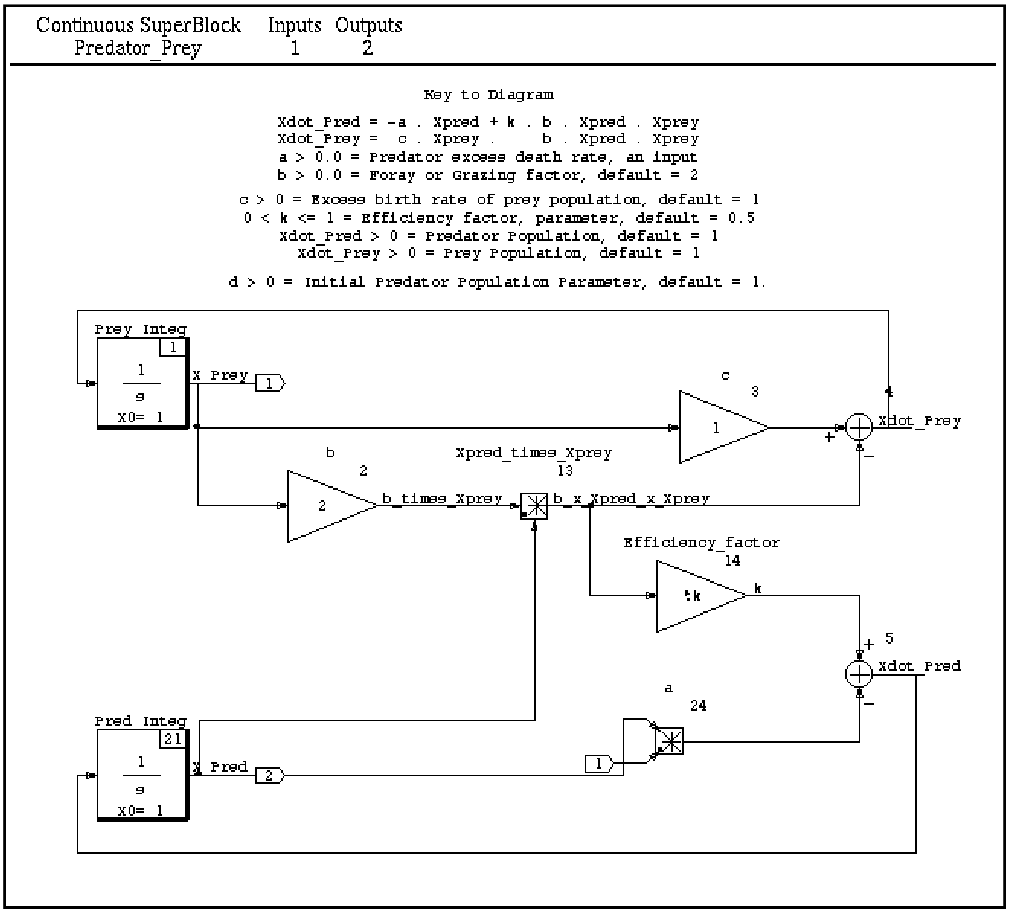
## 4.3.4 Opening a SuperBlock in the Editor

After loading a model, you can open a SuperBlock in the editor to edit or view it:

1. If needed, load the predator-prey model as described in *4.3.3 Loading a Model File*, p.57.
2. To see a list of all SuperBlocks currently loaded, click in the left pane on the SuperBlocks node. A listing of the SuperBlocks appears in the right pane.
3. In the right pane, double-click the Predator_Prey SuperBlock.

   This opens an Editor window and displays the contents of the SuperBlock (see Figure 4-7).

Figure 4-7 **Predator_Prey SuperBlock Displayed in Editor Window**



```
Continuous SuperBlock    Inputs  Outputs
    Predator_Prey          1       2

                    Key to Diagram

    Xdot_Pred = -a . Xpred + k . b . Xpred . Xprey
    Xdot_Prey =  c . Xprey .     b . Xpred . Xprey
    a > 0.0 = Predator excess death rate, an input
    b > 0.0 = Foray or Grazing factor, default = 2

 c > 0 = Excess birth rate of prey population, default = 1
 0 < k <= 1 = Efficiency factor, parameter, default = 0.5
    Xdot_Pred > 0 = Predator Population, default = 1
      Xdot_Prey > 0 = Prey Population, default = 1

d > 0 = Initial Predator Population Parameter, default = 1.
```

### *4.3.5 Simulating the Model from the Xmath Commands Window*

After a model is loaded, you can simulate it. This section describes performing a simulation directly from the Xmath command area and also from the SystemBuild Editor. Load the predator-prey model as described in *4.3.3 Loading a Model File*, p.57.

To simulate the predator-prey model from the Xmath commands area:

1. Activate the Xmath window by clicking on the Xmath window's frame.
2. Click within the Xmath command area.
3. Create a time vector and assign the input vector to a variable:

   ```
   t=[0:.01: 50]';
   u=ones(t);
   ```

4. Input the value of the efficiency factor **k**:

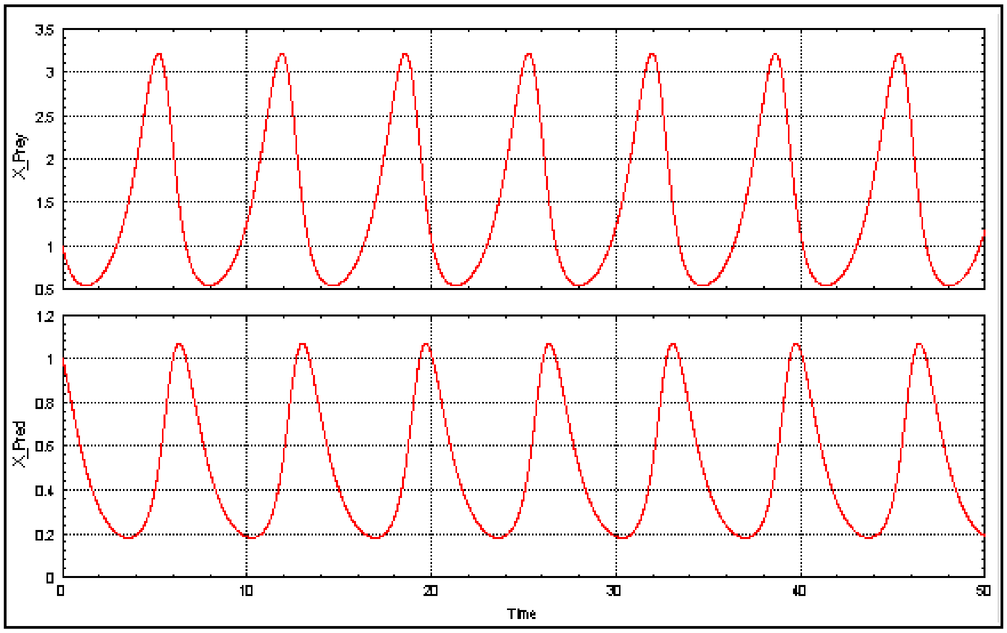
   ```
   k=.333;
   ```

5. In the Xmath command area, type:

   ```
   y=sim("Predator_Prey",t,u,{graph});
   ```

   Watch the log area of the Xmath window as the model is analyzed and simulated. The simulation output plot, which appears in a separate window, is shown in Figure 4-8

Figure 4-8 **Plot of a Predator-Prey Simulation Output**



To simulate the model from the SystemBuild Editor:

**NOTE:** If you performed the simulation from the Xmath command line above and haven't deleted the variables, you can start at Step 5.

1. Activate the Xmath window by clicking on the Xmath window's frame.
2. Click within the Xmath command area.
3. Create a time vector and assign the input vector to a variable:

```
t=[0:.01: 50]';
u=ones(t);
```

4. Input the value of the efficiency factor **k**:

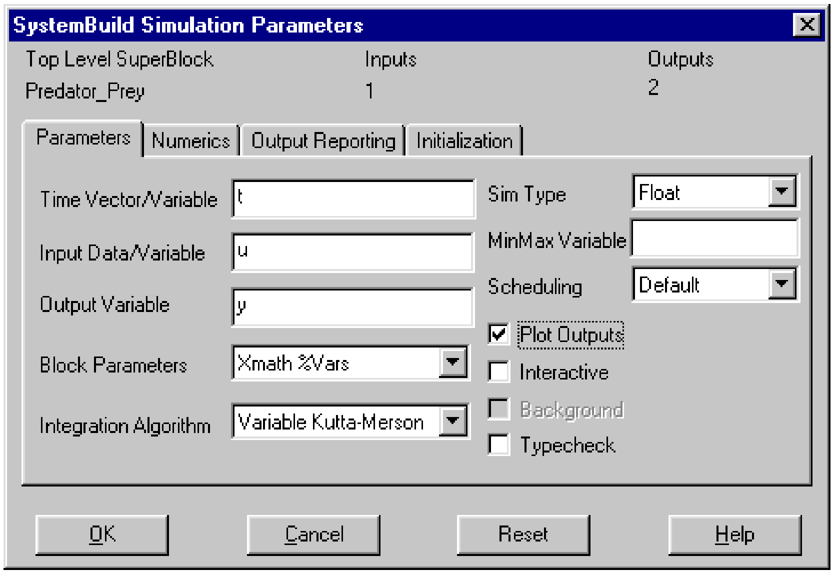
```
k=.333;
```

5. Activate the Predator_Prey SuperBlock SystemBuild Editor (see *4.3.4 Opening a SuperBlock in the Editor*, p.58).

6. In the SystemBuild Editor, select Tools→Simulate from the pull-down menu.

   The SystemBuild Simulation Parameters dialog appears (see Figure 4-9).

Figure 4-9 **SystemBuild Simulation Parameters Dialog**


SystemBuild Simulation Parameters

| Top Level SuperBlock | Inputs | Outputs |
|---|---|---|
| Predator_Prey | 1 | 2 |

Time Vector/Variable: t
Input Data/Variable: u
Output Variable: y
Block Parameters: Xmath %Vars
Integration Algorithm: Variable Kutta-Merson
Sim Type: Float
MinMax Variable:
Scheduling: Default
[✓] Plot Outputs
[ ] Interactive
[ ] Background
[ ] Typecheck

7. In the SystemBuild Simulation Parameters dialog, enter **t** in the TimeVector/Variable field, **u** in the Input Data Variable field, and **y** in the Output Variable field; enable the Plot Outputs check box, and click OK.

   You can monitor the log area of the Xmath window as the model is analyzed and simulated. The simulation output plot appears in a separate window(Figure 4-8).

## *4.3.6 Deleting a SuperBlock*

To delete a SuperBlock:

From the Catalog Browser (either pane, provided the SuperBlock names appear), select a SuperBlock. Then select Edit→Delete.

⚠ **CAUTION:** Deletion of SuperBlocks cannot be undone.

**NOTE:** Once you delete a SuperBlock, it is no longer visible to the Catalog Browser. Any SuperBlock that references the deleted SuperBlock, contains an "Undefined" SuperBlock indicator.

## 4.3.7 Navigating a SuperBlock Hierarchy

The use of hierarchy in your SystemBuild models is crucial to the successful implementation of a system. As mentioned earlier, you use SuperBlocks to create a model hierarchy. This section presents some of the methods for navigating up and down a SuperBlock hierarchy.

You need a fresh start for this exercise:

1. Delete all of the SuperBlocks you may have created, or exit the Catalog Browser (File→Exit), and restart SystemBuild.
2. Load a model with a SuperBlock hierarchy:

   From the Xmath command area, enter:

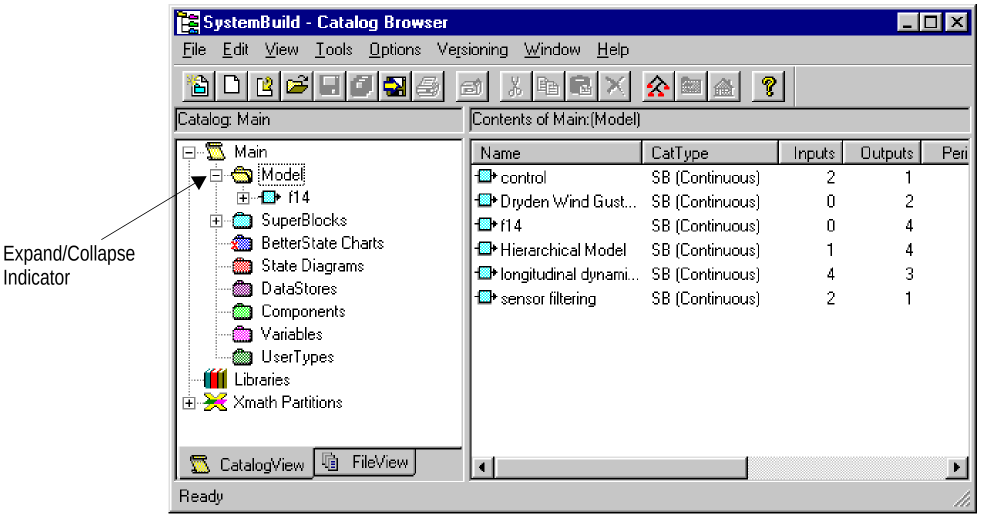
   ```
   load "$SYSBLD\demo\f14_demo\f14new.cat";
   ```

### *Navigating with the Catalog Browser*

When you navigate with the Catalog Browser, you use the left pane of the browser to expand and collapse SuperBlocks within the tree.

After the **f14** model is loaded, the Catalog Browser displays the types of catalog objects (SuperBlocks, BetterState Charts, and so forth) in the left pane. Click on the Model folder icon to see a full list of model objects, including SuperBlocks, in the right pane (see Figure 4-10).

Figure 4-10 **Catalog Browser After Loading the f14 Model**



| Name | CatType | Inputs | Outputs | Peri |
|---|---|---|---|---|
| control | SB (Continuous) | 2 | 1 | |
| Dryden Wind Gust... | SB (Continuous) | 0 | 2 | |
| f14 | SB (Continuous) | 0 | 4 | |
| Hierarchical Model | SB (Continuous) | 1 | 4 | |
| longitudinal dynami... | SB (Continuous) | 4 | 3 | |
| sensor filtering | SB (Continuous) | 2 | 1 | |

Notice the expand/collapse indicator for the Model folder (left pane). Plus (+), indicates that the folder is collapsed (and at least one additional hierarchical layer can be expanded).

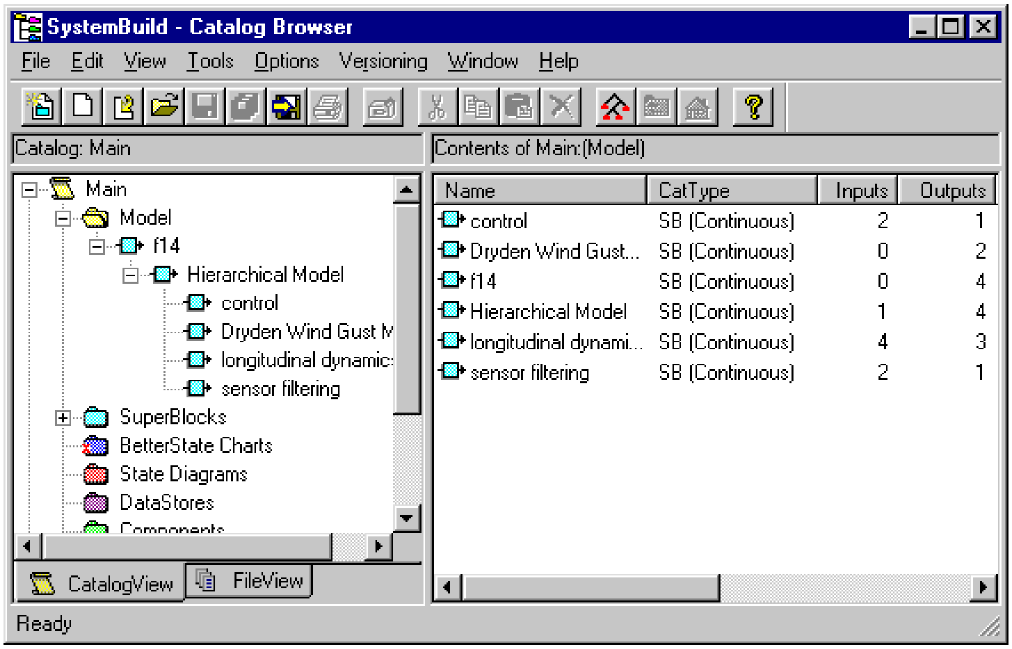
To navigate from the Catalog Browser:

1. Expand the hierarchy of the Model folder in the left pane by double-clicking the folder or by single-clicking the expand indicator, the plus (+) sign.
2. Continue expanding each level of the model (see Figure 4-11).
3. Open the SuperBlock named **sensor filtering** by double-clicking the SuperBlock in the right pane.

   An Editor window appears with the selected SuperBlock on view. You can edit the contents of the SuperBlock.

   To edit another SuperBlock, return to the Catalog Browser, and double-click any SuperBlock in the right pane.

Figure 4-11 **Expanded Hierarchy of the f14 Model**



| Name | CatType | Inputs | Outputs |
|---|---|---|---|
| control | SB (Continuous) | 2 | 1 |
| Dryden Wind Gust... | SB (Continuous) | 0 | 2 |
| f14 | SB (Continuous) | 0 | 4 |
| Hierarchical Model | SB (Continuous) | 1 | 4 |
| longitudinal dynami... | SB (Continuous) | 4 | 3 |
| sensor filtering | SB (Continuous) | 2 | 1 |

### *Navigating from the Editor Window*

You can navigate up and down a hierarchy from within the Editor window using menu items.

To navigate from the Editor window:

1. From the Catalog Browser, open the **sensor filtering** SuperBlock (see Step 1–Step 3 above).
2. With the Editor window, select View→Parent→Hierarchical Model to view this SuperBlock's parent.

   The Editor window now displays the SuperBlock named **Hierarchical Model**.
3. To move down the SuperBlock hierarchy in the Editor window, click the desired SuperBlock icon; then select Edit→Open.

   The Editor window now displays the selected SuperBlock.

**NOTE:** The MATRIX$_X$ demo package includes a simulation of the F14 model.

To run the demo simulation:

1. Enter the command **demo** from the Xmath Commands window.

   The Xmath Demos dialog appears.

2. Enable the SystemBuild Demos... radio button and press OK.

   If the Save SysBld & Xmath workspace dialog appears, enable the Yes->Save radio button and click OK.

   The SystemBuild Demos dialog appears.

3. Enable the F14 Jet Simulation radio button and click OK.

   Follow the prompts to run the demo.

### 4.3.8 Printing from the Editor Window

SystemBuild provides the standard Windows Print dialog for printing from the Editor window on Windows operating systems.

To print the contents of an Editor window:

Select File→Print.

Printing uses the settings you last made in the Page Setup dialog, also available from the File menu. The default printer is specified by the **$PRINTER** environment variable.

## 4.4 SystemBuild Tutorial

This section presents basic procedures used in the design, development, and simulation of SystemBuild block diagrams:

- *Designing a Block Diagram*
- *Creating and Editing a Block Diagram*
- *Simulating a SuperBlock*
- *Encapsulating a SuperBlock*
- *Using a BetterStateChart Block to Model Events*

**NOTE:** This tutorial is designed to lead you through the construction of basic and intermediate SystemBuild block diagrams and BetterState charts. If instead, you wish to examine the completed models, the block diagrams and state charts constructed in this tutorial can be loaded into the CatalogBrowser by executing the following commands from the Xmath Commands window.

```
load "$SYSBLD\examples\gs_tutorial\vibe1.cat";
```

The **vibe1** catalog contains a solution of the basic spring-mass damper constructed in the first four sections of the SystemBuild tutorial. This catalog can be used to skip ahead to *4.4.5 Using a BetterStateChart Block to Model Events*.

```
load "$SYSBLD\examples\gs_tutorial\vibe2.cat";
```

**vibe2** contains a solution of the intermediate spring-mass damper model with event modeling using the BetterState block.

```
load "$SYSBLD\examples\gs_tutorial\vibe3.cat";
```

The tutorial concludes with a challenge exercise. **vibe3** contains a solution.

### 4.4.1 Designing a Block Diagram

To develop a useful block diagram representation of a physical system, you need to know:

- the analytic behavior of the physical system components.
- how block diagram elements correspond to physical system components.
- how to use SystemBuild editors and dialogs to develop a block diagram.

### The Spring-Mass Damper Model

In this tutorial, you develop a block diagram representation for a physical system incorporating a spring-mass damper. The following well-known equation is derived by making standard assumptions about the system behavior:

$$F(t) = m\ddot{x}(t) + c\dot{x}(t) + kx(t) \tag{4-1}$$

Equation 4.1 follows the common convention of indicating time derivatives by dots. $F$ denotes the external force applied to a mass $m$. The spring introduces a force proportional to and opposite its elongation. The scalar value $k$ depends on the spring, and is called its *stiffness*. Motion of the mass is damped by a force proportional to and opposite its velocity. The scalar value $c$ is referred to as the *damping* constant.

Equation 4.1 provides a mathematical model that can be represented directly by a SystemBuild block diagram. Alternatively, the system analysis that produced the equation can be used to develop a block diagram. But first you need to know how SystemBuild blocks model physical systems.

### SystemBuild Block Basics

SystemBuild block diagrams are composed of interconnected blocks. Most blocks in a block diagram receive input signals and produce output signals. A signal is a scalar value that can vary over time. The process performed by a given block to produce its outputs may depend on user-defined properties.

Blocks are combined in a block diagram by connecting outputs to inputs. A group of interconnected blocks can collectively define a SuperBlock. Most SuperBlocks also have input and output signals connected to one or more of its blocks. A SuperBlock can be simulated by specifying each of its input signals with respect to some time vector. A SuperBlock can be a building block in a higher-level SuperBlock. A SuperBlock hierarchy is defined in this manner.

In this tutorial, you use several basic SystemBuild block types to develop the block diagrams of a SuperBlock hierarchy. The introductory block information included here is intended to motivate that development; it is not a complete description of the capabilities and options of the block types.

### Constant Block

A Constant block has no input signal. Its output signal is a constant.

### ElementDivision Block

The output signal of an ElementDivision block is input signal #1 divided by input signal #2.

### Integrator Block

The output signal of an Integrator block is the integration over time of its input signal. The initial value of the output signal of an Integrator can be defined, and an Integrator can be triggered to reset its output signal to a specified value.

### Gain Block

The output signal of a Gain block is its input signal multiplied by a constant.

### Summer Block

The output signal of a Summer block is the sum of its input signals. Each input signal has a sign which determines if it added to, or subtracted from the output signal.

### ElementProduct Block

The output signal of an ElementProduct block is the product of its input signals.

### ZeroCrossing Block

During a simulation, a ZeroCrossing block detects the instant at which its input signal crosses zero. The simulation is recomputed to include this additional time point. The ZeroCrossing block output signal toggles between zero and one at such crossings.

### BetterState Block

A BetterState block implements a finite state machine. Input signals trigger state transitions. (When a BetterState block is used in a continuous SuperBlock, its events must be triggered by ZeroCrossing blocks.) Output signals are generated by user-defined code associated with states and state transitions.

## Getting Started on a Design

Identifying the inputs and outputs of the top-level SuperBlock is a good way to start a design. For the spring-mass damper, there is one input—the external force applied to the mass. The outputs of a top-level SuperBlock are the signals you choose to monitor for the purpose of observing the behavior of the modeled system. Position and velocity of the mass are useful choices here.

Next, make a rough plan for modeling and connecting elements of the physical system. For the spring-mass damper, you have force acting on a mass. That determines an acceleration that can be integrated to give velocity and position. The velocity and position can be used to determine the spring and damping components of the force.

In developing a block diagram, try to identify subsystems that can be built independently. Simulating submodels can help to verify that a complex SuperBlock hierarchy simulation is valid. Also, by encapsulating the subsystems in your area of interest, you can optimize reuse in subsequent designs.

In this tutorial, you start building the spring-mass damper by modeling force acting on a mass.

### 4.4.2 Creating and Editing a Block Diagram

A block diagram is the graphical representation of a SystemBuilld model. To create and edit a new block diagram:

1. Create a SuperBlock.
2. Add blocks to the block diagram (of the SuperBlock).
3. Edit block properties.
4. Connect blocks to each other.
5. Connect blocks to SuperBlock inputs and outputs.
6. Save the SuperBlock

**NOTE:** Many of the operations described in these procedures can be accomplished by alternative methods and shortcuts. To simplify the instructions, only one method is specified in most cases.

#### *Creating a SuperBlock*

To create a new SuperBlock called **vibe**:

1. Start Xmath as described in *3.2.3 Starting Xmath*, p.34.
2. In the Xmath command area, type:

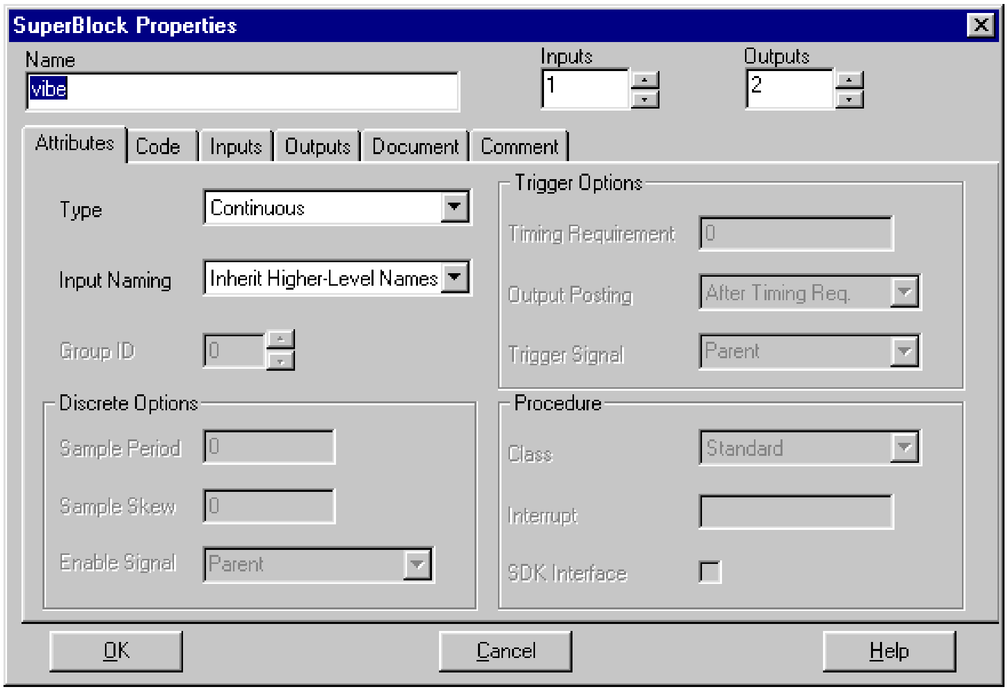
   ```
   build
   ```

   After loading, the SystemBuild Catalog Browser is displayed. If necessary, click on the bar at the top of the window to make it active and bring it to the front.

3. In the Catalog Browser, create a new SuperBlock by selecting File→New→SuperBlock.

   The SuperBlock Properties dialog is displayed. The Attributes tab is selected and all properties of the SuperBlock are set to their default values. (Figure 4-12 shows the SuperBlock Properties dialog as it appears during Step 4 below).

Figure 4-12 **SuperBlock Properties Dialog**


SuperBlock Properties

Name: vibe

Inputs: 1

Outputs: 2

Type: Continuous

Input Naming: Inherit Higher-Level Names

Group ID: 0

Discrete Options

Sample Period: 0

Sample Skew: 0

Enable Signal: Parent

Trigger Options

Timing Requirement: 0

Output Posting: After Timing Req.

Trigger Signal: Parent

Procedure

Class: Standard

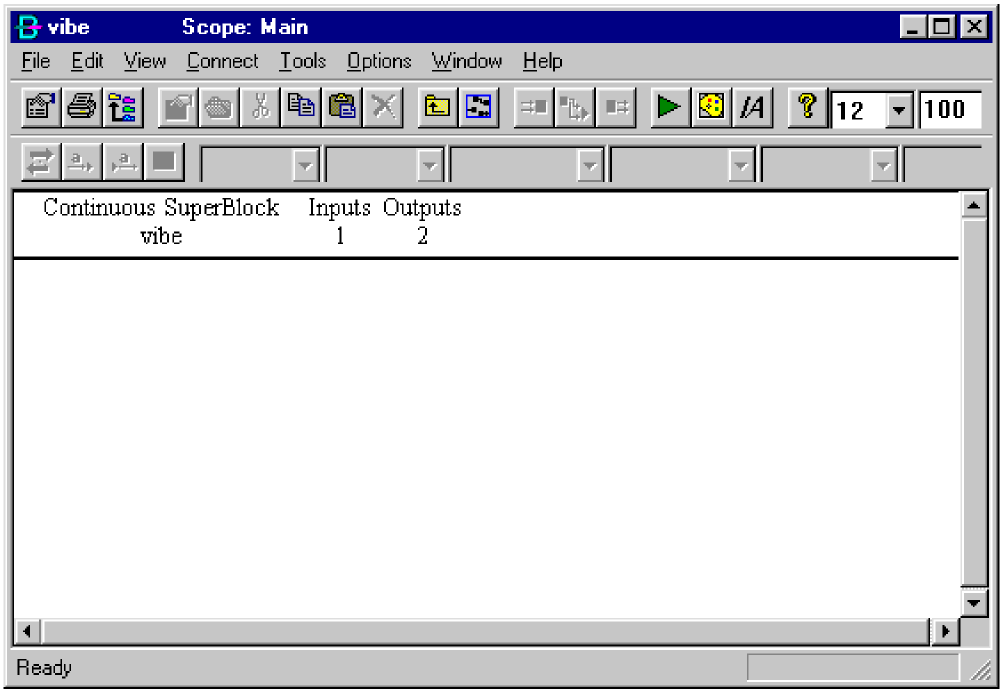
Interrupt:

SDK Interface: ☐

4. With the SuperBlock Properties dialog:
   a. Click in the Name field; name the new SuperBlock **vibe**.
   b. Verify that the Type of the SuperBlock is **Continuous**.
   c. Set the number of Inputs to **1**.
   d. Set the number of Outputs to **2**.
   e. Click OK to accept current values and close the dialog.

      A SuperBlock Editor is displayed for the new SuperBlock. The information bar at the top contains the type, name, and number of inputs and outputs. The area in which block diagrams are constructed is empty (see Figure 4-13).

Figure 4-13 **SuperBlock Editor (Initial View)**



```
Continuous SuperBlock   Inputs  Outputs
        vibe               1       2
```

### *Adding Blocks to the Block Diagram*

In this section, you add four blocks to the block diagram of the **vibe** SuperBlock: a Constant block, an ElementDivision block, and two Integrator blocks.

To add a Constant block to the block diagram:

1. Open the Palette Browser by selecting Window→Palette Browser. Position your windows so that the Palette Browser is alongside the Editor.
2. In the Palette Browser, select the Matrix Equations palette.

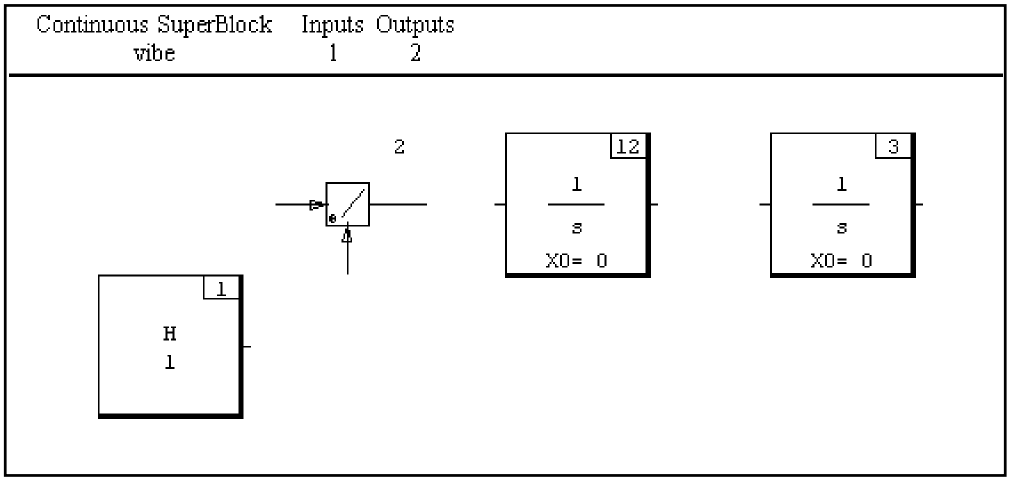
3. With MB1, drag and drop a Constant block from the Matrix Equations palette into the Editor.

In the same manner, add an ElementDivision block from the Algebraic palette, and two Integrator blocks from the Dynamic palette.

Blocks are positioned in a block diagram by dragging with MB1. Position the blocks as in Figure 4-14.

**NOTE:** The block ID is displayed in the upper right corner of each block. Your ID numbers might not match those in Figure 4-14.

Figure 4-14 **Adding Blocks to vibe**



## Editing Block Properties

Each block in a block diagram contains properties that can be edited to adjust aspects of its performance. In this section, you open the Block Properties dialog for each block in **vibe** and edit various properties.

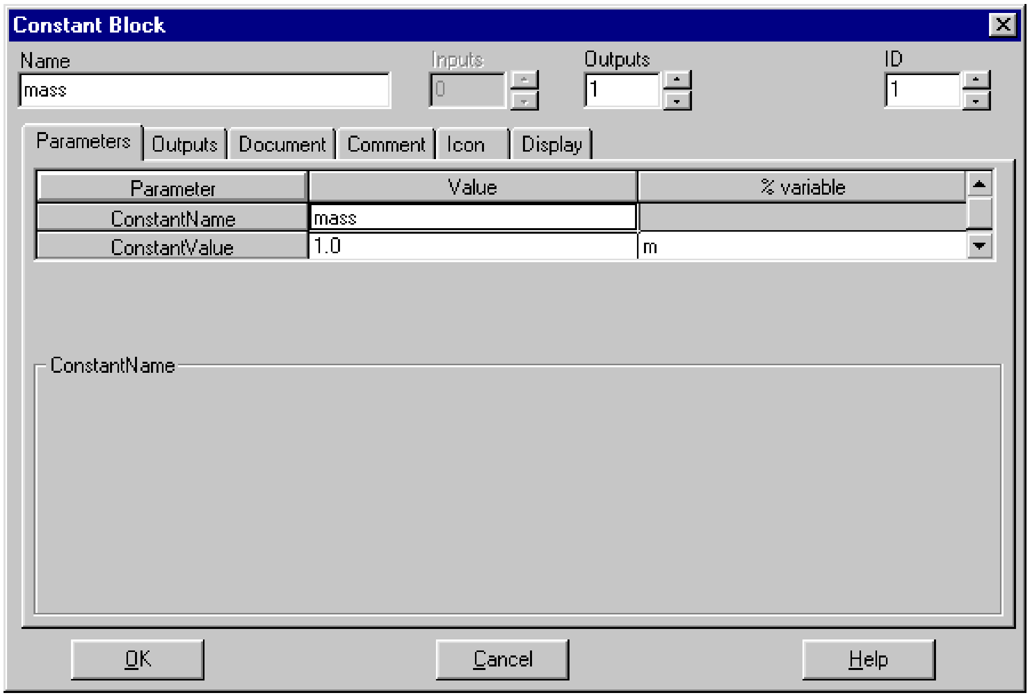
To edit block properties for the Constant block:

1. In the Editor, move the mouse cursor over the Constant block (the left most block as shown in Figure 4-14) and press the Enter key.

   The Constant Block properties dialog is displayed. The Parameters tab is selected and all properties are set to their default values. (Figure 4-15 shows the Constant Block Properties dialog as it appears during Step 3 below).

2. Click in the Name field; name the Constant block **mass**.

Figure 4-15 **Constant Block Properties Dialog: Parameters Tab**



Constant Block

Name: mass | Inputs: 0 | Outputs: 1 | ID: 1

Parameters | Outputs | Document | Comment | Icon | Display

| Parameter | Value | % variable |
|---|---|---|
| ConstantName | mass | |
| ConstantValue | 1.0 | m |

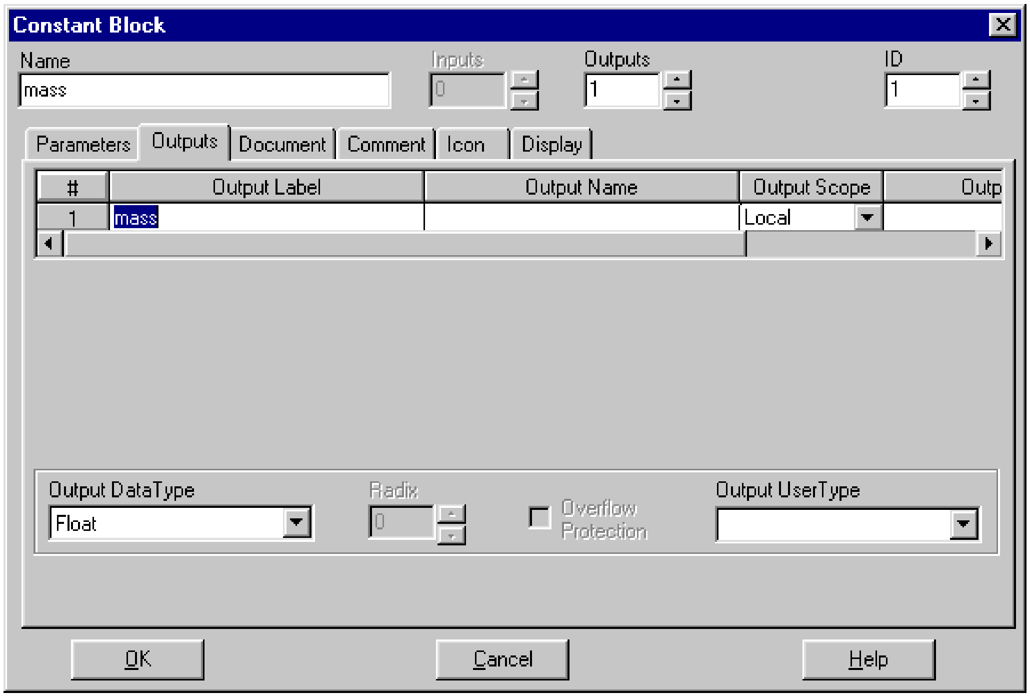
ConstantName

OK | Cancel | Help

3. Verify that the Parameters tab of the Block Properties dialog is selected (see Figure 4-15).
   a. Locate the field in the Parameter table, in the Value column, and the ConstantName row. Click in that field and replace **H** with the name **mass**.
   b. Locate the field in the Parameter table, in the % variable column, and the ConstantValue row. Click in that field and type **m**.

**NOTE:** The % *variable* property allows you to set parameters of your model by assigning values to corresponding variables in the Xmath workspace. Before simulating this model, you set the mass parameter in the Xmath command window by typing an assignment to the Xmath variable **m** (see Step 3, p.86). In this way you can simulate the model with different parameter settings without editing the block diagram.

Figure 4-16 **Constant Block Properties Dialog: Outputs Tab**


Constant Block

Name: mass
Inputs: 0
Outputs: 1
ID: 1

Parameters | Outputs | Document | Comment | Icon | Display

| # | Output Label | Output Name | Output Scope | Outp |
|---|---|---|---|---|
| 1 | mass | | Local | |

Output DataType: Float
Radix: 0
Overflow Protection
Output UserType:

4. Click the Outputs tab. (see Figure 4-16).

   Locate the field in the Outputs table, in the Output Label column, and row **1**. Click in that field and type **mass**.

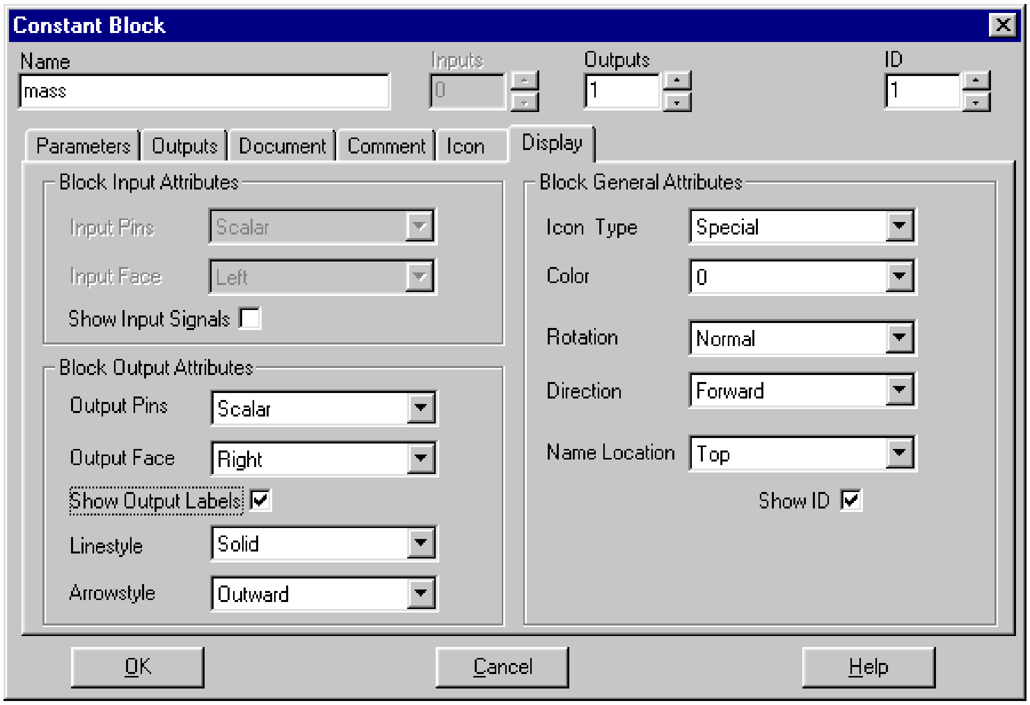
5. Click the Display tab (see Figure 4-17).

   Enable the Show Output Labels check box.

**NOTE:** When the Show Output Labels check box is enabled for a block, the block's output labels are displayed on its output connectors.

6. Click OK to accept current values and close the dialog.

Figure 4-17 **Constant Block Properties Dialog: Display Tab**


Constant Block

Name: mass
Inputs: 0
Outputs: 1
ID: 1

Block Input Attributes
- Input Pins: Scalar
- Input Face: Left
- Show Input Signals: ☐

Block Output Attributes
- Output Pins: Scalar
- Output Face: Right
- Show Output Labels: ✓
- Linestyle: Solid
- Arrowstyle: Outward

Block General Attributes
- Icon Type: Special
- Color: 0
- Rotation: Normal
- Direction: Forward
- Name Location: Top
- Show ID: ✓

Follow the instructions below to edit block properties for the remaining blocks in **vibe**. You give each block a name, label its outputs, and enable the Show Output Labels check box. For additional detail, refer to steps 2, 4, and 5 above in the instructions for editing properties in a Constant block.

You also set additional properties as noted in the instructions.

To edit block properties for the ElementDivision block:

1. In the Editor, move the mouse cursor over the ElementDivision block (second block from the left as shown in Figure 4-14, p.74) and press the Enter key.

   The ElementDivision Block properties dialog is displayed. The Inputs tab is selected and all properties are set to their default values. (Figure 4-18 shows the ElementDivision Block properties dialog as it appears during Step 3 below).

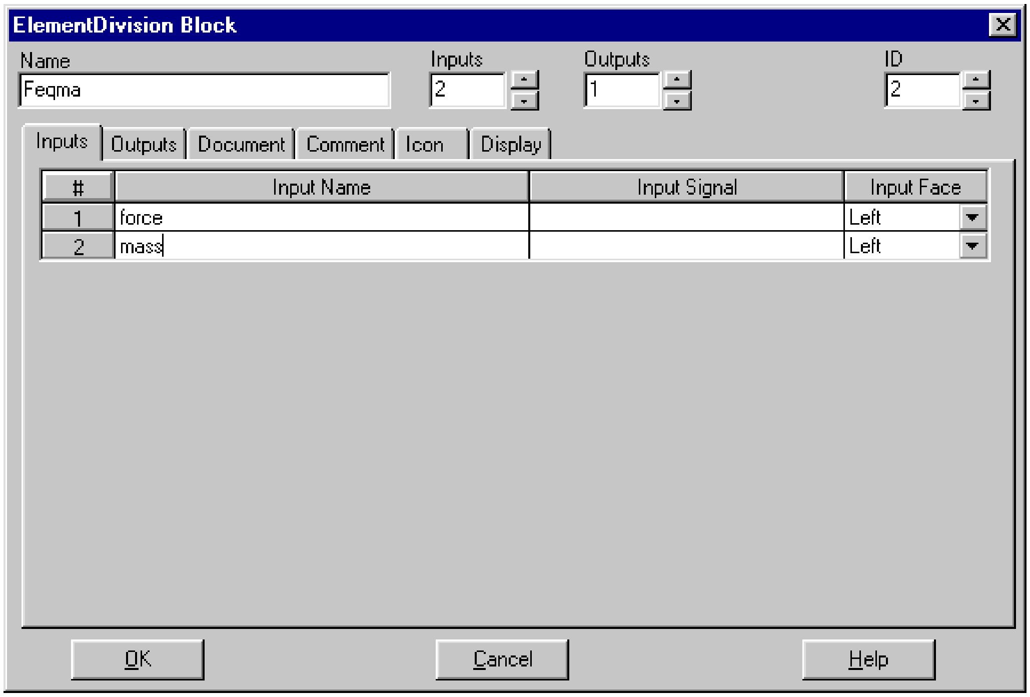
2. Name the block **Feqma**.

3. **Feqma** divides force by mass to calculate acceleration. Naming block inputs can make block connections easier.

   In the Input Name column: name input 1 **force**; name input 2 **mass**.

4. Click the Outputs tab and set the Output Label to **acc**.
5. Click the Display tab and enable the Show Output Labels check box.
6. Click OK to accept current values and close the dialog.

Figure 4-18 **ElementDivision Block Properties Dialog: Inputs Tab**


ElementDivision Block

Name: Feqma
Inputs: 2
Outputs: 1
ID: 2

| # | Input Name | Input Signal | Input Face |
|---|---|---|---|
| 1 | force | | Left |
| 2 | mass | | Left |

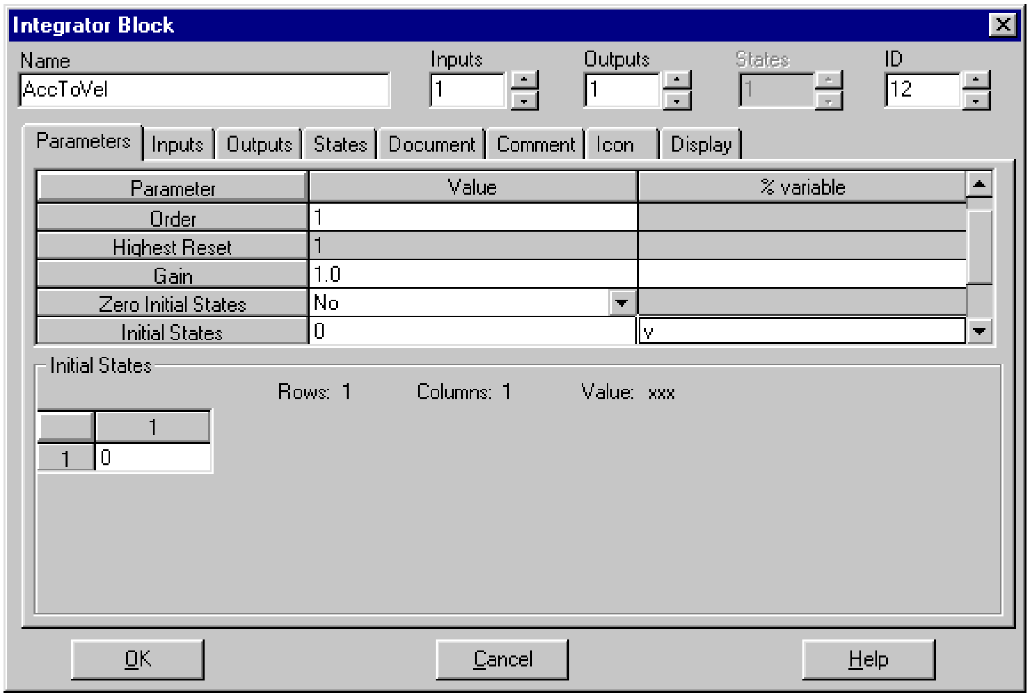
To edit block properties for the first Integrator block:

1. In the Editor, move the mouse cursor over the first Integrator block (second block from the right as shown in Figure 4-14, p.74) and press the Enter key.

   The Integrator Block properties dialog is displayed. The Parameters tab is selected and all properties are set to their default values. (Figure 4-19 shows the Integrator Block properties dialog as it appears during Step 3 below).

2. Name the block **accToVel**.

Figure 4-19 **Integrator Block Properties Dialog: Parameters Tab**



Integrator Block

| Name | Inputs | Outputs | States | ID |
|---|---|---|---|---|
| AccToVel | 1 | 1 | 1 | 12 |

| Parameter | Value | % variable |
|---|---|---|
| Order | 1 | |
| Highest Reset | 1 | |
| Gain | 1.0 | |
| Zero Initial States | No | |
| Initial States | 0 | v |

Initial States

Rows: 1 Columns: 1 Value: xxx

| | 1 |
|---|---|
| 1 | 0 |

3. Verify that the Parameters tab of the Block Properties dialog is selected (see Figure 4-19).

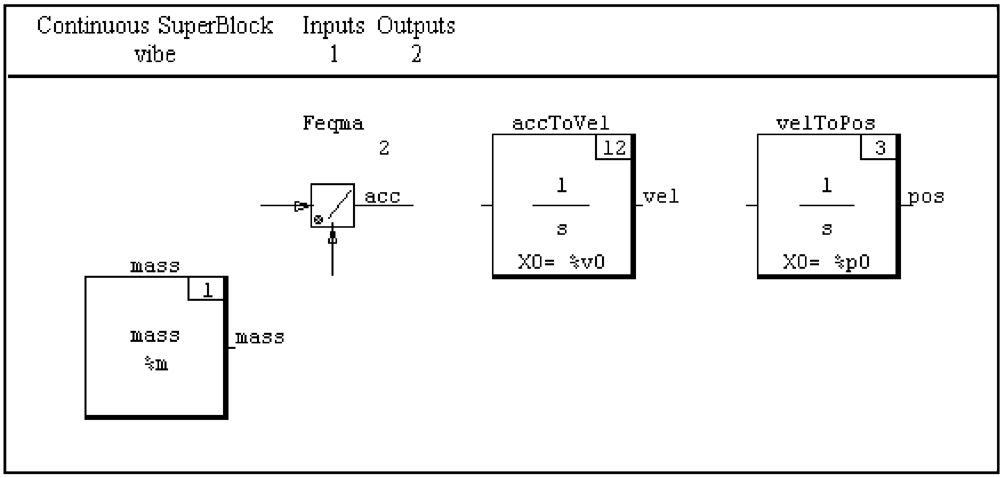
   Scroll down in the Parameter table to access the field in the % variable column, and the Initial States row. Click in that field and type **v0**.

4. Click the Outputs tab and set the Output Label to **vel**.
5. Click the Display tab and enable the Show Output Labels check box.
6. Click OK to accept current values and close the dialog.

Repeat the previous steps for the second integrator block. Name it **velToPos**. Give it an Initial States % variable named **p0**. Set the Output Label to **pos**.

After Editing Block Properties your block diagram should resemble Figure 4-20. The short lines extending out from the sides of the blocks are input and output *pins*. Each output pin is now labeled.

Figure 4-20 **Block Diagram after Editing Block Properties**



## Connecting Blocks

When two blocks are connected, an output signal of one becomes the input signal of the other (a block cannot be connected to itself). Connections between the blocks of a SuperBlock are called *internal connections*. An internal connection is *directed* from an output pin of the *source* block to an input pin of the *destination* block.

**NOTE:** A source block output pin can be connected to zero, one, or more than one destinations. However, a destination block input pin can be connected to at most one source: either a source block output pin or a SuperBlock input.

To connect the Constant block to the AlgebraicExpression block:

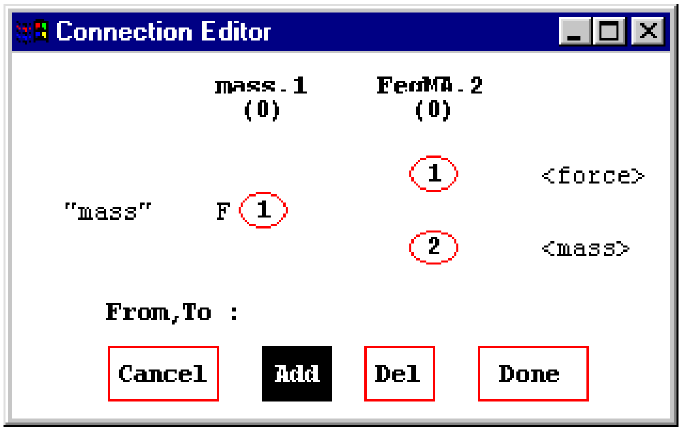
1. Click the Constant block with MB2 (middle mouse button).
2. Click the AlgebraicExpression block with MB2.

   The Connection Editor is displayed (see Figure 4-21).

   - Source block information is displayed on the left.
   - Destination block information is displayed on the right.
   - Output labels and input names are displayed when available.

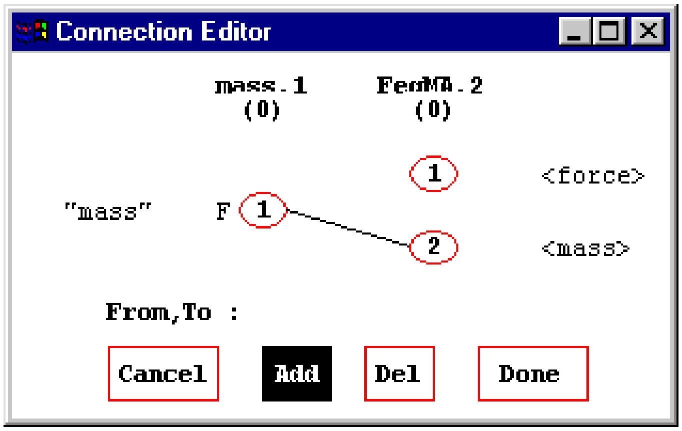
- Data types are displayed when available; **F** (Float) is the default.
- Source block outputs are numbered in a column opposite the numbered inputs of the destination block. Click on the numbers with MB1 to add and delete connections.

Figure 4-21 **Connection Editor**



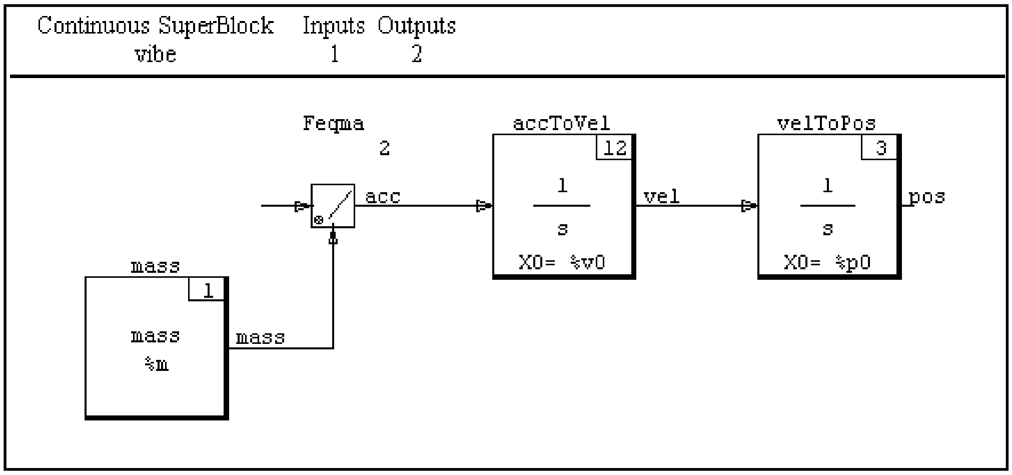
3. Verify that the Add button is highlighted. With MB1, click output 1 (left column), and click input 2 (right column). A line is drawn to indicate the connection as shown in Figure 4-22.
4. Click Done to accept the connection and close the dialog.

Figure 4-22 **Adding a Connection with the Connection Editor**



Complete the internal connections as shown in Figure 4-23. In each case, use MB2 to click the source block, and then the destination block.

Figure 4-23 **Block Diagram after Internal Connections**



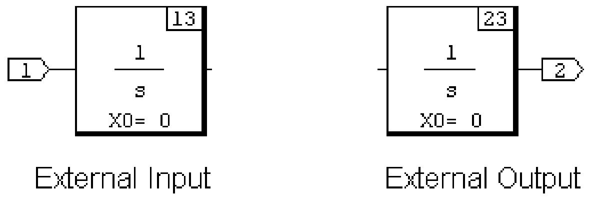
**NOTE:** When only one possible connection can be made between the source and destination blocks, the connection is made without displaying the Connection Editor.

### *Connecting SuperBlock Inputs and Outputs*

Connections from SuperBlock inputs to blocks, and from blocks to SuperBlock outputs are called *external connections*. An external input connection is directed from a SuperBlock input to an input pin of the destination block. An external output connection is directed from an output pin of the source block to a SuperBlock output.

**NOTE:** A SuperBlock input can be connected to zero, one, or more destination blocks. However, a SuperBlock output must be connected to exactly one source block.

Figure 4-24 **External Input and Output Flags**



External input and output connections are represented in a block diagram by flags containing the number of the corresponding input or output. Figure 4-24 shows typical external connection flags: a SuperBlock input 1, and a SuperBlock output 2.

In this section, you connect the external input and output signals for the **vibe** SuperBlock.

To connect the **vibe** SuperBlock input to the **Feqma** ElementDivision block:

1. Move the mouse cursor to an open space in the editor (not over any block), and click with MB2.
2. Click the ElementDivision block with MB2.

   The Connection Editor is displayed. (The layout of information is analogous to that displayed for internal connections, as shown in Figure 4-21).
3. Verify that the Add button is highlighted. With MB1, click external input 1 (left column), and click block input 1 (right column). A line is drawn to indicate the connection.
4. Click Done to accept the connection and close the dialog.

To connect the **accToVel** Integrator block to the **vibe** SuperBlock output 1:

1. Click the **accToVel** Integrator block with MB2.
2. Move the mouse cursor to an open space in the editor (not over any block), and click with MB2.

   The Connection Editor is displayed. (Again, the layout of information is analogous to that displayed for internal connections.)
3. Verify that the Add button is highlighted. With MB1, click block output 1 (left column), and click external output 1 (left column). A line is drawn to indicate the connection.

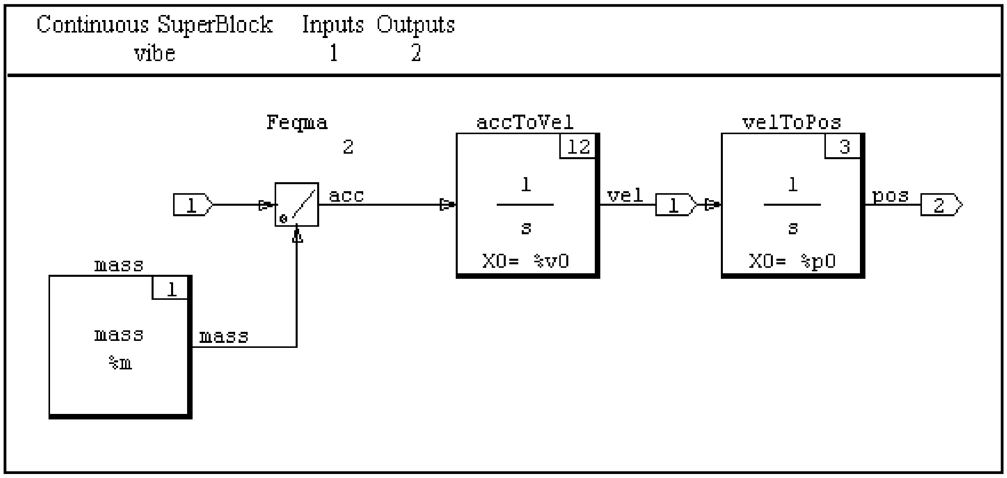
4. Click Done to accept the connection and close the dialog.

Repeat the previous steps to connect the **velToPos** Integrator block to **vibe** SuperBlock output 2.

When complete, your block diagram should resemble Figure 4-25.

**NOTE:** The current block diagram models force acting on a mass. The spring-mass damper model will be completed in *4.4.4 Encapsulating a SuperBlock*, p.88.

Figure 4-25 **Block Diagram after External Connections**



### *Saving a SuperBlock*

To avoid loss of your work, you should save your block diagrams at regular intervals during development. In this section, you save the **vibe** SuperBlock.

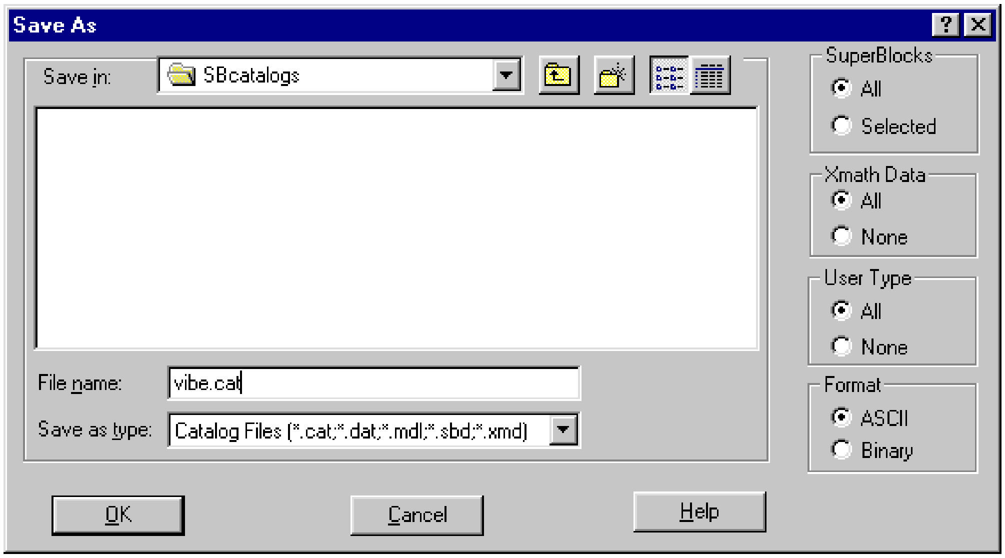
**NOTE:** SuperBlocks and BetterStateChart blocks can be saved individually, or grouped in catalogs. The method described here saves all SuperBlocks and BetterStateChart blocks in the current catalog of the Catalog Browser to a single catalog file.

To save the **vibe** SuperBlock to a catalog file:

1. Make the Editor the active window and update the **vibe** SuperBlock by selecting File→Update.
2. Make the Catalog Browser the active window and refresh its contents by selecting View→Update. A list of SuperBlocks in the current catalog is displayed on the right.
3. Select File→Save As.

   The Save As dialog is displayed. (Figure 4-26 shows the Save As dialog as it appears during Step 5 below).
4. Select a directory in which to save SuperBlock catalogs.

Figure 4-26 **Save As dialog**


Save As

Save in: SBcatalogs

File name: vibe.cat

Save as type: Catalog Files (*.cat;*.dat;*.mdl;*.sbd;*.xmd)

SuperBlocks: [✓ All] Selected

Xmath Data: [✓ All] None

User Type: [✓ All] None

Format: [✓ ASCII] Binary

5. Click in the File name field and type **vibe.cat** (see Figure 4-26).

**NOTE:** If you type a file name with no extension, the default extension **.dat** is appended to the file name when the file is saved.

6. Click OK to save the file and close the dialog.

### 4.4.3 Simulating a SuperBlock

In this section, you simulate the SuperBlock **vibe**. The current model applies an input force to a mass. The resulting velocity and position of the mass are output.

Simulations proceed with respect to a user-defined time vector. Because **vibe** is a *continuous* SuperBlock, its simulation algorithms are relatively independent of the granularity of the time vector sequence. However, the input and output of a continuous simulation are indexed by its time vector, so the granularity must accommodate those factors.

To simulate **vibe**, an input force vector is specified whose elements correspond to those of the time vector. Here, you model the force of gravity. Therefore, the input signal is constant with respect to time.

There are three % variable parameters to define: the mass **m**, the initial velocity **v0**, and the initial position **p0**. In this tutorial, you assign values with MKS units.

To simulate the SuperBlock **vibe** from the Xmath command area:

1. Define a time vector. In the Xmath command area, type:

   ```
   t=[0:0.01:5]';
   ```

**NOTE:** Remember to type the semicolon to suppress output.

   Time must be specified in a column vector. You have created a regular column vector with 501 elements: **0,0.01,0.02,...,4.99,5.00.** The simulation will proceed for 5 seconds, with a computational granularity of 0.01 seconds.

2. Define an input force vector (MKS gravity is approximately -9.8 meters/sec$^2$);

   ```
   u=-9.8*ones(t);
   ```

   The input variable u is constructed to have the same dimensions as the time vector t. Here, u is a column vector of 501 elements.

3. Specify a mass of 1 kilogram, starting at rest and position 0:

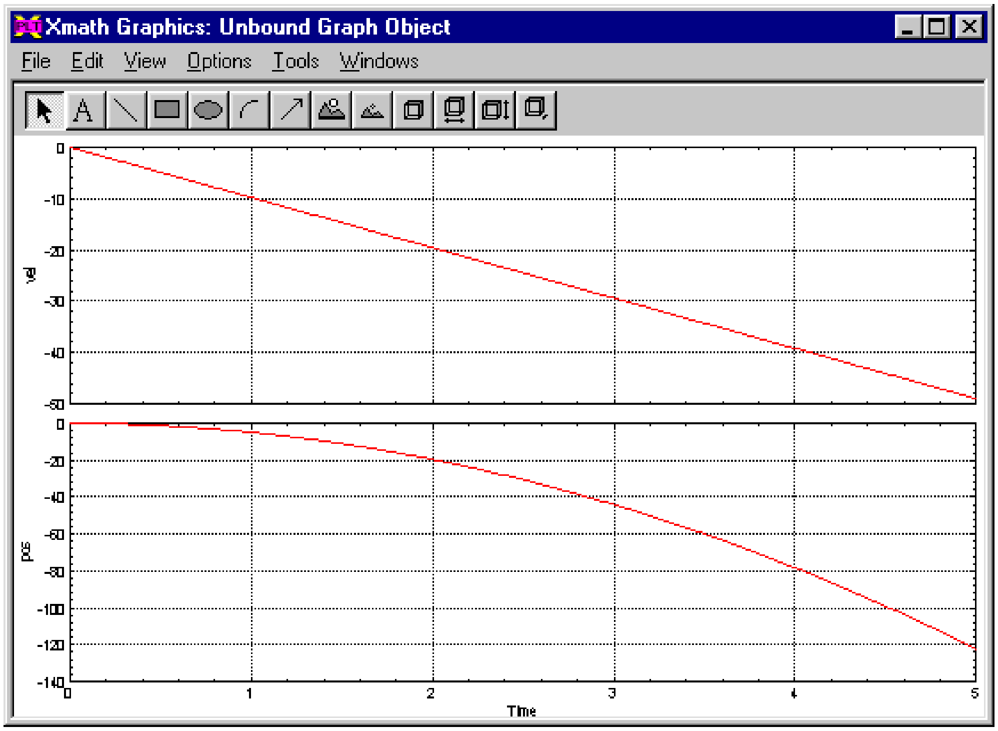
   ```
   m=1;
   v0=0;
   p0=0;
   ```

4. Execute the simulation:

   ```
   y=sim("vibe",t,u,{graph});
   ```

Information about the simulation is displayed in the Xmath log area. When the simulation completes, velocity and position are plotted in an Xmath Graphics window (see Figure 4-27).

Figure 4-27 **Simulation Results**



The simulation result **y** is a pdm consisting of 501 row vectors. Each vector has two scalar elements, a velocity and a position. You can inspect the velocity and position values at a given time by computing its position in the pdm. For example, to see the result at time 3.83 seconds, type:

```
y(3.83*100+1)
```

Exercise the model by simulating other values for the parameters **m**, **v0**, and **p0**. You can also change the input force. Try **u=cos(4*t);**.

### 4.4.4 Encapsulating a SuperBlock

In this section, you encapsulate a SuperBlock and develop a hierarchical block diagram.

To encapsulate a SuperBlock:

1. Make the Editor the active window and update the **vibe** SuperBlock by selecting File→Update.
2. Select the ElementDivision block and both Integrator blocks of the block diagram by holding down the Control key and clicking each in turn with MB1. A heavy rectangular border indicates that a block is selected.
3. Select Edit→Make SuperBlock. The block diagram now shows a new SuperBlock block, along with the **mass** Constant block.

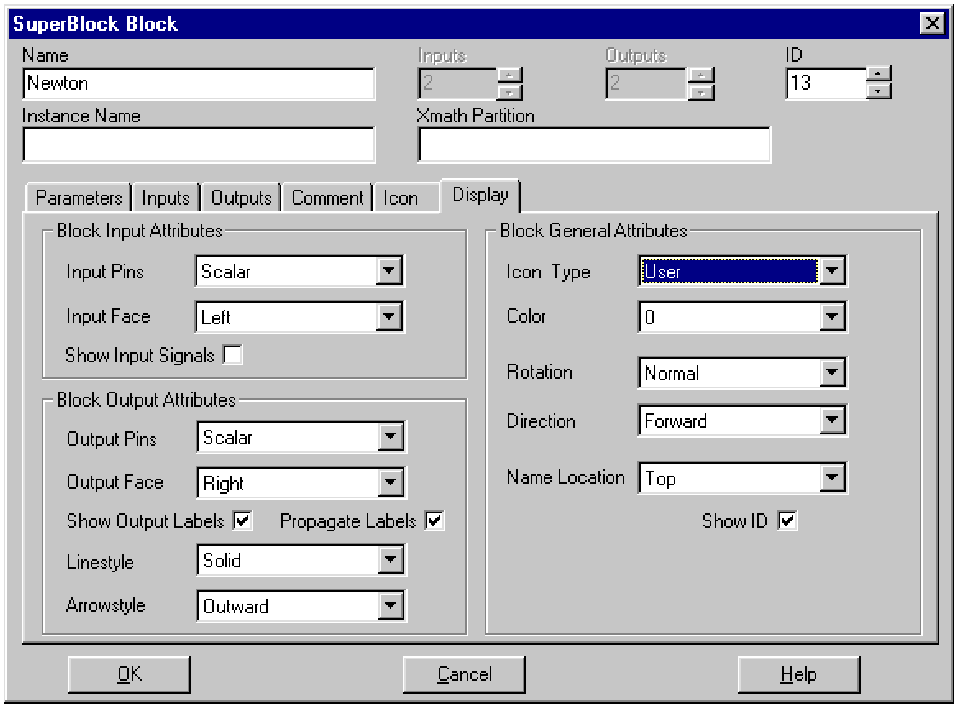
To edit properties of the new SuperBlock block:

1. In the Editor, move the mouse cursor over the SuperBlock block and press the Enter key.

   The SuperBlock Block properties dialog is displayed. The Parameters tab is selected and all properties are set to their default values. (Figure 4-28 shows the SuperBlock Block properties dialog as it appears during Step 3 below).
2. Click in the Name field; replace the default name with **Newton**.
3. Click the Display tab (see Figure 4-28).

   Locate the drop-down combo box labeled Icon Type. Change its value to **User**.

Figure 4-28 **SuperBlock Block Properties: Encapsulation**


SuperBlock Block

| | |
|---|---|
| Name | Newton |
| Inputs | 2 |
| Outputs | 2 |
| ID | 13 |
| Instance Name | |
| Xmath Partition | |

Parameters | Inputs | Outputs | Comment | Icon | Display

Block Input Attributes

| | |
|---|---|
| Input Pins | Scalar |
| Input Face | Left |
| Show Input Signals | ☐ |

Block Output Attributes

| | |
|---|---|
| Output Pins | Scalar |
| Output Face | Right |
| Show Output Labels | ☑ |
| Propagate Labels | ☑ |
| Linestyle | Solid |
| Arrowstyle | Outward |

Block General Attributes

| | |
|---|---|
| Icon Type | User |
| Color | 0 |
| Rotation | Normal |
| Direction | Forward |
| Name Location | Top |
| Show ID | ☑ |

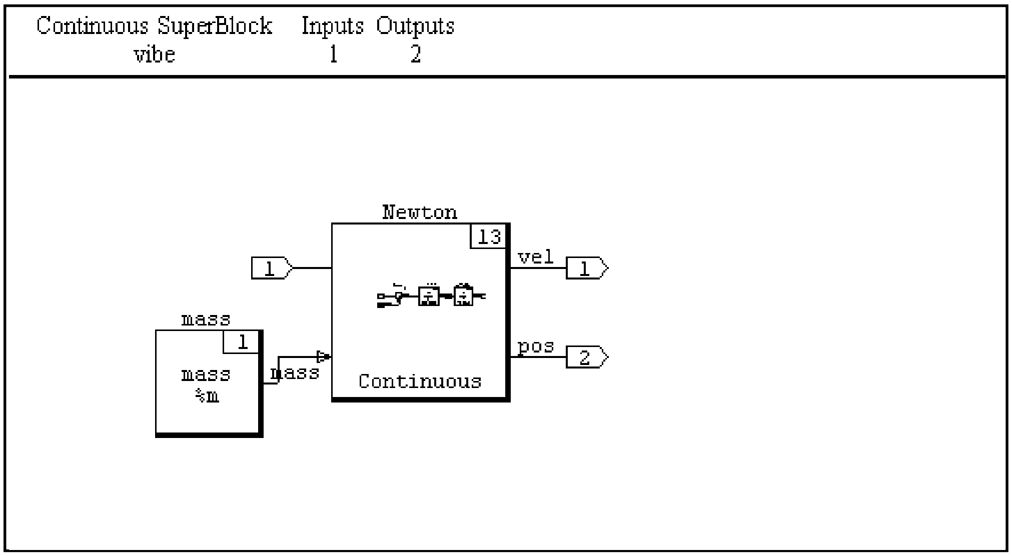
4. Click OK to accept current values and close the dialog.

The Superblock editor now displays the **mass** Constant block, and the **Newton** SuperBlock block. Position and resize the blocks so that they resemble the block diagram displayed in Figure 4-29. Recall, blocks are positioned by dragging with MB1.

To enlarge the **Newton** SuperBlock block, move the mouse cursor over the block, and press e twice.

To reduce the **mass** Constant block, move the mouse cursor over the block, and press r twice.

Figure 4-29 **Newton SuperBlock Block**



To complete a block diagram representation of a damped spring, a few more blocks are needed. A solution is shown in Figure 4-30. Two Gain blocks have been added. One produces the damping force by multiplying velocity by the damping constant. Position and stiffness are used by the other to determine the spring force. These forces are subtracted from the **vibe** input force by a Summer block that passes the resultant force to the Newton SuperBlock block.

To implement the damped spring representation:

1. Add blocks to the block diagram.

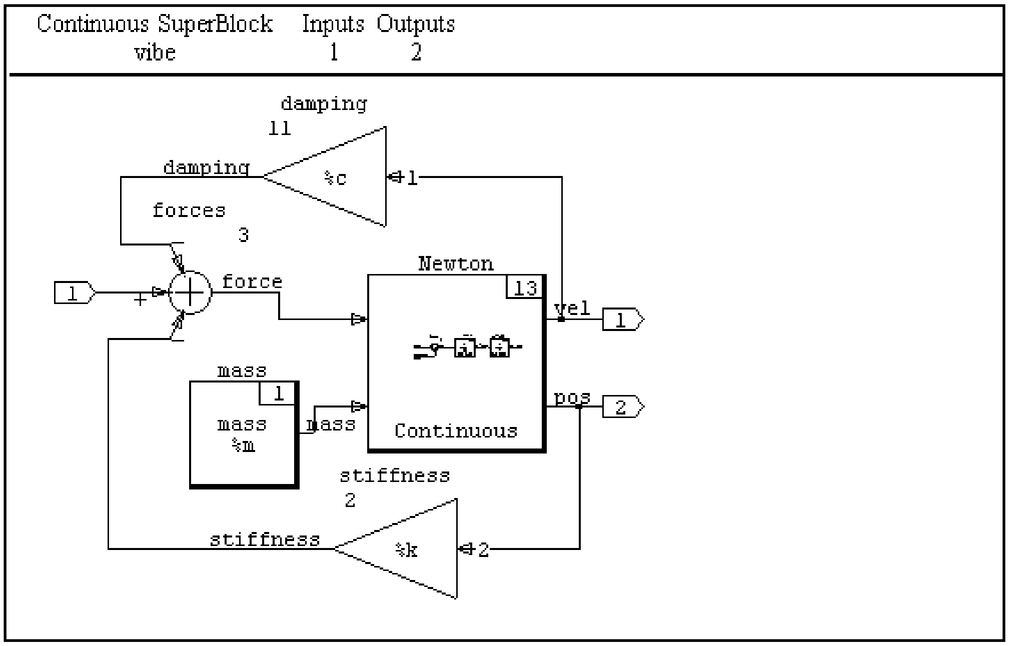
   Open and position the Palette Browser. Select the Algebraic palette. With MB1, drag and drop a Gain block and a Summer block into the Editor.

2. Flip, reduce, and duplicate the Gain block.

   Move the mouse cursor over the Gain block and press f r d (one key at a time).

   You now have two gain blocks that have been flipped horizontally. Input pins are now on the right; output pins are on the left.

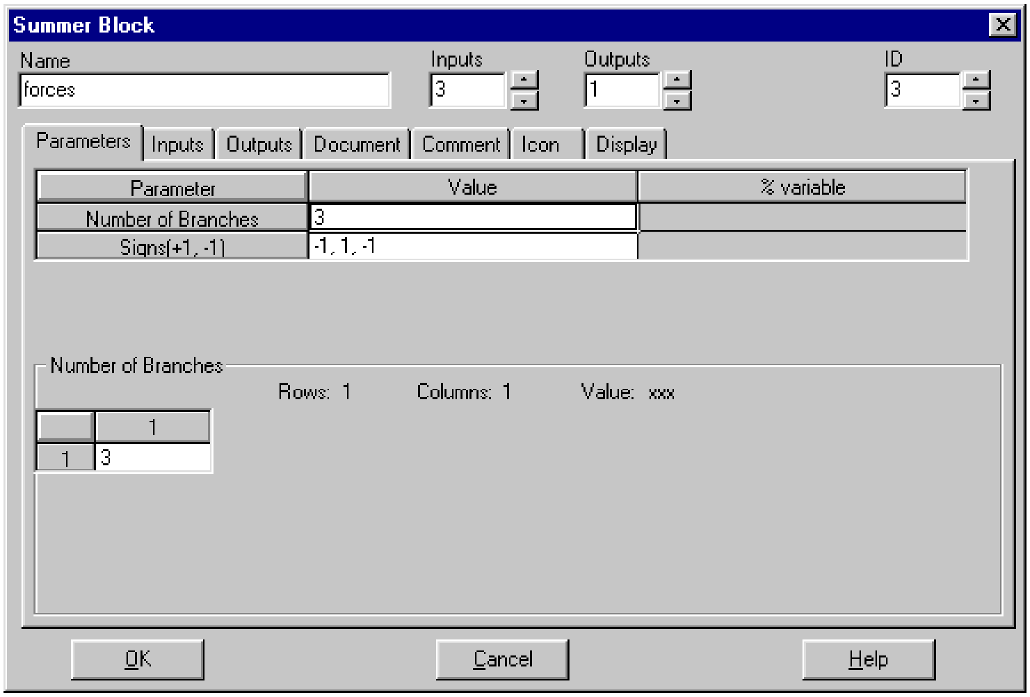
Figure 4-30 **Block Diagram for Damped Spring**



3. Position the new blocks as shown in Figure 4-30.
4. Edit properties for the Summer block.
    a. Move the mouse cursor over the Summer block and press the Enter key.

       The Summer Block properties dialog is displayed. The Parameters tab is selected and all properties are set to their default values. (Figure 4-31 shows the Summer Block properties dialog as it appears during Step e below).
    b. Name the block **forces**.
    c. Change the number of inputs to 3.
    d. In the Parameter table, click in the field in the Value column, and the Number Branches row. Change its value to 3.
    e. In the Parameter table, click in the field in the Value column, and the Signs(+1,-1) row. Change its value to -1,1,-1.

Figure 4-31 **Summer Block Properties Dialog**


Summer Block

Name: forces

Inputs: 3

Outputs: 1

ID: 3

Parameters | Inputs | Outputs | Document | Comment | Icon | Display

| Parameter | Value | % variable |
|---|---|---|
| Number of Branches | 3 | |
| Signs(+1, -1) | -1, 1, -1 | |

Number of Branches

Rows: 1 Columns: 1 Value: xxx

| | 1 |
|---|---|
| 1 | 3 |

f. Click the Inputs tab and name the inputs: **damping**, **external**, **stiffness**.

g. Click the Outputs tab and label the output: **force**,

h. Click the Display tab, change the value of Icon Type to **User**, and enable the Show Output Labels check box.

i. Click OK to accept current values and close the dialog.

5. Edit properties for the Gain blocks.

   You give each Gain block a name, a % variable assignment, label its output, and enable the Show Output Labels check box. For additional detail, refer to the instructions for setting these properties in a Constant block, in the section Editing Block Properties, p.74.

   a. Name the upper Gain block **stiffness**, give it a % variable named **k**, label its output **stiffness**, and enable the Show Output Labels check box.

   b. Name the lower Gain block **damping**, give it a % variable named **c**, label its output **damping**, and enable the Show Output Labels check box.

6. Make block diagram connections as shown in Figure 4-30. For additional detail, see Connecting Blocks, p.80 and Connecting SuperBlock Inputs and Outputs, p.82.

7. Save the current catalog. For additional detail, see Saving a SuperBlock, p.84.

**NOTE:** The current tutorial block diagrams can be loaded into the CatalogBrowser by executing the following command from the Xmath Commands window.

```
load "$SYSBLD\examples\gs_tutorial\vibe1.cat";
```

The **vibe1** catalog contains a solution of the basic spring-mass damper constructed in the first four sections of the SystemBuild tutorial.

To simulate the block diagram representation of the damped spring:

1. In the Xmath command area, verify or reenter the original values for **t**, **u**, **m**, **p0**, and **v0**:

```
t = [0:0.01:5]';
u = -9.8 * ones(t);
m = 1;
v0 = 0;
p0 = 0;
```

2. Define values for the stiffness k and damping constant c:

```
k = 100;
c = 1;
```

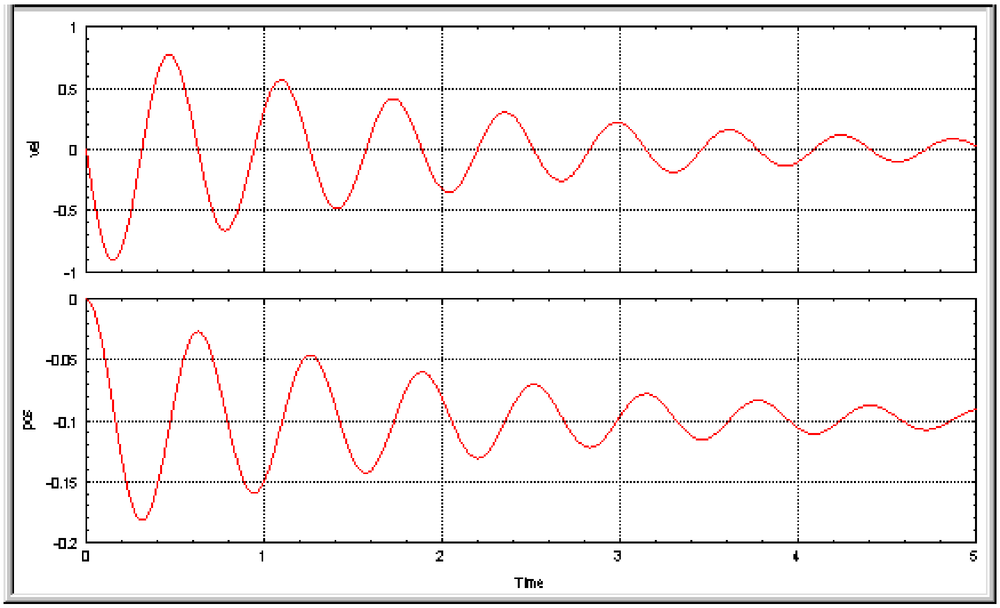
3. Execute the simulation:

```
y = sim("vibe", t, u, {graph});
```

   When the simulation completes, velocity and position are plotted in an Xmath Graphics window (see Figure 4-32).

Exercise the model by simulating other values for the parameters and the input force. Try **u = zeros(t); p0 = 1;**.

Figure 4-32 **Damped Spring Simulation Plot**



### *Exercise*

While using the Palette Browser, you have probably noticed many more pre-defined block types. To read brief descriptions of SystemBuild block types:

1. In the Xmath command area, enter:

   ```
   help blocks
   ```

2. When the Blocks topic appears, scroll down to the tables that list the block types by palette, and read the descriptions there. For more detailed information about block types, follow the table links.

## *4.4.5 Using a BetterStateChart Block to Model Events*

In this section, you enhance the **vibe** SuperBlock model to simulate events.

**NOTE:** This tutorial is designed to lead you through the construction of basic and intermediate SystemBuild block diagrams and BetterState charts. If you wish to skip the first four sections of the tutorial, the current tutorial catalog can be loaded into the CatalogBrowser by executing the following command from the Xmath Commands window.

```
load "$SYSBLD\examples\gs_tutorial\vibe1.cat";
```

Suppose that the space of the damped spring model is divided into two regions with different damping constants. Also, suppose that a barrier is added from which the mass rebounds. The resulting model includes events that produce discontinuous behaviors. To effectively simulate events, you need to detect them, and then modify affected values in the model. Modeling events often presents an additional challenge: *algebraic loops* must be avoided, or properly managed.

An algebraic loop is created when a signal path in a block diagram forms a loop, without passing through the input of an Integrator block. Look again at Figure 4-30. Note that each of the two signal loops pass through the **force** input to the **Newton** SuperBlock, which is subsequently routed through the input of both integrators. The standard processing algorithms of continuous SuperBlocks can not handle an algebraic loop unless its signal is controlled by one of several special SystemBuild features. In this tutorial, the use of a BetterState block creates an *asynchronous subsystem* that controls signal flow through algebraic loops.

To simplify implementation, the physical system you model in this section:

- sets a damping region boundary such that the region above the boundary is damped according to a damping constant; the region below the boundary is undamped (damping constant = 0).
- sets a barrier below the damping region boundary (in the undamped region).
- assumes a rebound *restitution factor* = 1.

To implement this enhancement to the damped spring model:

- Modify the **Newton** SuperBlock so that its integrators are resettable, give it more inputs for the signals needed to do the resets, and add acceleration to the outputs, so that it can be displayed with simulation results.
- Modify the damping loop so that the damping force can be switched on and off; add blocks that set the damping region boundary and barrier positions.
- Create a new SuperBlock for controlling events. It detects boundary crossings and barrier impacts, and uses ZeroCrossing blocks to generate event signal inputs to a BetterStateChart block.
- Develop a BetterState chart that maintains the damping state, processes boundary crossings and barrier impact events, and has three outputs:
  - a signal to switch the damping force.
  - a value for resetting velocity.
  - an integrator reset trigger.
- Make final connections in the block diagram.

### *Resettable Integrator*

In this section you learn how to modify the block properties of an Integrator block to make it resettable. A resettable integrator has two additional input signals: one containing the reset value, and one to trigger resets. These additional inputs to integrator blocks do not break algebraic loops, as does the input signal to be integrated.

To modify the **Newton** SuperBlock:

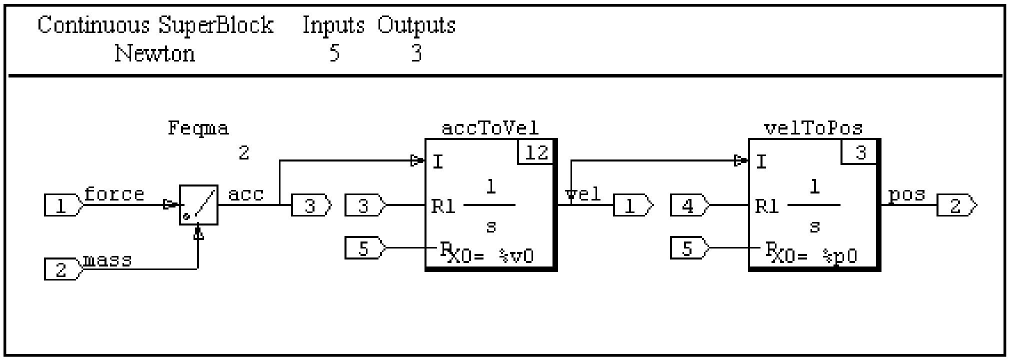
1. Access the block diagram of the **Newton** SuperBlock.
   a. Make the Editor the active window and update the **vibe** SuperBlock by selecting File→Update.
   b. Double click on the **Newton** SuperBlock block. The Editor now displays the block diagram of the **Newton** SuperBlock.

2. Edit properties of the accToVel Integrator block.
   a. In the Parameter table, locate the drop-down combo box in the Value column and the Resettable row. Change its value to **Double Edge**.

      A *double edge* reset is triggered by a signal that changes from **<=0** to **>0**, or from **>0** to **<=0**. The trigger in this model resets the integrators by switching between **0** and **1**.
   b. Click the Input tab and name the inputs: **acc**, **velReset**, and **trigger**.
3. Change the velToPos Integrator block reset to Double Edge.

   Name its inputs: **vel**, **posReset**, and **trigger**.
4. Edit the **Newton** SuperBlock Properties dialog:
   a. Open the dialog by selecting File→SuperBlock Properties....
   b. Change the number of inputs to **5** and outputs to **3**.
   c. Click OK to accept current values and close the dialog.
5. Complete connections, and make minor position adjustments so that your **Newton** block diagram resembles Figure 4-33.

Figure 4-33 **Newton with Resettable Integrators**



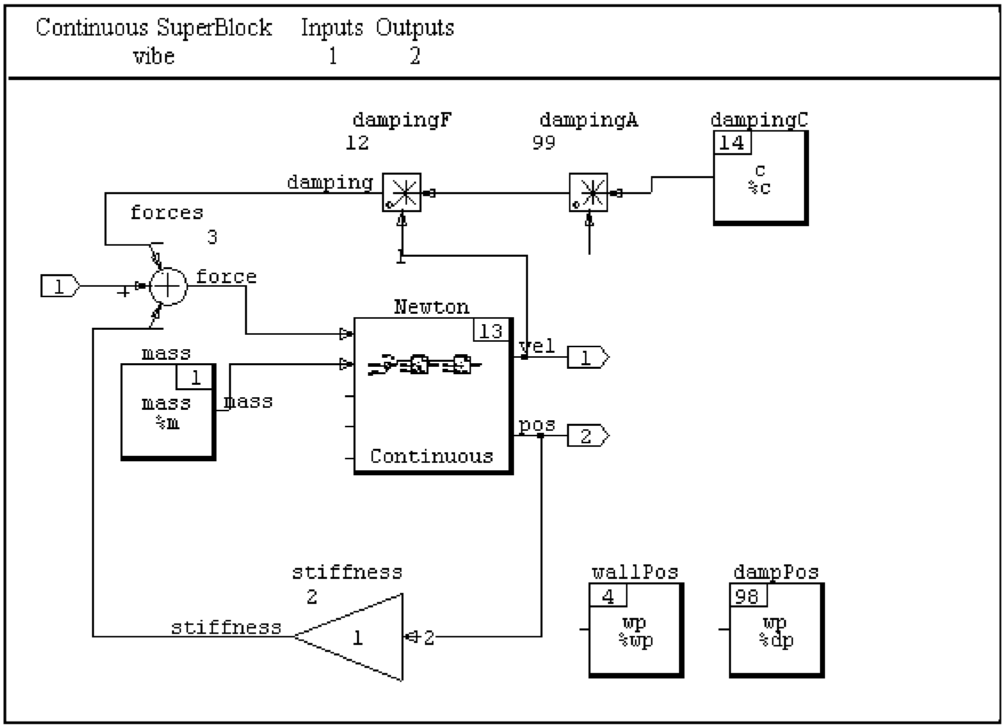
### ***Signal Switch***

In this section, you implement a simple signal switch that enables a model to turn a signal on and off.

To modify the damping loop, and add blocks for damping boundary and barrier position:

1. In the SuperBlock editor, return to the **vibe** block diagram by selecting View→Parent→vibe, followed by File→Update.
2. Delete the **damping** Gain block:
   a. Select the block by clicking it with MB1.
   b. Select Edit→Delete.
3. Add blocks to **vibe**:
   a. Add an ElementProduct block from the Algebraic palette, and a Constant block from the Matrix Equations palette.
   b. Flip and duplicate the ElementProduct block.
   c. Flip, reduce twice, and make 2 duplicates of the Constant block.
   d. Position the **vibe** blocks as shown in Figure 4-34.
4. Edit block properties:
   a. Name the left ElementProduct block **dampingF,** name its inputs **dampingC** and **velocity**, label its output **damping**, and enable its Show Outputs Labels check box.
   b. Name the right ElementProduct block **dampingA,** name its inputs **dampingC** and **switch**, and label its output **damping**.
   c. Name the upper right Constant block **dampingC,** change its ConstantName to **c**, give it a % variable **c**, and label its output **damping**.
   d. Name the lower left Constant block **wallPos,** change its ConstantName to **wp** give it a % variable **wp**, and label its output **wallPos**.
   e. Name the upper right Constant block **dampPos,** change its ConstantName to **dp** give it a % variable **dp**, and label its output **dampPos**.
5. Make block connections for the damping loop as shown in Figure 4-34. More connections are added later.

Figure 4-34 **vibe Modifications**



Next you create a Superblock to control events. Instead of encapsulating blocks already in place, you add a new SuperBlock block to **vibe**, and then construct its block diagram.

## *Event Controller*

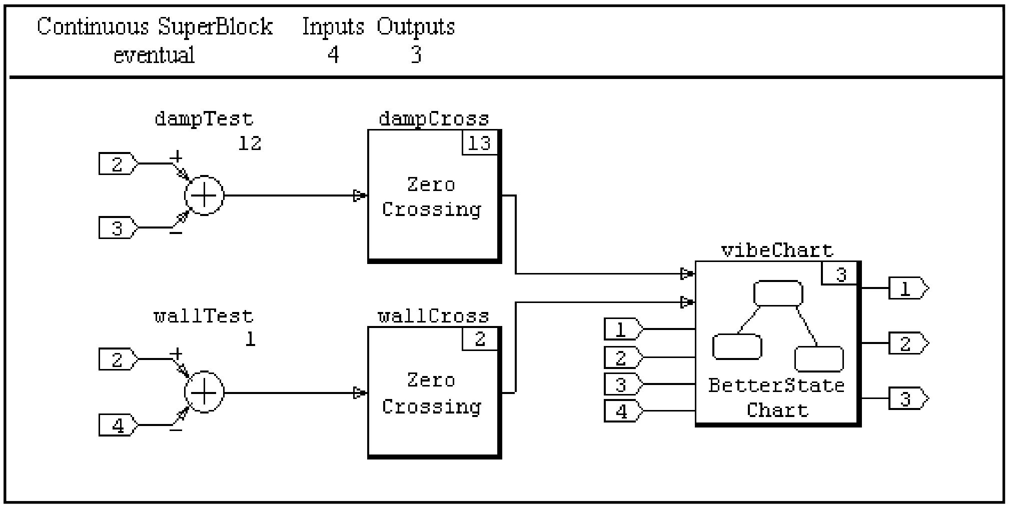
In this section, you'll create an event controller for **vibe**. An event controller recognizes that an event has happened, and causes actions to occur as a result. Actions can be based on state information stored by the controller, and can include changes in output signal values, and internal state transitions.

To create a new SuperBlock for controlling events:

1. Add a SuperBlock block to **vibe**:
   a. Drag and drop a SuperBlock block from the SuperBlocks palette to the open area on the right side of the **vibe** block diagram. Make it the same size as the **Newton** SuperBlock.
   b. Name the block **eventual** Give it 4 inputs and 3 outputs and change its Icon Type to **User**.
2. Navigate to the **eventual** block diagram and add blocks:
   a. Double click the new SuperBlock block. The SuperBlock editor now displays an empty block diagram.
   b. Drag and drop a Summer block from the Algebraic palette, a ZeroCrossing block from the User Programmed palette, and a BetterStateChart block from the BetterState palette.
   c. Duplicate the Summer and ZeroCrossing blocks, and enlarge the BetterStateChart block twice.
   d. Position the blocks as shown in Figure 4-35.
3. Edit block properties:
   a. Name the upper Summer block **dampTest**, name its inputs **pos** and **dampPos**, label its output **dampTest**, and change its Icon Type to **Simple**.
   b. Name the lower Summer block **wallTest**, name its inputs **pos** and **wallPos**, label its output **wallTest**, and change its Icon Type to **Simple**.
   c. Name the ZeroCrossing blocks **dampCross** and **wallCross**, and use those names to label their outputs.
   d. Name the BetterStateChart block **vibeChart**.

You make connections in the eventual SuperBlock after developing the BetterState chart.

Figure 4-35 **eventual SuperBlock**



### *BetterStateChart Block*

A BetterStateChart block performs a central role in an event controller. It maintains current state information, and allows specification of the actions taken as events are processed. For more information about the BetterStateChart block, see the *BetterState User's Guide*.

To develop the **vibeChart** BetterState chart:

1. Display the BetterState Statechart window for **vibeChart**.
2. Verify settings in the Chart Properties dialog.
3. Specify events and variables in the Data Dictionary dialog.
4. Add states to **vibeChart**.
5. Edit state properties.
6. Add transitions to **vibeChart**.
7. Edit transition properties.

## *BetterState Statechart*

To display the BetterState Statechart window for **vibeChart:**

Double click on the BetterStateChart block.The BetterState process is activated and the BetterState Statechart window is displayed. The chart area is empty. See Figure 4-36.

**NOTE:** There may be a delay as the BetterState process is loaded.

**NOTE:** State chart appearance can be controlled through settings available through Visual Settings dialogs. The settings for the chart displayed in this tutorial are selected for visual clarity in b/w display. The chart you construct will differ in color and text styles.

Figure 4-36 **The BetterState Statechart Window**

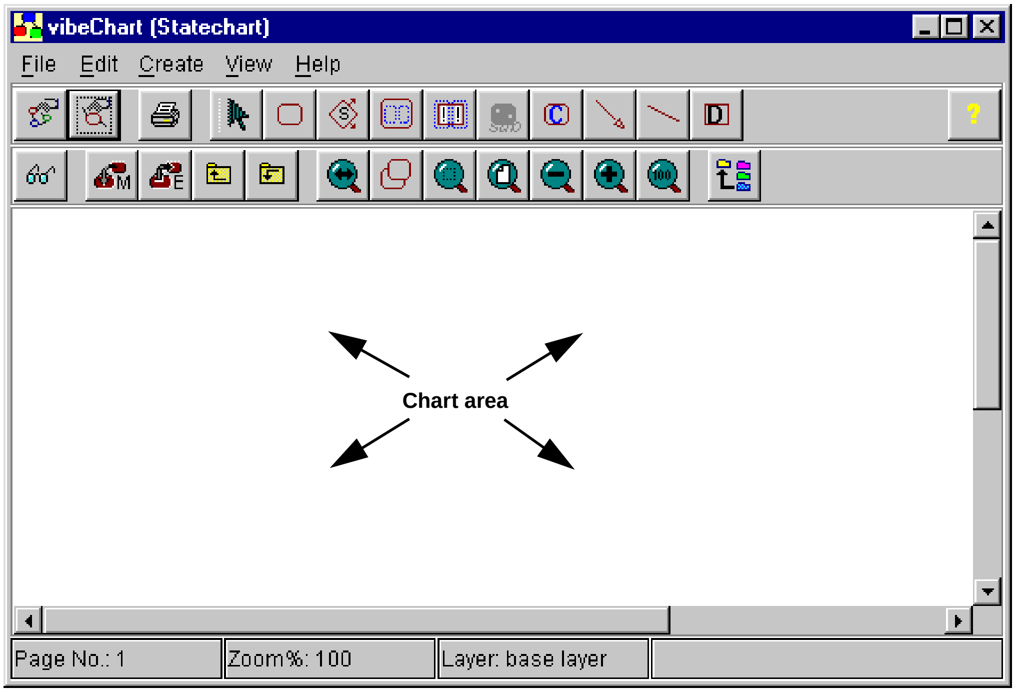


Figure 4-37 **BetterState Chart Properties dialog**

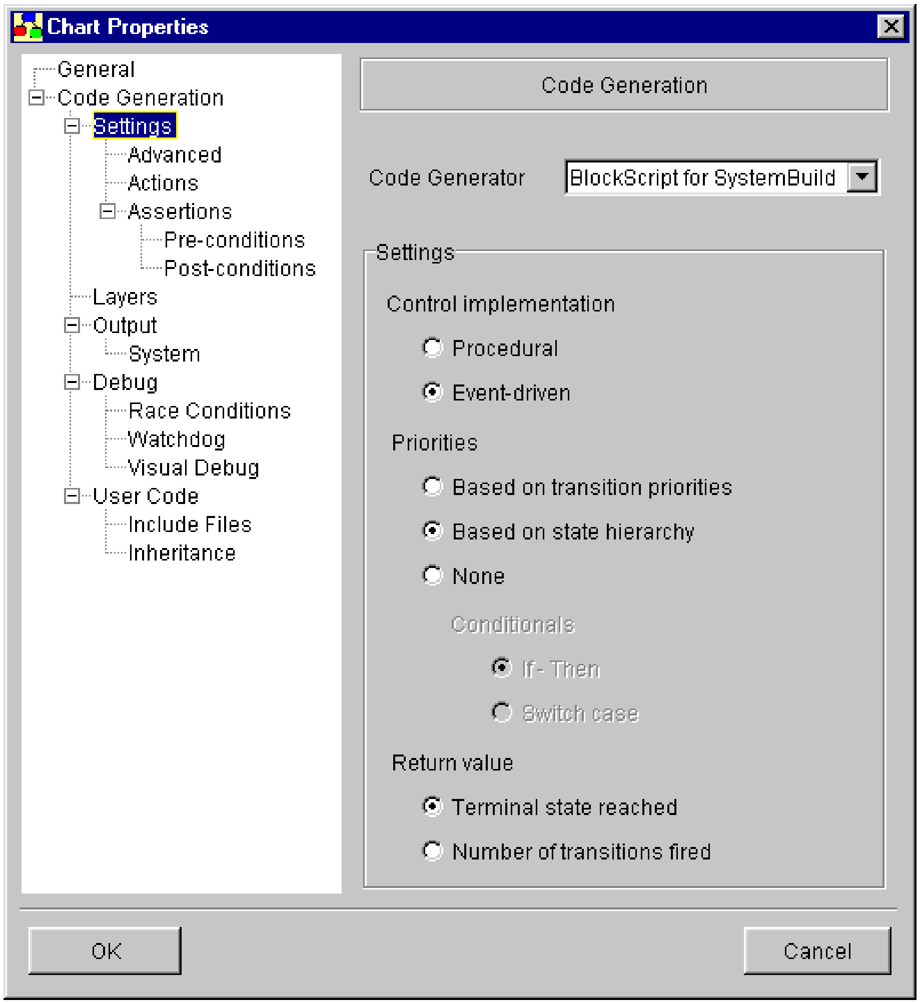


Chart Properties

- General
- Code Generation
  - Settings
    - Advanced
    - Actions
    - Assertions
      - Pre-conditions
      - Post-conditions
  - Layers
  - Output
    - System
  - Debug
    - Race Conditions
    - Watchdog
    - Visual Debug
  - User Code
    - Include Files
    - Inheritance

Code Generation

Code Generator: BlockScript for SystemBuild

Settings

Control implementation

- ○ Procedural
- ● Event-driven

Priorities

- ○ Based on transition priorities
- ● Based on state hierarchy
- ○ None

Conditionals

- ● If- Then
- ○ Switch case

Return value

- ● Terminal state reached
- ○ Number of transitions fired

### *Chart Properties*

To verify settings in the Chart Properties dialog:

1. From the Statechart window, select File→Chart Properties....

   The Chart Properties dialog is displayed. (Figure 4-37 shows the Chart Properties dialog as it appears during Step 2 below).

2. Click the Code Generation/Settings node and verify:
   a. The Code Generator combo box field contains **BlockScript for SystemBuild**.
   b. In the Control implementation group, the Event-driven radio button is enabled.

### *Data Dictionary*

To specify events and variables in the Data Dictionary dialog:

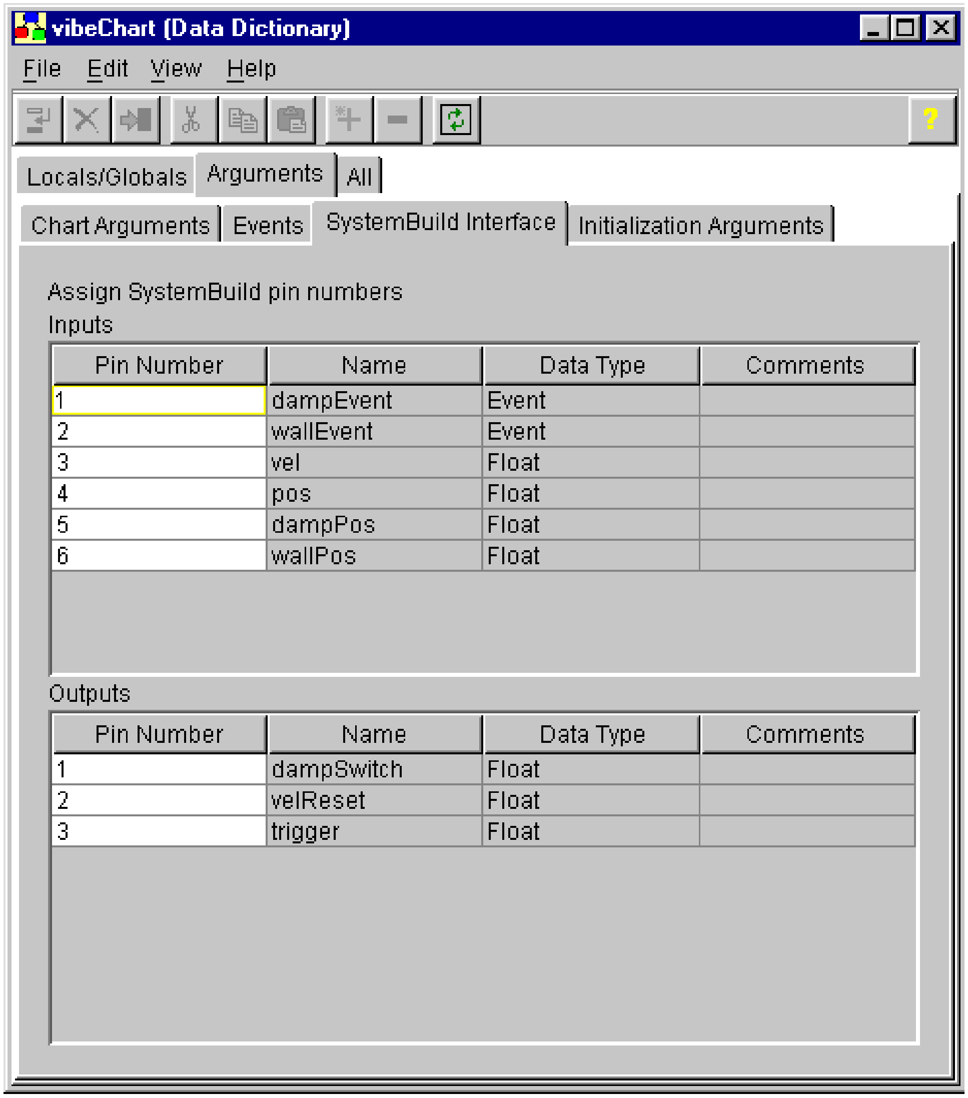
1. From the Statechart window, select File→Data Dictionary....

   The Data Dictionary dialog is displayed. (Figure 4-38 shows the Data Dictionary dialog as it appears during Step 5 below).
2. To enter event names in the Data Dictionary dialog:
   a. Select the Arguments tab. In the Arguments pane, select the Events tab.
   b. In the Define event inputs and outputs editable combo box field, type **dampEvent** and press the Enter key.
   c. Double click on the entry **dampEvent** to select it, type **wallEvent**, and press the Enter key.
   d. Click the Define event inputs and outputs combo box arrow to verify that the list now contains both **dampEvent** and **wallEvent**.
3. To enter input variable names in the Data Dictionary dialog:
   a. In the Arguments pane, select the Chart Arguments tab.
   b. In the Inputs table, double click in the Name field of row 1. Type **vel** and press the Enter key.
   c. Enter the input variable names **pos**, **dampPos**, and **wallPos** in the Name fields of rows 2 through 4.
4. With the Chart Arguments tab still selected, enter the output variable names **dampSwitch**, **velReset**, and **trigger** in the Outputs table Name fields of rows 1 through 3.
5. Verify all entries in the Data Dictionary dialog.

   In the Arguments pane, select the SystemBuild Interface tab. The information displayed should agree with that shown in Figure 4-38.

   Select File→Close to close the dialog.

Figure 4-38 **Data Dictionary dialog**



Assign SystemBuild pin numbers

Inputs

| Pin Number | Name | Data Type | Comments |
|---|---|---|---|
| 1 | dampEvent | Event | |
| 2 | wallEvent | Event | |
| 3 | vel | Float | |
| 4 | pos | Float | |
| 5 | dampPos | Float | |
| 6 | wallPos | Float | |

Outputs

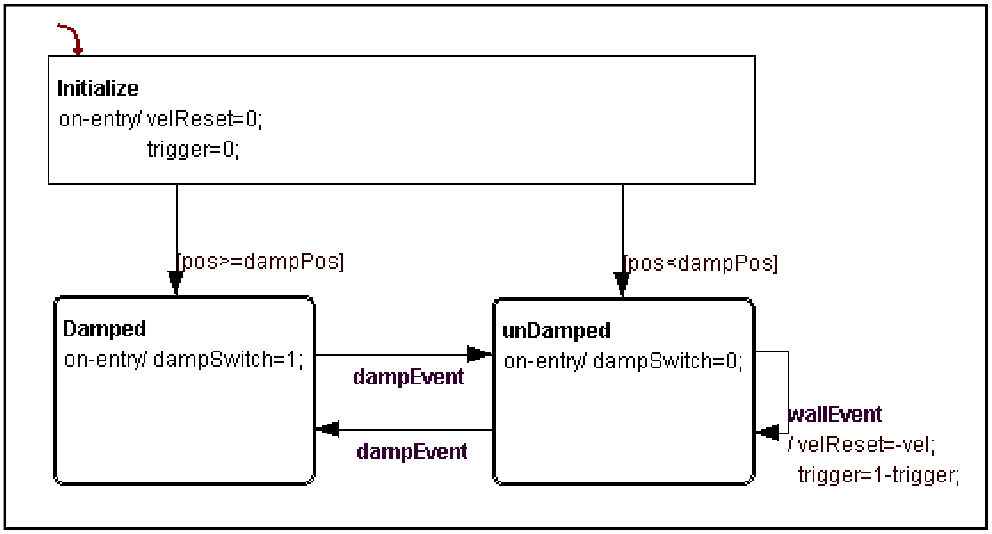
| Pin Number | Name | Data Type | Comments |
|---|---|---|---|
| 1 | dampSwitch | Float | |
| 2 | velReset | Float | |
| 3 | trigger | Float | |

### *States*

To add states to **vibeChart**:

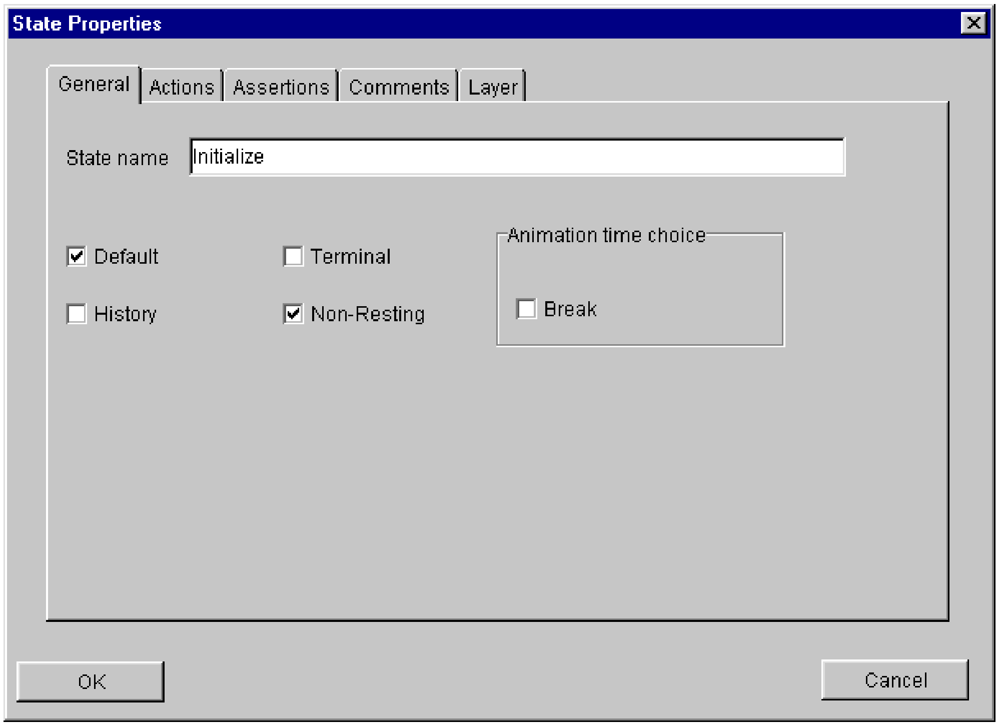
1. On the Statechart window, select Create→State. Click in the chart area. Drag and release to create a rectangle. Create two more rectangles in the chart area.

Figure 4-39 **vibeChart**



2. Resize and position your state rectangles to resemble those in Figure 4-39:
   a. Select Create→Select Mode.
   b. Select a rectangle by clicking in it.
   c. Resize by dragging the resize handles of a selected rectangle.
   d. Position by clicking inside a selected rectangle and dragging.

Figure 4-40 **State Properties dialog**


State Properties

General | Actions | Assertions | Comments | Layer

State name: Initialize

[✓] Default
[ ] Terminal
[ ] History
[✓] Non-Resting

Animation time choice
[ ] Break

OK
Cancel

### *State Properties*

To edit state properties:

1. On the Statechart window, select Create→Select Mode and then double click inside the top state rectangle. The State Properties dialog for that state is displayed. (Figure 4-40 shows the State Properties dialog as it appears during Step a below).

   a. Name the state **Initialize**, and enable the Default and Non-Resting check boxes.

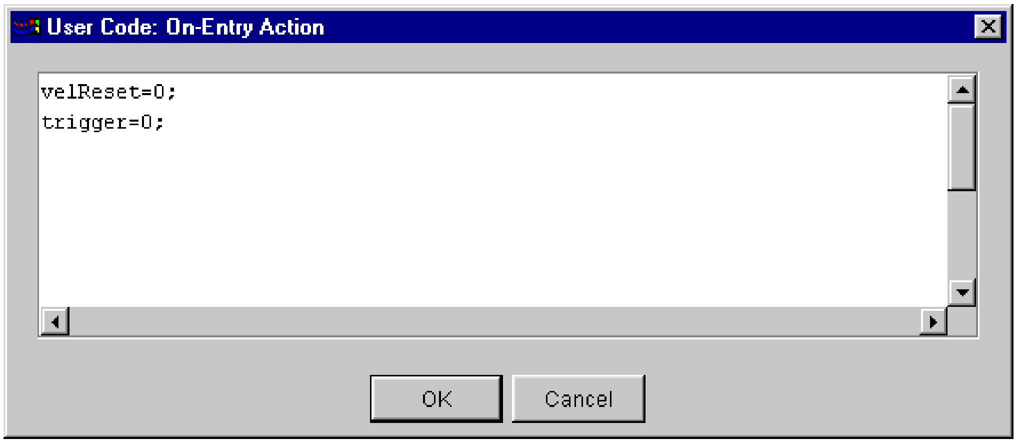
      The *default* state is entered at the start of a simulation. A simulation does not remain in a *non-resting* state. **vibeChart**'s **Initialize** state is entered only during initialization of simulations. At that time, it assigns values to two of the output variables, and then determines the correct state to assume based on initial values of its input variables.

b. Click the Actions tab. Verify that the User Code radio button is selected for Edit On-Entry Action. Click the Edit On-Entry Action button.

   The User Code dialog for the On-Entry Action of the **Initialize** state is displayed. Enter initializations for the variables velReset and trigger as shown in Figure 4-41.

c. Click the OK button to verify that the correct code is entered, and exit the User Code dialog. Click the OK button of the State Properties dialog to return to the Statechart window.

Figure 4-41 **User Code dialog**



```
velReset=0;
trigger=0;
```

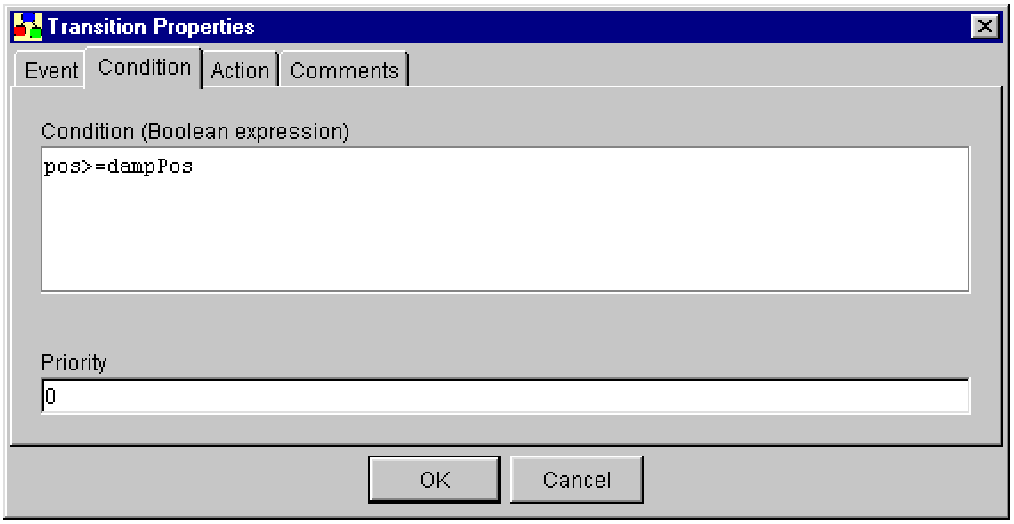
2. Double click on the lower left state to display its State Properties dialog.
   a. Name the state **Damped**. (Do not enable any of the check boxes.)
   b. Navigate to the User Code dialog for the On-Entry Action of the **Damped** state. Enter the code line:
      **dampSwitch=1;**
   c. Return to the Statechart window.
3. Name the remaining state **unDamped**. Give it an On-Entry Action:
   **dampSwitch=0;**

## *Transitions*

To add transitions to vibeChart:

1. On the Statechart window, select Create→Transition. Click inside the **Initialize** state, drag into the **Damped** state, and release. A transition arrow now connects the **Initialize** rectangle to the **Damped** rectangle.
2. Add additional transitions:
   **Initialize → unDamped**
   **Damped → unDamped**
   **unDamped → Damped.**
3. Create a transition from **unDamped** back to itself:
   a. Click inside the **unDamped** state.
   b. Drag into empty area to the right of **unDamped.**
   c. Drag back into the **unDamped** state and release.
4. Position your transition arrows to resemble those in Figure 4-42:
   a. Select Create→Select Mode.
   b. Select a transition arrow by clicking on it.
   c. Position by dragging the handles of a selected transition arrow.

Figure 4-42 **Transition Properties dialog**



Condition (Boolean expression)

pos>=dampPos

Priority

0

### *Transition Properties*

To edit transition properties:

1. On the Statechart window, select Create→Select Mode and then double click on the transition arrow from **Initialize** to **Damped.** The Transition Properties dialog for that transition is displayed. (Figure 4-42 shows the Transition Properties dialog as it appears during Step a below).

   a. Verify that the Condition tab is selected and enter the transition condition in the Condition (Boolean expression) text entry box as shown in Figure 4-42.

   b. Click the OK button to accept the edited properties and exit the Transition Properties dialog.

2. Give the **Initialize** to **Damped** transition the Condition (Boolean expression): **pos<dampPos**

3. Edit the **Damped** to **unDamped** transition properties:

   a. Verify that the Event tab is selected and set the Event name combo box field to **dampEvent**.

   b. Click OK to exit the Transition Properties dialog.

4. Set the **unDamped** to **Damped** transition Event name to **dampEvent.**

5. Edit the **unDamped** to **unDamped** transition properties:

   a. Set the Event name combo box field to **wallEvent**.

   b. Select the Action tab. Verify that the User Code radio button is selected and click the Edit On-Entry Action button.

      In the User Code dialog, enter the code lines:
      **velReset=-vel;**
      **trigger=1-trigger;**

      Click OK to exit the User Code dialog.

   c. Click OK to exit the Transition Properties dialog.

**NOTE:** Recall that a simplifying assumption was made concerning the position of the barrier wall: the wall is located within the undamped region. Therefore, it is not necessary for the damped state to handle a wall event.

If an event triggers a state chart process, and the transitions of the current state do not reference that event, no state transition occurs. However, if the current state contains a During Action, the code for that action is executed.

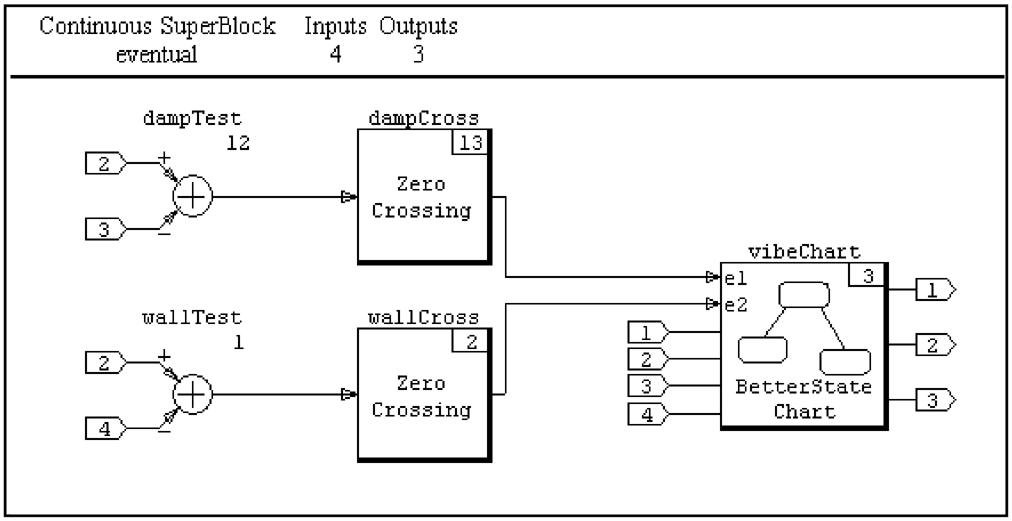
### Return to SystemBuild for Final Connections

Your state chart should now look like the one in Figure 4-39. Recall that visual properties have been modified for tutorial Statechart figures.

To make final connections in the **eventual** and **vibe** block diagrams:

1. Select File→Close Window to return to the **eventual** block diagram in the SystemBuild SuperBlock editor.
2. In the SuperBlock editor, select File→Update.
3. Make connections in the **eventual** block diagram as shown in Figure 4-43.

Figure 4-43 **The eventual Block Diagram**

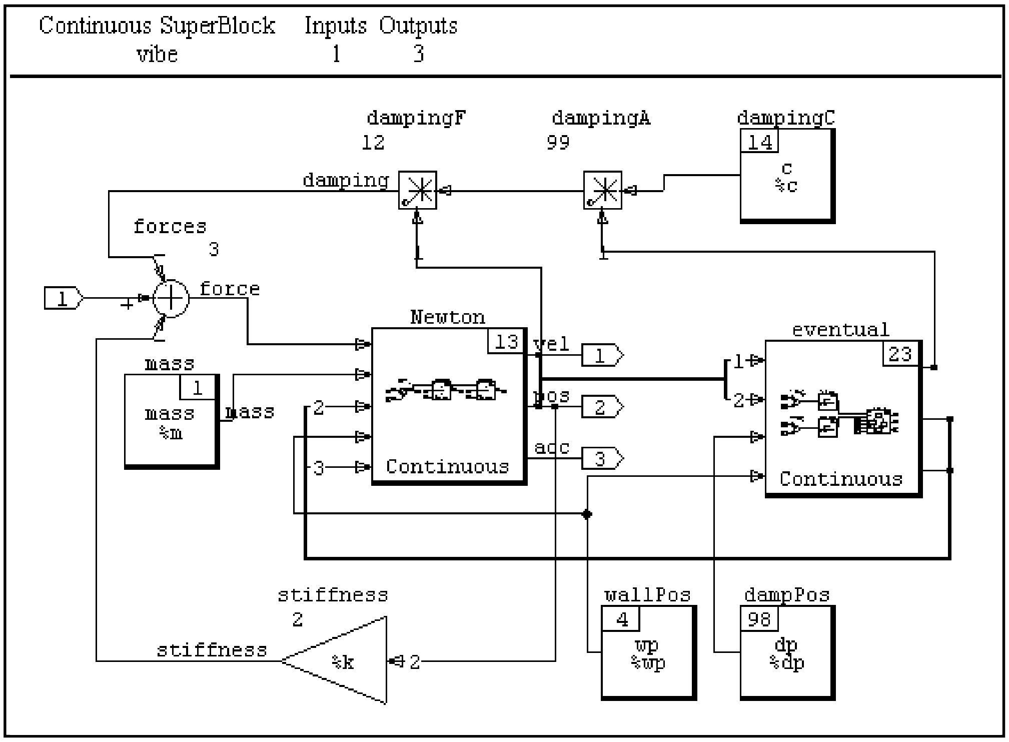


4. Return to the **vibe** block diagram by selecting View→Parent→vibe, followed by File→Update.
   a. Change the number of **vibe** external outputs to **3**.
   b. Connect **Newton** output 3 to **vibe** output 3.
   c. Make additional **vibe** connections as shown in Figure 4-44.

Before you test the event enhanced model, take a minute to save your work.

In the first simulation, you test barrier events only. Gravity is used to drop the mass from an initial position beneath the damping boundary, but above the barrier. Spring stiffness and damping constant are set to zero.

Figure 4-44 **The vibe Block Diagram.**



**NOTE:** The current tutorial block diagrams and state chart can be loaded into the CatalogBrowser by executing the following command from the Xmath Commands window.

```
load "$SYSBLD\examples\gs_tutorial\vibe2.cat";
```

**vibe2** contains a solution of the intermediate spring-mass damper model with event modeling using the BetterState block.

To simulate barrier events with **vibe**:

1. In the Xmath command area, enter the following values:

```
t = [0:0.001:5]';
u = -9.8 * ones(t);
m = 1;
v0 = 0;
p0 = 7;
c=0;
k=0;
dp=10;
wp=0;
```

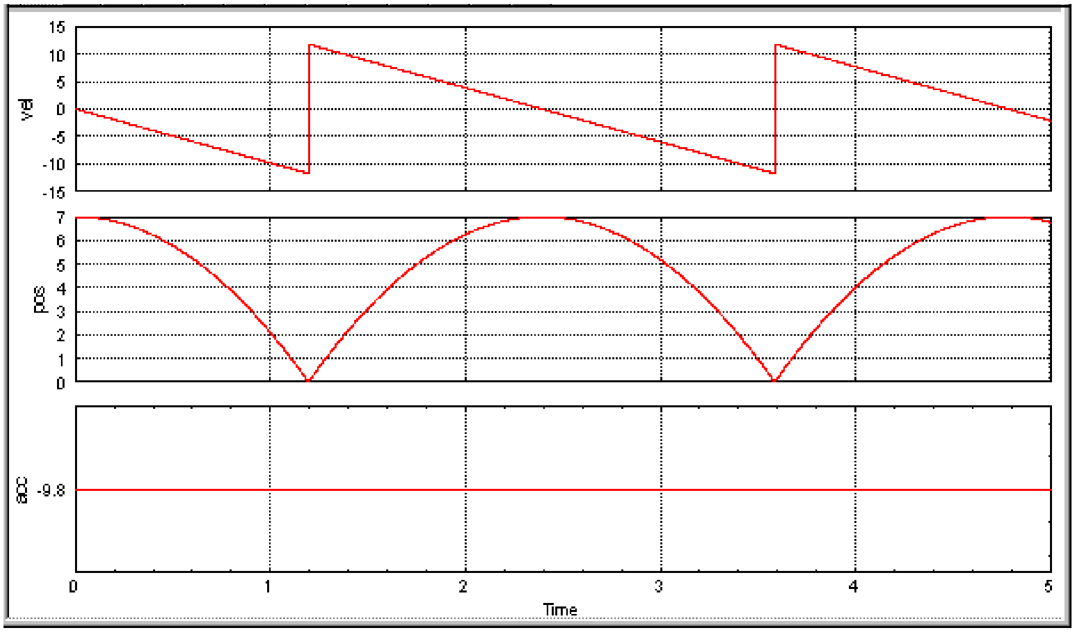
2. Execute the simulation:

```
y = sim("vibe",t,u,{graph});
```

The simulation proceeds as before, but there is a delay as the BetterStateChart block code is compiled and linked. Also, the simulation output log shows that two zero-crossings occurred.

The plot results in Figure 4-45 verify that these zero-crossings correspond to rebounds from the barrier.

Figure 4-45 **vibe Barrier Events**



In the final simulation, you include damping boundary events. The external input force is set to zero, but the spring produces a force when the mass is displaced from the spring equilibrium point. The damping boundary is set between the initial position and the equilibrium point (zero). The barrier is set on the opposite side of the equilibrium point.

To simulate damping boundary and barrier events with **vibe**:

1. In the Xmath command area, enter the following values:

```
t = [0:0.001:5]';
u = zeros(t);
m = 1;
v0 = 0;
p0 = 7;
c=5;
k=20;
dp=2;
wp=-2;
```

2. Execute the simulation:

```
y = sim("vibe",t,u,{graph});
```

The simulation log shows many more zero-crossings. The plot results in Figure 4-46 verify that some are form barrier events, and some are from damping boundary events. Note that the velocity curve is discontinuous at barrier events, and the acceleration curve is discontinuous at damping boundary events.

## *Final Exercise*

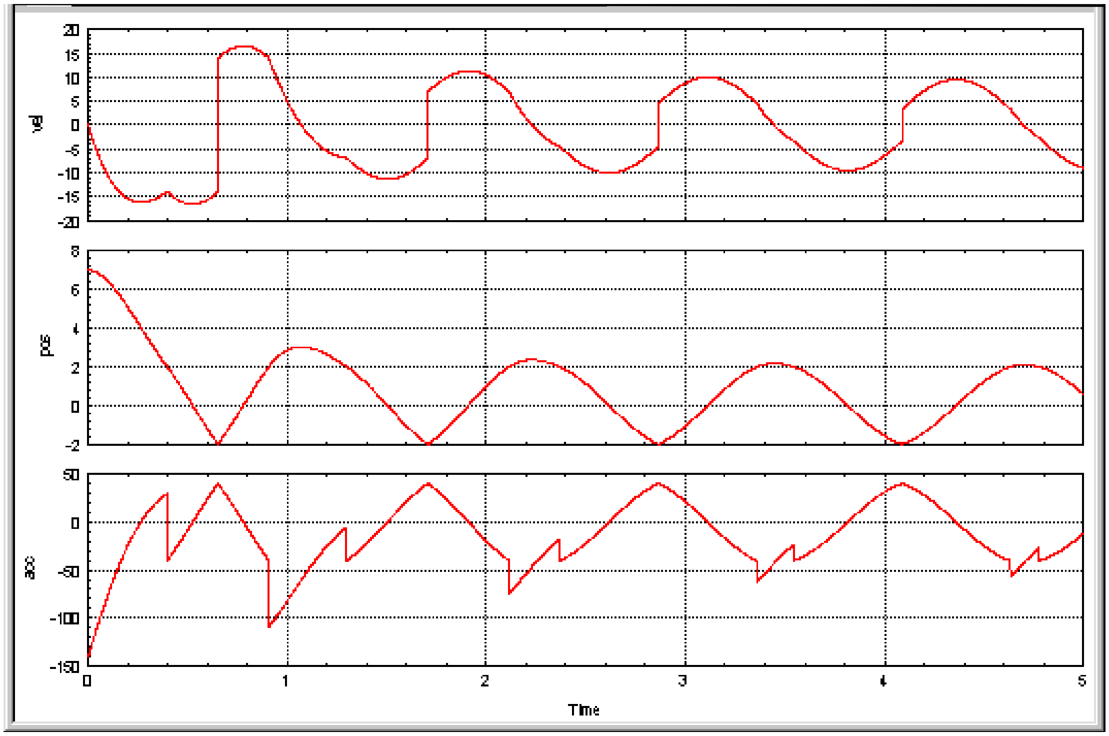
Recall that several simplifying assumptions were made in modeling damping region and barrier events. Test your understanding of SystemBuild and BetterState, as well as your design skills, by adding the following enhancements to the current model:

- Damping constants can be specified for both damping regions.
- Barriers can be placed both above and below the initial position of the mass.
- Restitution factors can be specified for both barriers.

**NOTE:** A solution to the final exercise can be loaded into the CatalogBrowser by executing the following command from the Xmath Commands window.

```
load "$SYSBLD\examples\gs_tutorial\vibe3.cat";
```

Figure 4-46 **vibe Barrier and Damping Boundary Events**



# 5 AutoCode

AutoCode software lets you generate ANSI C or Ada code automatically from SystemBuild models.

You can generate code from the Catalog Browser in SystemBuild or use the **autocode** Xmath command. The generated code represents a complete implementation of the model and can be targeted to run on computers or on an actual controller. The default target is a stand-alone simulation that you can execute on your computer; you can load the results of the simulation back into Xmath for analysis.

## 5.1 Generating Non-Customized Code

To generate code for the sample Discrete Cruise System model, follow the steps below. on your terminal.

To generate code for the Discrete Cruise System SystemBuild model:

1. If Xmath is not currently running, start Xmath as described in *3.2.3 Starting Xmath*, p.34.
2. Verify that you have write permission for the current directory, and that it is where you want to save your code. If not, enter the command below from the Xmath command window, substituting your directory name:

   ```
   set directory ="write_enabled_directory"
   ```

3. From the Xmath command line, type the following command to load the model:

   ```
   load "$SYSBLD\demo\cruise_demo\cruise_d.cat";
   ```

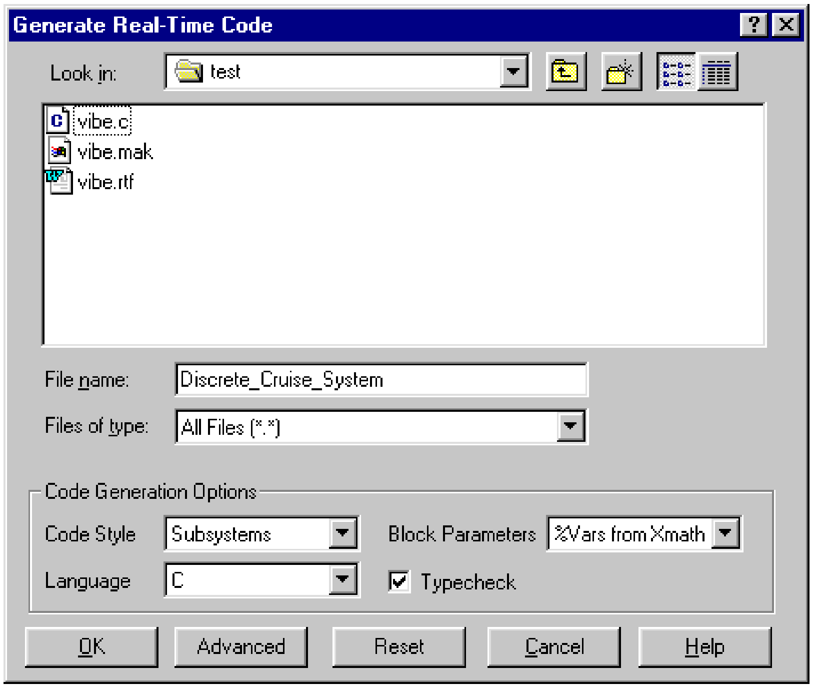
**NOTE:** Environment variables are only recognized on the Xmath command line. For other loading methods, you must know the full pathname of the SystemBuild directory.

4. From the SystemBuild Catalog Browser, select the Discrete Cruise System SuperBlock.

**NOTE:** You must generate code from a top level SuperBlock.

5. From the Catalog Browser, select Tools→AutoCode to bring up the Generate Real-Time Code dialog (see Figure 5-1).
6. Enter a name in the File name field, or accept the default, **Discrete_Cruise_System**.

Figure 5-1 **Generate Real-Time Code Dialog**



File name: Discrete_Cruise_System
Files of type: All Files (*.*)
Code Style: Subsystems
Block Parameters: %Vars from Xmath
Language: C
Typecheck [✓]

7. Click OK to start the code generation process.
8. Activate the Xmath Commands window to monitor the progress of the code generation.
9. Once the code generation is complete, look for a statement similar to the following in the Xmath log area:

```
Output generated in your_directory\Discrete_Cruise_System.c.

Code generation complete.
```

10. (Optional) Display the output file in the Xmath Output window by entering a command similar to the following in the Xmath Commands window:

```
oscmd ("type your_directory\Discrete_Cruise_System.c")
```

## 5.2 Generating Customized Code

To customize your AutoCode output, click Advanced on the Generate Real-Time Code dialog; this brings up the Advanced dialog (see Figure 5-2).

You can use the Advanced dialog from the AutoCode Code Generation dialog or use keywords with the **autocode** Xmath command to customize the generated code as follows:

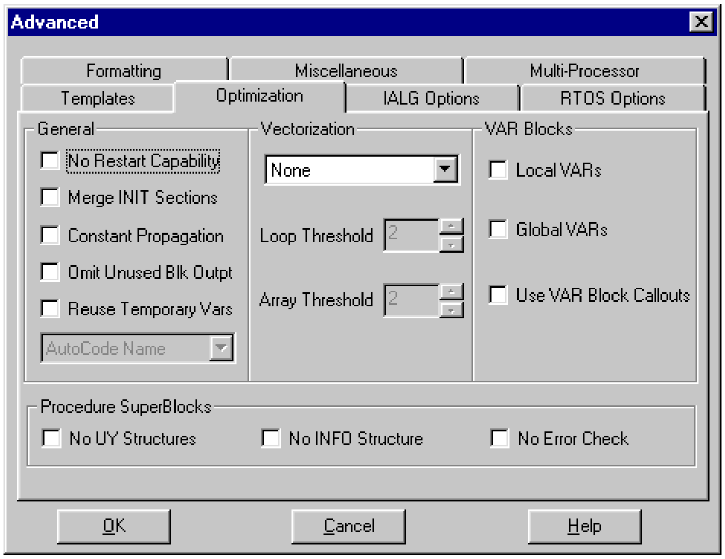
- Specifying a template file on the Templates tab allows you to control the formatting of the output of AutoCode to meet a variety of software needs; you can modify the overall architecture of generated code, customize the scheduler, modify data structures and external I/O calls, add user code, and so forth. Using the Template Programming Language (TPL), you can tailor any part of the code except the hierarchy logic and the elementary blocks. Numerous templates are available, including one to customize the generated code for the pSOSystem real-time operating system. For more information on templates, see the *Template Programming Language User's Guide*.
- Formatting options (Formatting tab) let you set maximums, such as the number of significant digits, the length of variable names, and columns per row. From here, you can also specify indentation between levels, as well as set a number of other parameters.

- The IALG (Integration Algorithms) Options tab lets you select an integration algorithm such as Euler or Runge Kutta.
- The Multi-Processor tab lets you specify a processor, startup, background, interrupt, skew, priority, or map file.
- The Optimization tab lets you make general, vectorization, and VAR block settings that affect code size and efficiency (see the *Autocode Reference* for details).
- The Miscellaneous tab lets you select an options file, the type of scheduler, output scope control, and various other settings.
- The RTOS (real-time operating system) Options tab lets you specify a configuration file and set additional options.

Once you have customized your settings, you click OK in the Advanced dialog; then you generate code by clicking OK in the Generate Real-Time Code dialog.

For information about compiling, executing, and using the generated code, see the *AutoCode User's Guide.* For information about autocode keywords, see the topic *AutoCode* in the MATRIX$_X$ online Help.

Figure 5-2 **Advanced Dialog**


Advanced

Formatting | Miscellaneous | Multi-Processor
Templates | Optimization | IALG Options | RTOS Options

General
- [ ] No Restart Capability
- [ ] Merge INIT Sections
- [ ] Constant Propagation
- [ ] Omit Unused Blk Outpt
- [ ] Reuse Temporary Vars

AutoCode Name

Vectorization

None

Loop Threshold 2

Array Threshold 2

VAR Blocks
- [ ] Local VARs
- [ ] Global VARs
- [ ] Use VAR Block Callouts

Procedure SuperBlocks
- [ ] No UY Structures
- [ ] No INFO Structure
- [ ] No Error Check

OK | Cancel | Help

# 6 DocumentIt

The DocumentIt software generates block-level documentation for SystemBuild models. The DocumentIt software extracts the parameters of the SuperBlocks and elementary blocks in your model and any comments you have entered for each block; it then formats the documentation according to guidelines you define. You can generate documentation from the Xmath command area or from the SystemBuild Catalog Browser. You can invoke controls as arguments from the command area or make choices in a user dialog.

This chapter provides an introduction to using DocumentIt. For a complete description, see the *DocumentIt User's Guide.*

## 6.1 Generating Non-Customized Documentation

To generate documentation for the sample Discrete Cruise System model, follow the steps below. We assume that you have Xmath running on your terminal.

1. Make sure you are in a directory where you have write permission for saving your code. If not, enter the command below from the Xmath command window, substituting your directory name:

   ```
   set directory = "your_directory"
   ```

2. From the Xmath command line, type the following command to load the model:

```
load "$SYSBLD\demo\cruise_demo\cruise_d.cat";
```

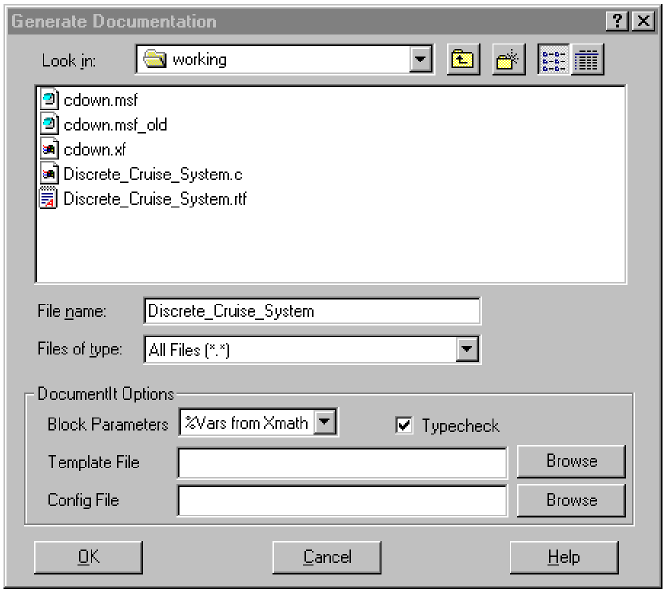
**NOTE:** Environment variables are recognized only in the Xmath command area. For loading with other methods, you must know the full pathname of the SystemBuild directory. See *3.2.1 Directories Defined by Environment Variables*, p.33 for additional information.)

3. From the SystemBuild Catalog Browser, select the Discrete Cruise System SuperBlock.

**NOTE:** You must generate documentation from a top level SuperBlock.

4. Select Tools→DocumentIt to bring up the Generate Documentation dialog (see Figure 6-1).

Figure 6-1 **Generate Documentation Dialog**



cdown.msf
cdown.msf_old
cdown.xf
Discrete_Cruise_System.c
Discrete_Cruise_System.rtf

File name: Discrete_Cruise_System
Files of type: All Files (*.*)

DocumentIt Options
Block Parameters: %Vars from Xmath
[✓] Typecheck
Template File:
Config File:

5. Choose a directory, and enter a name in the File name field or accept the default, **Discrete_Cruise_System**.

**NOTE:** You do not need to supply the extension. DocumentIt supplies the default, **.doc**, for you.

6. Click OK to start the document generation process.
7. Select the Xmath Commands window to monitor the progress of the documentation generation.
8. Once the document generation is complete, look for a statement similar to the following in the Xmath Log window:

```
Documentation generation complete.
Document generated and saved in file: Discrete_Cruise_System.doc.
```

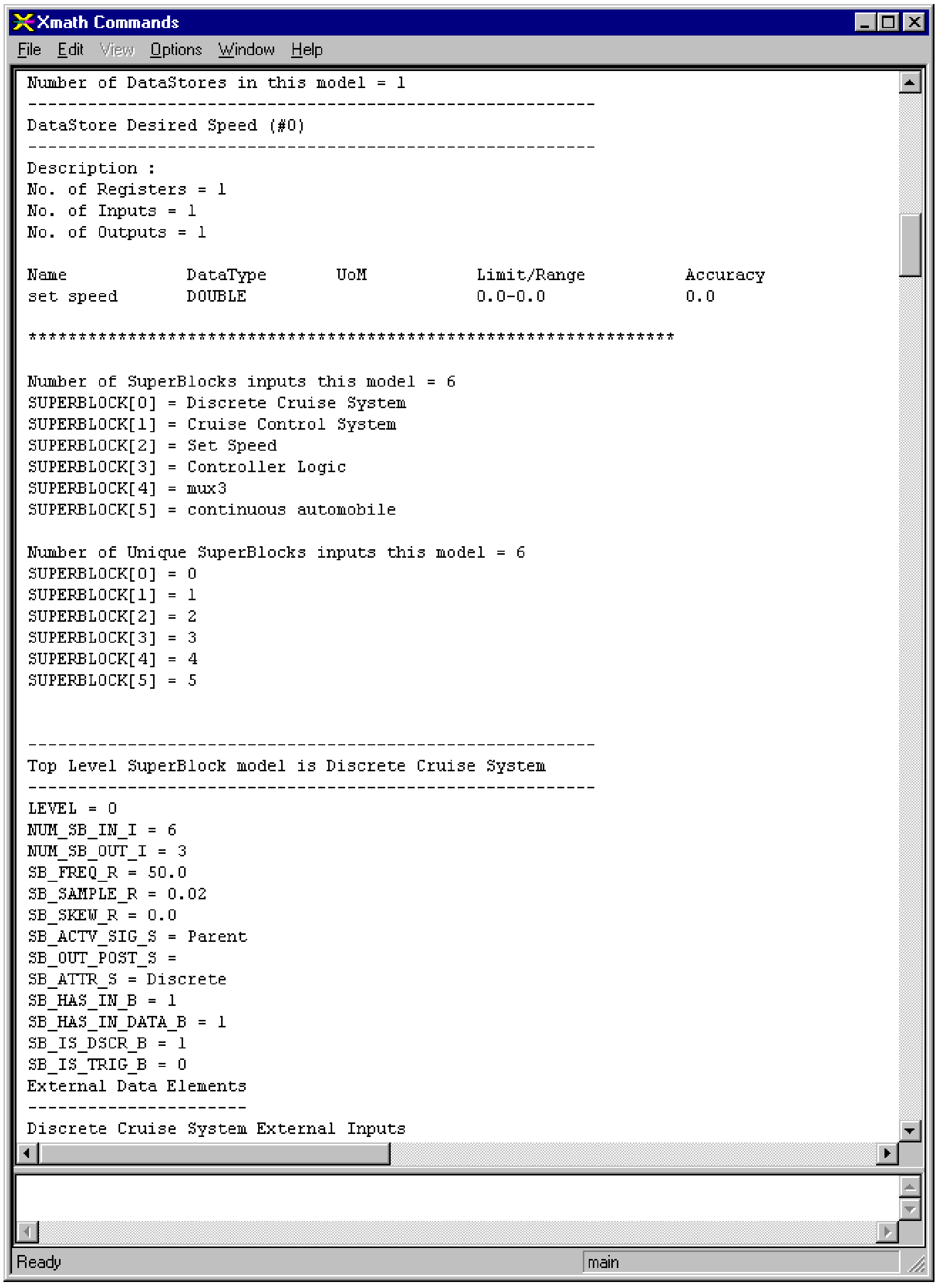
**NOTE:** The **.doc** file is in ASCII format. The current defaults also produce an**.rtf** file, which contains Microsoft Word markup commands.

9. (Optional) Display the output file in the Xmath Output window by entering a command similar to the following in the Xmath Commands window:

```
oscmd ("type your_directory\Discrete_Cruise_System.doc")
```

Figure 6-2 provides a samping of DocumentIt output from the Discrete_Cruise_System.doc document.

Figure 6-2 **Sample of DocumentIt Output**



```
Number of DataStores in this model = 1
---------------------------------------------------------
DataStore Desired Speed (#0)
---------------------------------------------------------
Description :
No. of Registers = 1
No. of Inputs = 1
No. of Outputs = 1

Name            DataType       UoM           Limit/Range          Accuracy
set speed       DOUBLE                       0.0-0.0              0.0

*****************************************************************

Number of SuperBlocks inputs this model = 6
SUPERBLOCK[0] = Discrete Cruise System
SUPERBLOCK[1] = Cruise Control System
SUPERBLOCK[2] = Set Speed
SUPERBLOCK[3] = Controller Logic
SUPERBLOCK[4] = mux3
SUPERBLOCK[5] = continuous automobile

Number of Unique SuperBlocks inputs this model = 6
SUPERBLOCK[0] = 0
SUPERBLOCK[1] = 1
SUPERBLOCK[2] = 2
SUPERBLOCK[3] = 3
SUPERBLOCK[4] = 4
SUPERBLOCK[5] = 5


---------------------------------------------------------
Top Level SuperBlock model is Discrete Cruise System
---------------------------------------------------------
LEVEL = 0
NUM_SB_IN_I = 6
NUM_SB_OUT_I = 3
SB_FREQ_R = 50.0
SB_SAMPLE_R = 0.02
SB_SKEW_R = 0.0
SB_ACTV_SIG_S = Parent
SB_OUT_POST_S =
SB_ATTR_S = Discrete
SB_HAS_IN_B = 1
SB_HAS_IN_DATA_B = 1
SB_IS_DSCR_B = 1
SB_IS_TRIG_B = 0
External Data Elements
----------------------
Discrete Cruise System External Inputs
```

## 6.2 Generating Customized Documentation

You can customize documentation generated with DocumentIt by using templates. Template files are ASCII files containing text, interspersed with template command parameters that specify DocumentIt output. The TPL programming language lets you modify the templates to control the output of DocumentIt to meet a variety of needs. Various templates are available.

In addition to template command parameters, you can also place publishing software markup commands (for example, FrameMaker, Microsoft Word, or WordPerfect markup commands) in template files, which DocumentIt writes directly to the ASCII output file. The markup commands automatically format the document when it is imported into the corresponding publishing software. See the *Template Programming Language User's Guide*.

Unlike AutoCode, DocumentIt does not have a dialog box for advanced features.

# 7 RealSim

The RealSim controller lets you do real-time simulations of feedback control system models designed in SystemBuild. In this way, you can see how a prototype performs in the real world before actually building the prototype.

The RealSim environment lets you run SystemBuild models in real time: connecting to real external hardware for real-time simulation, rapid prototyping, and hardware-in-the-loop modeling. You can build run-time graphical user interfaces that let you monitor values and change setpoints in the application running on a real-time computer in the same manner that you perform interactive simulations in SystemBuild. In addition to the software tools, real-time computers with analog and digital I/O are available to complete the RealSim environment.

## 7.1 Feedback Control Systems

Control involves interaction between two objects. For example, when you are driving, you watch the speedometer. If you are going too fast, you reduce your speed by letting up on the accelerator pedal. If you are going too slowly, you press down on the accelerator to increase your speed. This type of interaction is actually feedback control; that is, a sensor measures the controlled variable (the car speed), and the information obtained is fed back to influence the controlled variable.

### 7.1.1 Conventional Design

Conventional design of such a control system takes place in stages with several separate tools required for control design, software engineering, data acquisition, and testing.

Design of a control system typically involves engineers:

1. Creating an accurate plant (the variable that is controlled) model.
2. Simulating the model to see how it compares with reality. (Engineers collect data measuring the behavior of interest and then change the model, if necessary.)
3. Building a control system for the plant and then testing the entire control system, including the plant.
4. Implementing and testing the model in hardware, making modifications if necessary.
5. Testing results again in a functional prototype.

The conventional approach has three major flaws:

- It is expensive; you have to modify the prototype or the controller at each stage.
- It is a time-consuming process from conception to finished prototype.
- When you obtain a prototype, you do not know if you have an optimum design.

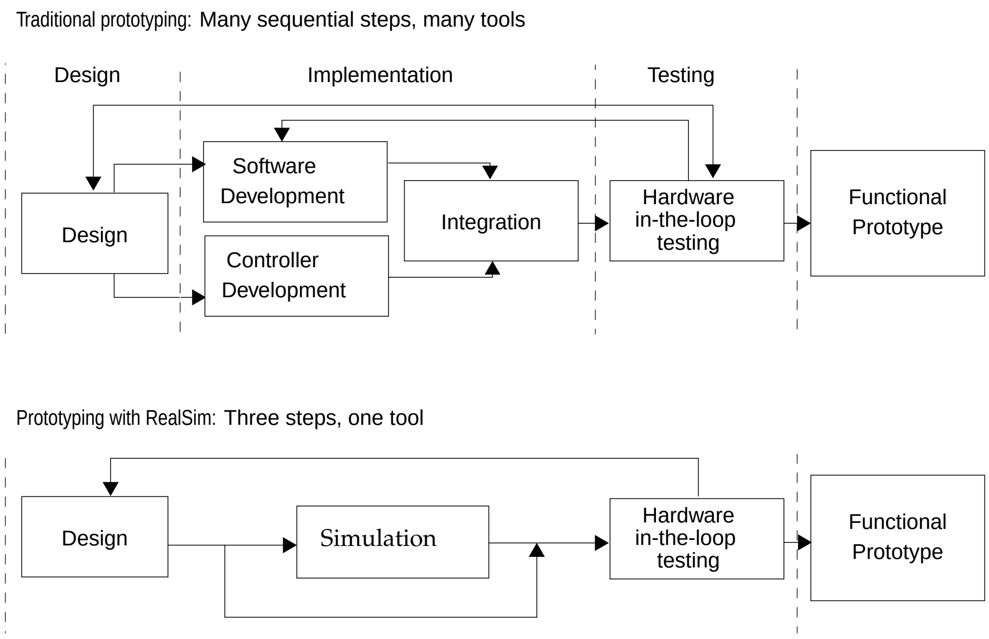
### 7.1.2 Rapid Prototyping

Rapid prototyping uses one tool and three steps to accomplish the following:

- Integrate tools for each stage of system development into a single environment.
- Advance design progress easily through development stages.
- Create a working prototype early in the design process.

Figure 7-1 compares conventional prototyping to rapid prototyping.

Figure 7-1 **Rapid Prototyping Concept**

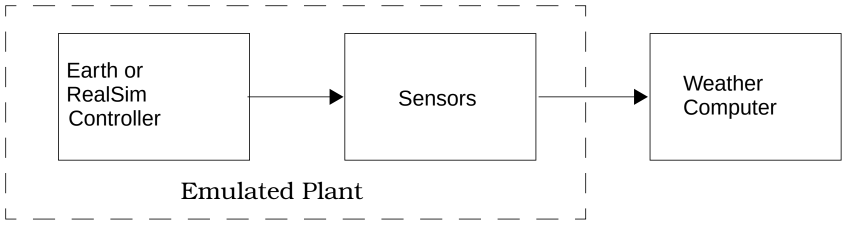

Traditional prototyping: Many sequential steps, many tools
Design
Implementation
Testing
Design
Software Development
Controller Development
Integration
Hardware in-the-loop testing
Functional Prototype
Prototyping with RealSim: Three steps, one tool
Design
Simulation
Hardware in-the-loop testing
Functional Prototype

The MATRIX$_X$ Product Family software implements rapid prototyping. The plant and controller are simulated together, considerably reducing time and development cost to produce a functional prototype. *7.1.4 Building and Testing a Feedback Control System Model*, p.130 describes how to use the MATRIX$_X$ software to build and test a model of a feedback control system.

## 7.1.3 Other Simulations with a RealSim Real-Time Controller

You can use a real-time controller to simulate a device (referred to as *plant emulation*). For example, computers are used in weather forecasting. Various sensors measure atmospheric conditions, the outputs of these sensors are analyzed by the computer, and the computer outputs a forecast. Suppose you want to develop a new system of sensors for improved forecasting and test it. You could connect the new sensors and check them out on a weather-forecasting computer. A faster, easier, and cheaper way is to find out the input and output requirements of the forecasting computer and simulate the climatic sensor output with a RealSim controller (see Figure 7-2). When your design is functional, you can then test the new sensor hardware attached to the real forecasting computer.

Figure 7-2 **RealSim in a Weather Prediction Testbed**



Suppose you want to improve one of the chips in a car's electrical system. You can use a RealSim controller to simulate the functions of that chip. To make changes to the chip, redesign the chip-emulation model by rebuilding it in SystemBuild, and then regenerate the real-time code with AutoCode. You can do this as many times as necessary until the simulated chip performs as desired.

Another possibility is the simulation of a microprocessor-based control system (rapid prototyping). If you have a system that is controlled by a microprocessor, you can simulate the microprocessor with the RealSim controller. As before, you test your design, make changes in SystemBuild, and come up with the optimal design with the RealSim controller performing the functions of the production microprocessor and the control code. When the design is optimized, you can replace the RealSim controller with the actual microprocessor. You can transfer the code generated by AutoCode from the RealSim controller directly to the microprocessor if the appropriate board-specific standalone utilities have been written. You do not have to write additional code in this case.

### *7.1.4 Building and Testing a Feedback Control System Model*

Typical design of a control system model using MATRIX$_X$ software proceeds as described in *4. SystemBuild*.

To build and test a feedback control system model:

1. Simulate the model on the host display to see how it compares with reality.

   As your model runs under the simulator, you can monitor it on the host display, changing signal values and viewing the internal signal values.

   You can also collect data using the data acquisition features of the RealSim controller and compare it with data from the actual plant. If necessary, you can change the model.

2. Linearize the model using SystemBuild.

3. Design a controller using Xmath.
4. Incorporate the controller design into the model via SystemBuild.
5. Simulate the whole design, plant plus controller, and plot the results on a graph to show how the complete system performs.

   The AutoCode code generator generates C language real-time code. Rapid code generation permits major modifications in control strategies instantly, without days or weeks of coding and debugging. Since the code is generated directly from the SystemBuild model, the resulting controller code exactly matches the original design.
6. Implement and test the controller design with real-time software.

You can implement the design you produced with Xmath and SystemBuild directly on the RealSim controller, saving time and improving product quality. You can build the entire prototype as hardware, or you can implement part of your design as hardware and leave the rest of the design simulated. This allows you to build your prototype in stages, with two major benefits:

- The expense of building the prototype can be divided.
- You can build and test a portion of the prototype as hardware to see if it functions as desired before spending time and money to build the complete prototype.

I/O boards let you connect prototype hardware to your model. (See the *RealSim PC Controller System Reference* and the *RealSim AC-1000 Controller System Reference*.) The external prototype hardware that you design replaces a part of the simulation model you designed under SystemBuild.

To connect hardware to your model by using an I/O board:

1. Change the assignment of prototype hardware from the simulation model to the I/O board.
2. Use SystemBuild to generate an RTF file.
3. Use the RealSim **autocode** command to generate real-time software.
4. Use **apbuild** to compile and link the code produced by AutoCode.
5. Use **rtmain** to download and run the application on the RealSim controller.

   You can collect data and plot the results, just as you did during the host simulation earlier. As before, if the plot indicates deficiencies in your model, you can go back to SystemBuild and improve your design. You can then perform the real-time simulation again.

You can also simulate embedded controller environments in real time, testing controllers offline before they undertake critical functions.

Each time you simulate the model and make improvements, you get closer to realizing the real-world model.

## 7.2 RealSim Controller Models

For detailed up-to-date information about RealSim controller models, and their hardware and software components, see the following:

- *RealSim PC Controller System Reference*
- *RealSim AC-1000 Controller System Reference*

## 7.3 RealSim Tutorials

Now that you have a basic understanding of the RealSim controller, you are ready to actually build and run a model on the RealSim controller. This section presents two tutorials:

- Running a Demonstration Model shows you how to copy and run a demonstration project on your RealSim controller. Details of the project's creation are omitted from this section. The tutorial shows you how to download and run one of the demonstration models.
- Building and Running a New Model provides the simulation of a simple model using a gain block.

**NOTE:** We assume that the MATRIX$_X$ Product Family software and RealSim controller have already been installed in accordance with the MATRIX$_X$ Product Family Installation Procedures and the *RealSim Installation Guide for PC Controllers* (*Windows Hosts*) or the *RealSim Installation Guide for AC-1000 Controllers* (*Windows NT Hosts*). We also assume that you have set up and powered up your controller as instructed in your system reference manual and installation guide.

For a complete procedure to build and run a model on the RealSim controller, see the *RealSim User's Guide*.

### *7.3.1 Running a Demonstration Model*

We provide various demonstration models that you can download and run on your RealSim controller. These demonstrations range in complexity from easy to very difficult. This section provides an example of one of the easier demonstrations that you can run.

This demonstration is an example of a common automotive cruise control system. The simulation allows you to activate the cruise control and set the speed of a simulated vehicle. The simulation also models a typical vehicle's acceleration and braking properties. It is designed to familiarize you with the use of the RealSim controller and its development tools by providing step-by-step instructions that describe how you:

- Use **copydemo** to copy a demonstration model
- Generate downloadable code
- Generate an I/O configuration for the model
- Run the simulation in real-time and monitor its progress

### *Preparing a Demonstration Model*

To download and run your simulation on a RealSim controller:

1. If a RealSim Command Prompt window is not already open, from your Windows taskbar select Start→ Programs→RealSim *version*→RealSim to open a window with the current RealSim environment.

⚠ **CAUTION:** Do not use any other Command Prompt window when simulating a RealSim project. You must have this RealSim environment.

2. On the command line, enter the **cpcprj** command to switch your current default directory to something like **c:\users\*your_name*\cpcprj**:

   ```
   cpcprj
   ```

3. Enter the **copydemo** command to copy a demo:

   ```
   copydemo
   ```

   If you are asked to choose which RealSim target because of multiple-controller installation, select **C_PC** for this demo.

   A list of available demonstration models appears.

4. For this example, choose the following demonstration model:

   ```
   super_cruise
   ```

5. When prompted for the location for the copy, press Return.

   This creates a subdirectory called **super_cruise** and copies the files for the **super_cruise** model to that directory.

6. When prompted for the controller target name, type the IP name, and press Return.

   You can obtain the IP name from whoever installed the RealSim controller.

7. When prompted for the RealSim target type, enter the target type name, for example, **C_PC**, and press Return.

8. When prompted for an animation package, choose the Altia Graphics:

   ```
   ALTIA
   ```

9. Continue to answer the remaining prompts using the default values.

   AutoCode generates code for the demonstration model, and the system compiles and links it.

### Activating Data Acquisition

With the demonstration model completely built, activate the RealSim GUI to gain access to the RealSim utilities. In this case, we need to use the Data Acquisition Editor.

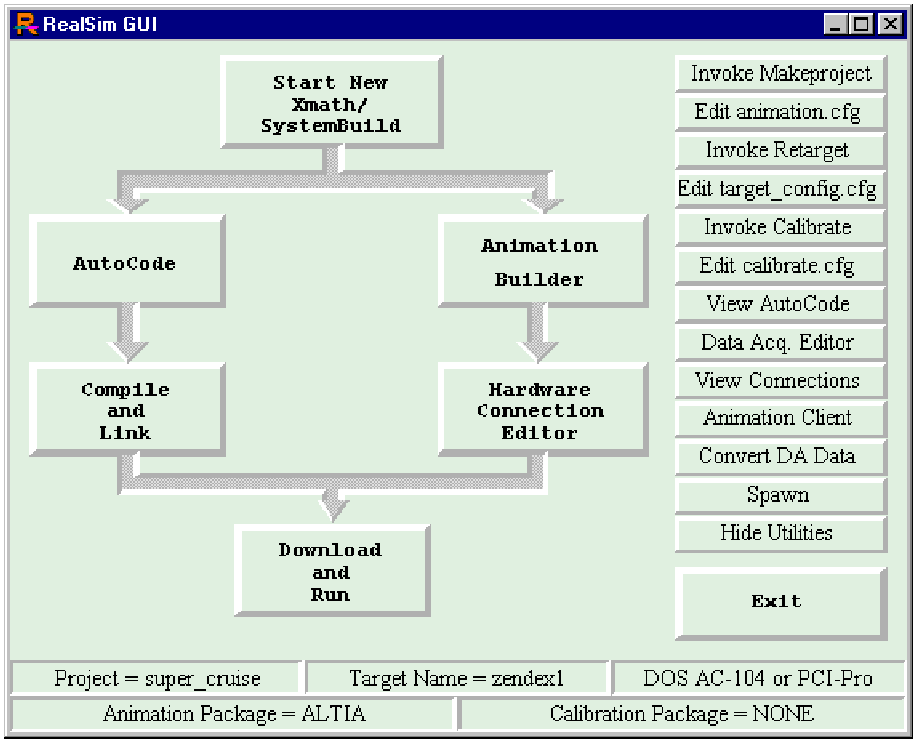
To gain access to the RealSim utilities:

1. From the RealSim command window, launch the RealSim GUI by entering:

   ```
   realsim
   ```

   The RealSim GUI provides a graphical interface to the rapid-prototyping process (see Figure 7-3).

Figure 7-3 **RealSim GUI for Demonstration Model**



Project = super_cruise | Target Name = zendex1 | DOS AC-104 or PCI-Pro
Animation Package = ALTIA | Calibration Package = NONE

See *7.3.2 Building and Running a New Model*, p.144 for more information on using this GUI.

2. Launch the Data Acquisition Editor by clicking the Data Acq. Editor button.

   The data acquisition screen appears.

   Notice that the DISPLAY field indicates **SB_INPUTS**.

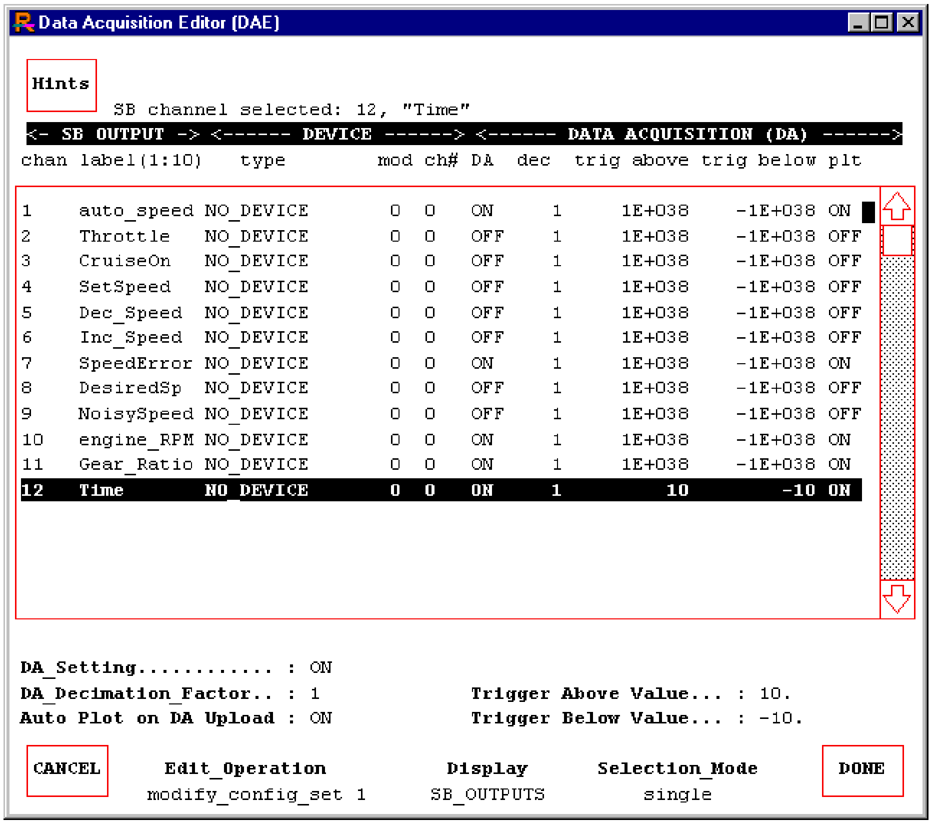
3. Click DISPLAY to change that value to **SB_OUTPUTS**.

   Now, the outputs of the **super_cruise** model appear.

4. Click channel #12, **Time**; notice that data acquisition (DA_Setting) is set, has trigger values (Trigger Above Value and Trigger Below Value), and has the auto-plot feature (Auto Plot on DA Upload) set

   Figure 7-4 shows the Data Acquisition Editor. The table summarizes the data acquisition settings for each of the outputs.

Figure 7-4 **Data Acquisition Editor**



```
SB channel selected: 12, "Time"
<- SB OUTPUT -> <------ DEVICE ------> <------ DATA ACQUISITION (DA) ------>
chan label(1:10)    type         mod ch# DA   dec   trig above trig below plt

1    auto_speed NO_DEVICE        0   0   ON     1      1E+038    -1E+038 ON
2    Throttle   NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
3    CruiseOn   NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
4    SetSpeed   NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
5    Dec_Speed  NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
6    Inc_Speed  NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
7    SpeedError NO_DEVICE        0   0   ON     1      1E+038    -1E+038 ON
8    DesiredSp  NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
9    NoisySpeed NO_DEVICE        0   0   OFF    1      1E+038    -1E+038 OFF
10   engine_RPM NO_DEVICE        0   0   ON     1      1E+038    -1E+038 ON
11   Gear_Ratio NO_DEVICE        0   0   ON     1      1E+038    -1E+038 ON
12   Time       NO_DEVICE        0   0   ON     1          10        -10 ON

DA_Setting........... : ON
DA_Decimation_Factor.. : 1            Trigger Above Value... : 10.
Auto Plot on DA Upload : ON           Trigger Below Value... : -10.

CANCEL    Edit_Operation        Display      Selection_Mode    DONE
          modify_config_set 1   SB_OUTPUTS   single
```

5. Click DONE to return to the RealSim GUI.

### *Running the Demonstration Model on the RealSim Controller*

After setting the data acquisition information, the demonstration model is ready to be simulated in real time. Since we have the Auto Plot data acquisition setting turned on, we have to use Xmath to view the plot.

To simulate the model in real time:

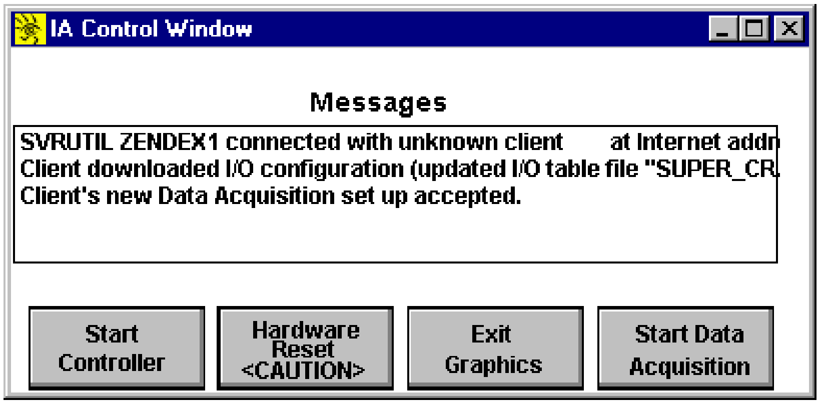
1. If you do not already have Xmath running, click the Start New Xmath/ SystemBuild button on the RealSim GUI to start Xmath.

   At this point do nothing more with Xmath.

2. On the RealSim GUI, click the Download and Run button.

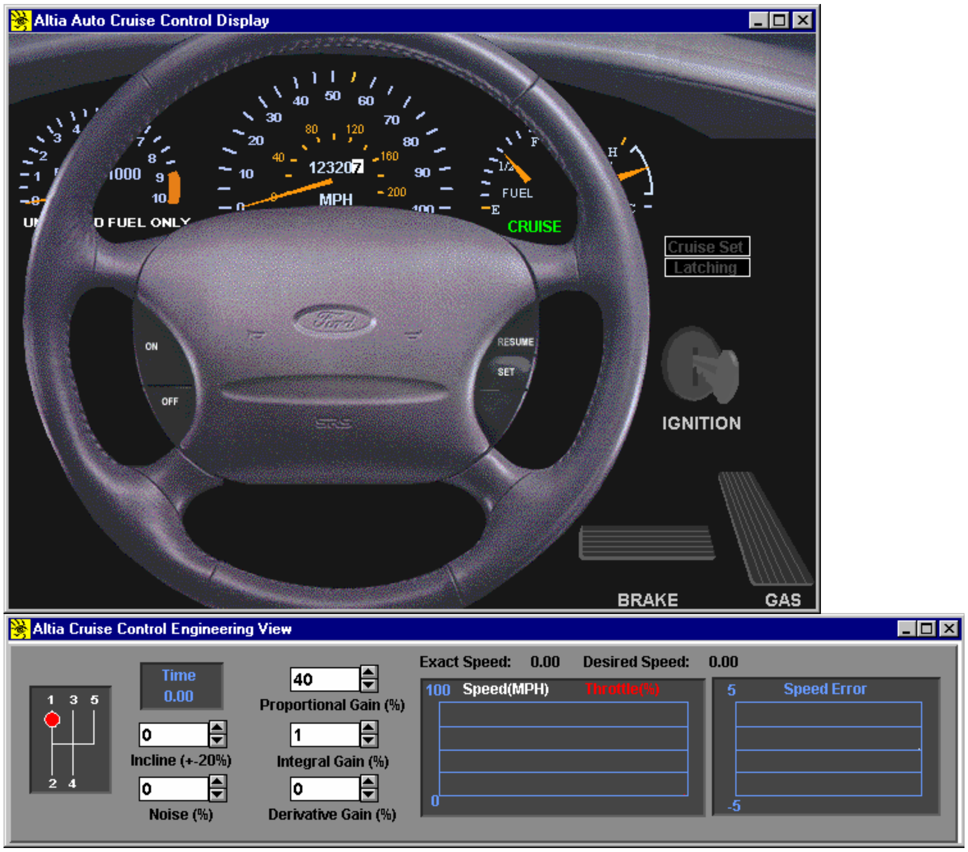
   In a Command Prompt window, you see the connection to the controller and download of the demonstration model. When the download is complete, the Altia graphics window and IA Control window appear (see Figure 7-5 and Figure 7-6).

Figure 7-5 **Altia IA Control Window**



```
SVRUTIL ZENDEX1 connected with unknown client       at Internet addr
Client downloaded I/O configuration (updated I/O table file "SUPER_CR
Client's new Data Acquisition set up accepted.
```

Figure 7-6 **Altia Graphics for super_cruise Demonstration Model**



Time 0.00

Incline (+-20%): 0

Noise (%): 0

Proportional Gain (%): 40

Integral Gain (%): 1

Derivative Gain (%): 0

Exact Speed: 0.00 Desired Speed: 0.00

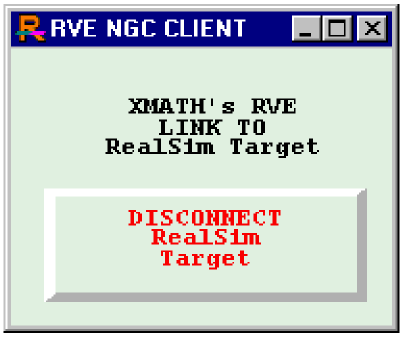
3. Once the Altia graphics and IA Control window appear, switch back to Xmath, and enter the following command into the Xmath Commands window:

```
lock=attach_realsim()
```

The purpose of this command is to establish a data connection between Xmath and the IA client (Altia in this case). This provides a data channel for the data acquisition data to move from the controller to the IA client and then to Xmath for plotting.

When the connection is made, the RVE NGC Client window appears (see Figure 7-7); it contains the text: **XMATH's RVE Link To RealSim Target.**

Figure 7-7 **RVE NGC Client Window**



4. Move the RVE NGC Client window aside or iconify it so as to not obscure the Altia graphic and IA Control window.
5. In the IA Control window, click the Start Controller button.

   The model is now running on the controller.

### *Using the Altia Graphics for Super_Cruise Demonstration Model*

With the super_cruise model now running, the Altia graphics are active. This section uses some of the graphics to show the simulation. You can change the position of the gas pedal, use the brake, or change simulation parameters (Incline and Noise, for example). Play with the various operations, and then have fun with the simulation!

#### *Changing the Position of the Gas pedal:*

To change the position of the gas pedal:

1. In the Altia graphics window, click near the *top* of the image of the GAS pedal. With MB1 held down, move the cursor up and down.

   Notice that the pedal tracks the mouse.
2. Release the mouse button to set the GAS pedal position.

   The value represented by the pedal *does not* change while you are holding the mouse button, so you must release the button before the value changes. The pedal remains where you leave it when you release the mouse.

Assuming that you pushed down on the pedal (more gas), notice that the speedometer begins to increase. Also, in the Altia Cruise Control Engineering View, notice that the Speed plot rises.

### *Changing the Position of the Brake Pedal*

To change the position of the brake pedal:

1. In the Altia graphics window, click near the *bottom* of the image of the BRAKE pedal. With MB1 held down, move the cursor up and down.

   Notice that the pedal tracks the mouse.

2. Release the mouse button, and the pedal remains.

   Again, the value represented by the pedal *does not* change until you release the mouse button. When the brake is on, BRAKE changes to BRAKE ON.

   Assuming that you pushed the BRAKE pedal down, notice that the vehicle's speed decreases.

### *Acquiring Data with the Altia Interactive Animation Client*

The Altia client does not currently support pre-triggered data acquisition, so you must manually trigger the data acquisition.

To manually trigger data acquisition:

1. If your vehicle is stopped, get it moving again by pressing the GAS pedal (see above).

2. When it is evident that vehicle's speed is changing, click the Start Data Acquisition button in the IA Control window.

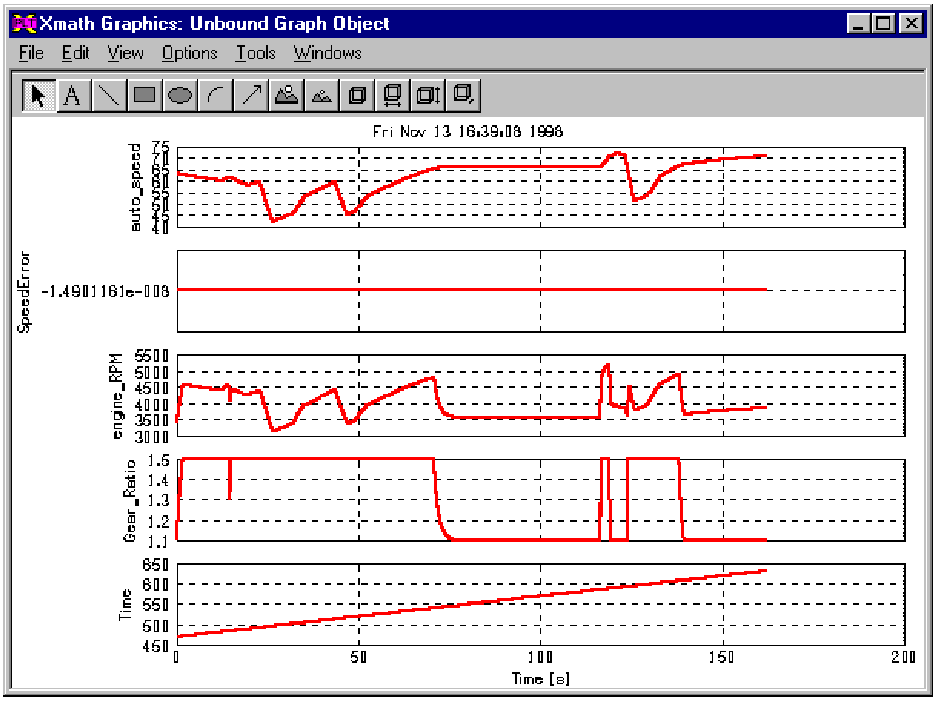
   Some messages appear in the message area about the acquisition.

   Notice that the name on the button previously labelled Start Data Acquisition has changed.

3. After three to five seconds, click the Stop Data Acquisition button.

   The data is stored into a **.raw** file that Xmath can read directly. Since we have set the auto-plot feature and we are connected to Xmath, the data is automatically plotted. Figure 7-8 is an example output; your plot probably looks different.

Figure 7-8 **Acquired Simulation Data**



For additional information on automatic plotting, see the *RealSim Command Reference*, Chapter 3 (**attach_realsim**, **read_rawfile**, and **realsim_ autoplot topics**) or Xmath documentation for details.

### Run-Time Variable Editing with the Altia Client

The Altia client does not currently support a graphical interface to run-time variable editing (RVE). You must use the equivalent Xmath commands. (See the *RealSim Command Reference*, Chapter 3, for details on the RealSim commands executed from Xmath to perform RVE: **rve_start**, **rve_get**, **rve_put**, **rve_stop**, **rve_update**, **rve_quit.**)

To perform run-time variable editing using Xmath commands:

1. With the simulation running, switch back to the Xmath Commands window and enter the commands below.

   a. Start RVE:

   ```
   rve_start()
   ```

   b. Get run-time parameter:

   ```
   rve_get("stiffness")
   ```

   c. Set new value of run-time parameter:

   ```
   rve_put("stiffness", 3.2)
   ```

   d. Update the value onto the controller:

   ```
   rve_update()
   ```

   e. Discontinue RVE:

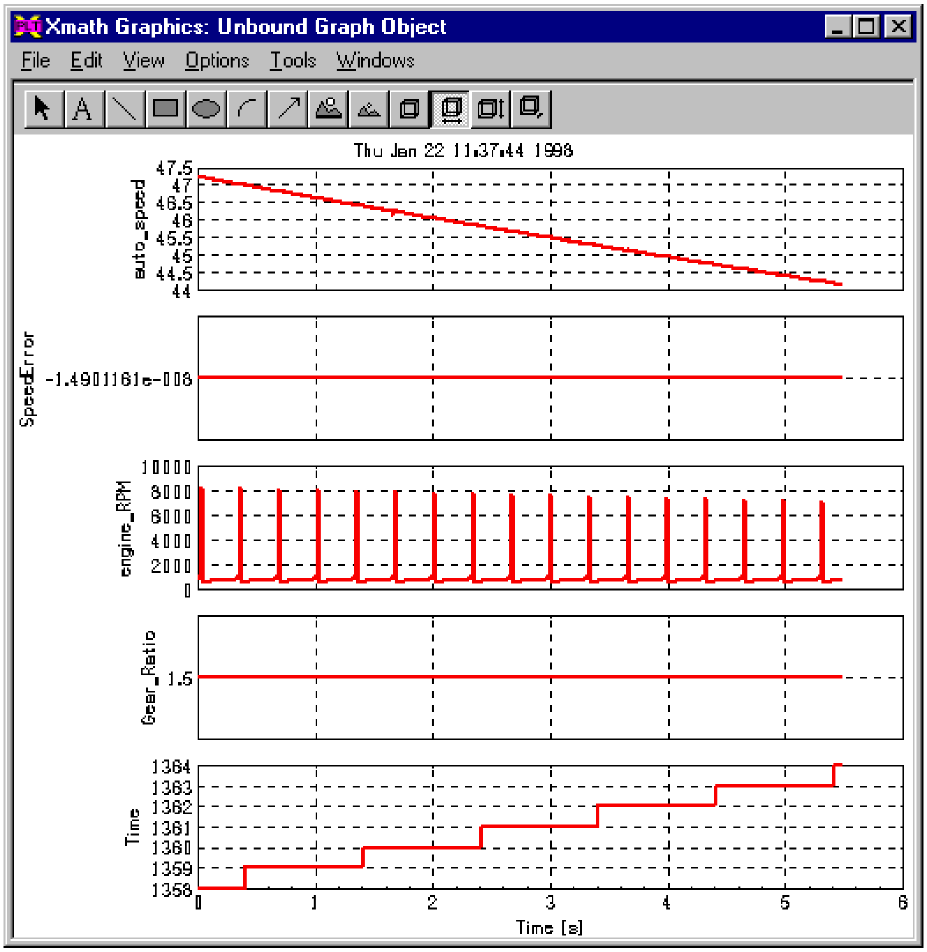
   ```
   rve_stop()
   ```

   The **stiffness** run-time parameter affects the stiffness of the drivetrain in our simulated vehicle. Changing the value as we did dramatically affects the model.

2. Click the Start Data Acquisition button in the IA Control window.
3. After three to five seconds, click the Stop Data Acquisition button.

   Since you already have auto-plotting set after data acquisition, you can see evidence of the parameter change in your plot. Figure 7-9, p.143 is a sample output; your plot probably looks different.

Figure 7-9 **Acquired Data After a Change in Parameter**



### *Ending the Simulation*

To end the simulation and stop the controller:

1. Click the Stop Controller button.

   At this point, you can restart the simulation by clicking the Restart Controller button. Restarting the controller restores all parameter settings, including the one changed by RVE.

If you restart the controller, return to Step 1 to end the simulation.

2. When you are all done and the controller is stopped, disconnect the connection to Xmath by clicking the DISCONNECT RealSim Target button in the NGC Client window.

   The NGC Client window disappears.

3. Click Hardware Reset to stop and reset the controller, close the Altia client, and return to the RealSim GUI.

**NOTE:** Clicking Exit Graphics instead of Hardware Reset closes the Altia client but *does not* stop the controller.

## 7.3.2 Building and Running a New Model

This section shows you how to build a simple model using a gain block with a separate animation picture and run it on the **C_PC** type target controller. (Instructions for other models are similar.)

**NOTE:** The procedures given in this section assume that you have set up and powered up your controller as instructed in your system reference.

### Creating a RealSim Project

To create a RealSim project:

1. If a RealSim Command Prompt window is not already open, from your Windows taskbar select Start→ Programs→RealSim *version*→RealSim to open a window with the current RealSim environment.

**CAUTION:** Do not use any other Command Prompt window when creating a RealSim project. You must have this RealSim environment.

2. Enter the **cpcprj** command to switch your current default directory to something like **c:\users\*your_name*\cpcprj**:

```
cpcprj
```

3. Create a subdirectory and change to it:

```
mkdir gain_ia
cd gain_ia
```

4. Bring up the RealSim graphical user interface (GUI):

   ```
   realsim
   ```

5. When the Invoke Makeproject dialog appears, click YES.

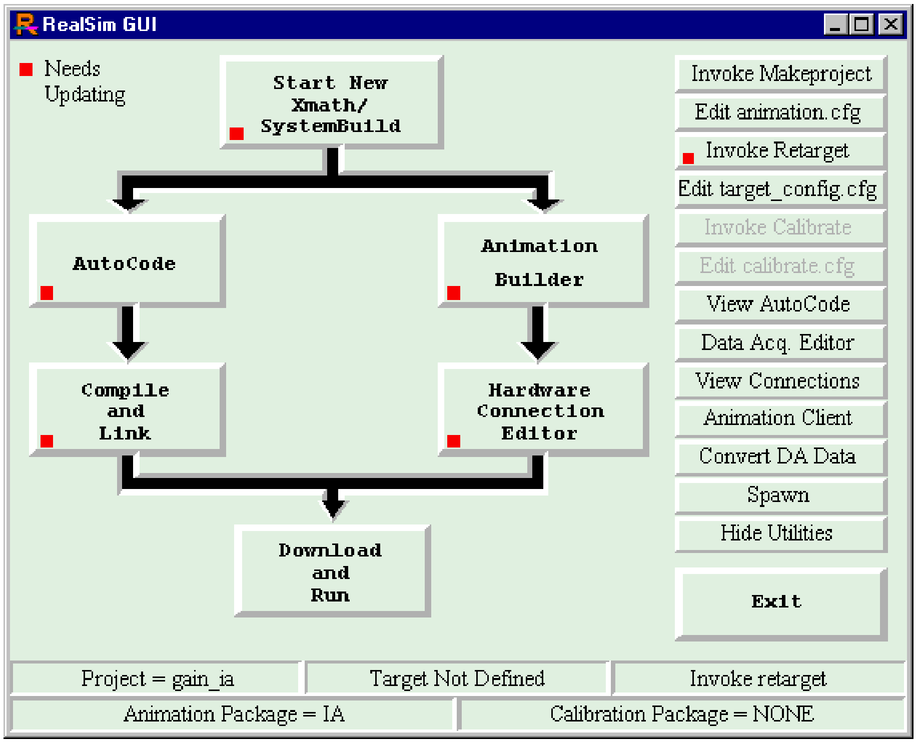
   Another Command Prompt window automatically opens. You might have to cycle through the open windows on your desktop to find this new window.

6. Click inside the new Command Prompt window.

7. For the Project Name prompt in the **gain_ia** directory, press Return.

   The RealSim GUI, shown in Figure 7-10, comes on view.

Figure 7-10 **RealSim GUI After Creation of gain_ia Project**



Project = gain_ia | Target Not Defined | Invoke retarget
Animation Package = IA | Calibration Package = NONE

8. If the RealSim GUI does not have the utility buttons shown on the right of Figure 7-10, click the Show Utilities button.

9. In the list of utilities on the GUI, click Invoke Retarget.

10. Answer the prompts (not all prompts appear on all controllers) as follows, replacing the responses provided with the correct responses for your controller. (Answer any other prompt that is not listed here by pressing Return.)

    a. **Controller host name [realsimcontroller]:** *your_target_name*

    b. **RealSim target type (C_PC C_PPC604) [C_PC]: C_PC**

    c. **Animation Package to use (IA VBIA ALTIA) [IA]: IA**

    When you complete this process, the following message appears:

    ```
    Successfully updated target_config.cfg
    ```

### *Creating the SystemBuild Model and an RTF File of the RealSim Top-Level SuperBlock*

In this section, we build a model in SystemBuild that will be used for SystemBuild and RealSim simulation. However, the requirements for each are a little different. Later, we add interactive animation to this model.

The requirements for the RealSim simulation consist of a top-level SuperBlock and a save file (RTF) of this SuperBlock. For this model, the top-level SuperBlock is called **gain_ia**; it consists of a simple Gain block and an AlgebraicExpression block.

To run the RealSim model in SystemBuild simulation, you must create a wrapper SuperBlock that contains a reference to your RealSim top-level SuperBlock (**gain_ia**) and the interactive animation UserCode block (**usria1**). The outputs of the RealSim top-level SuperBlock must be hooked up to the inputs of the UCB, and the outputs of the UCB must be hooked up to the inputs of the RealSim top-level SuperBlock. The **usria1** UserCode block performs the function of calling the interactive animation display during SystemBuild simulation.

To create a SystemBuild model and an RTF file of the RealSim top-level SuperBlock:

1. If you have not already started Xmath, click the Start New Xmath/SystemBuild button on the RealSim GUI.

2. When Xmath window comes up, type the following in the commands area:

    ```
    build
    ```

    After a short time, SystemBuild is loaded, and the Catalog Browser and the SystemBuild Editor window are present on your screen.

3. From the Catalog Browser, create a new SuperBlock (see *4.3.1 Creating a New SuperBlock*, p.55).

   The SuperBlock Properties dialog comes on view.

4. In the SuperBlock Properties dialog:

   a. Enter **gain_ia** in the Name field.

   b. Set Inputs to **1**.

   c. Set Outputs to **2**.

   d. Change the SuperBlock Type to **Discrete**.

   e. Click OK.

5. Place a gain block within the **gain_ia** SuperBlock (see *4.3.2 Creating a New Block in a SuperBlock*, p.56).

6. Open the gain block's block properties by placing the mouse pointer over the gain block and pressing Return.

   a. Enter **gain_block** in the Name field.

   b. On the Outputs tab, enter **gainoutput** in the table cell for the #1 output of the Output Label column.

   c. On the Display tab, enable Show Output Labels.

   d. Click OK.

7. Place an algebraic expression block (from the Algebraic palette) with the **gain_ia** SuperBlock.

8. Open the gain block's block properties by placing the mouse pointer over the gain block and pressing Return.

   a. Enter **time_block** in the Name field.

   b. On the Code tab, enter the following equation: **Y = T;**

      This sets the output equal to time.

   c. Click OK.

9. Connect the gain block to external inputs and outputs (see *Connecting Blocks*).

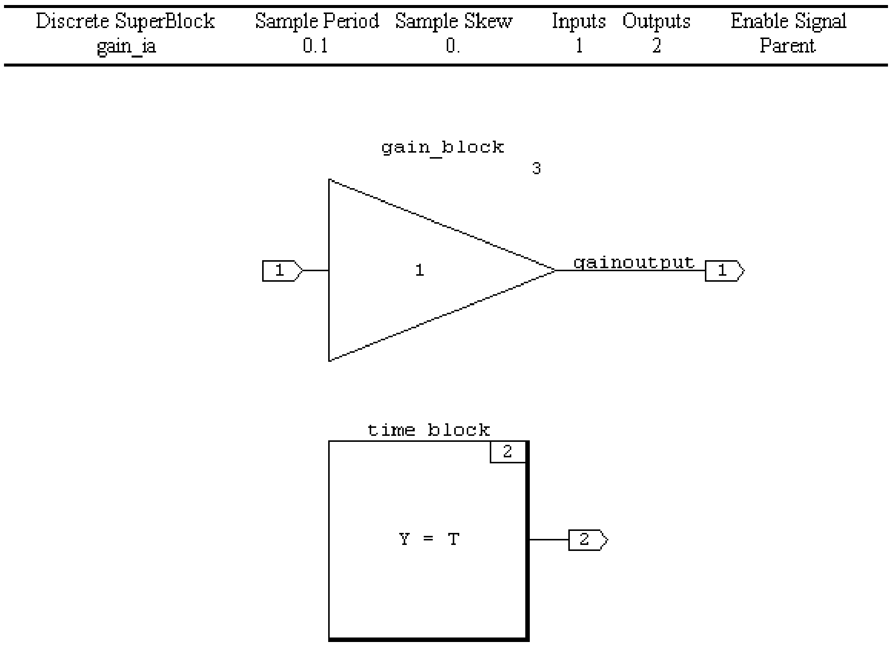
   a. Connect the first external input of **gain_ia** to the input of **gain_block**.

   b. Connect the output of **gain_block** to the first external output of **gain_ia**.

c. Connect the output of **time_block** to the second external output of **gain_ia**.

Your diagram should now look similar to the one in Figure 7-11.

Figure 7-11 **SuperBlock gain_ia After Connections**



| Discrete SuperBlock | Sample Period | Sample Skew | Inputs | Outputs | Enable Signal |
|---|---|---|---|---|---|
| gain_ia | 0.1 | 0. | 1 | 2 | Parent |

10. Update the current SuperBlock in the Editor window into the Catalog Browser by selecting File→Update, and close this Editor window by selecting File→Close.
11. From the Catalog Browser, create another new SuperBlock.
12. In the SuperBlock Properties dialog:
    a. Enter in the Name field: **gain_top**.
    b. Set Inputs to **0** and Outputs to **1**.
    c. Change the SuperBlock Type to **Discrete**.
    d. Ensure that the Sample Period is **0.1**.
    e. Click OK.

13. Create a UserCode block (UCB) in the newly created SuperBlock.

    The UCB is found on the User Programmed palette of the Palette Browser.

**NOTE:** This code block brings the Interactive_Animation panel into the SystemBuild simulation.

14. Open the UCB's block properties by placing the mouse pointer over the user code block and pressing Return.

    a. In the Name field, name the block **ia**.

    b. On the Parameters tab, blank out the File Name parameter value so that there is no filename specified.

    c. For the Function Name parameter value, enter **usria1**.

**NOTE:** The above name ends with the numeral one (1).

    d. Set the Inputs field to **2** and the Outputs field to **1**.

    e. Click OK.

15. Select the **ia** block in the SystemBuild Editor window, and then select Edit→Flip Horizontal to flip the orientation of the newly created SuperBlock.

16. Update the current SuperBlock in the Editor window into the Catalog Browser by selecting File→Update.

17. Create a reference to the existing SuperBlock, **gain_ia**:

    a. Bring up the Catalog Browser.

    b. In the Catalog Browser, click the **SuperBlocks** folder in the left pane.

    c. In the right pane, click the icon representing **gain_ia,** and drag the SuperBlock into the SystemBuild Editor window containing **gain_top**.

18. Connect the SuperBlock reference and the UCB as follows:

    a. Connect the first output of the **gain_ia** SuperBlock reference to the first input of the UCB.

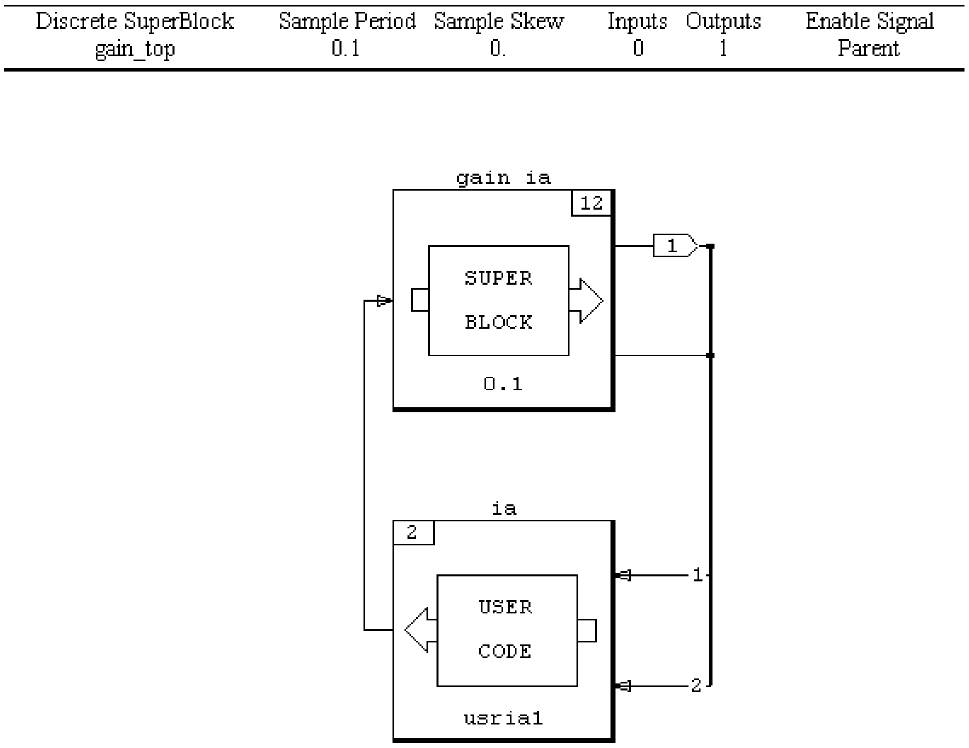
    b. Connect the second output of the **gain_ia** SuperBlock reference to the second input of the UCB.

    c. Connect the output of the UCB to the input of the **gain_ia** SuperBlock reference.

d. Connect the first output of the **gain_ia** SuperBlock to the external output of **gain_top** SuperBlock.

You should now have a diagram similar to Figure 7-12.

Figure 7-12 **Completed gain_top SuperBlock Model**



| Discrete SuperBlock | Sample Period | Sample Skew | Inputs | Outputs | Enable Signal |
|---|---|---|---|---|---|
| gain_top | 0.1 | 0. | 0 | 1 | Parent |

19. From the SystemBuild Editor window containing **gain_top**, select File→Update, and close this Editor window by selecting File→Close.

20. From the Catalog Browser, save your model, and name it **gain_top.cat** (see *Saving a SuperBlock*).

21. Generate an RTF file (**.rtf**) for the **gain_ia** SuperBlock:

    a. Select the **gain_ia** SuperBlock in the right pane of the Catalog Browser.

    b. Select Tools→AutoCode in the Catalog Browser to open the Generate Real-Time Code dialog.

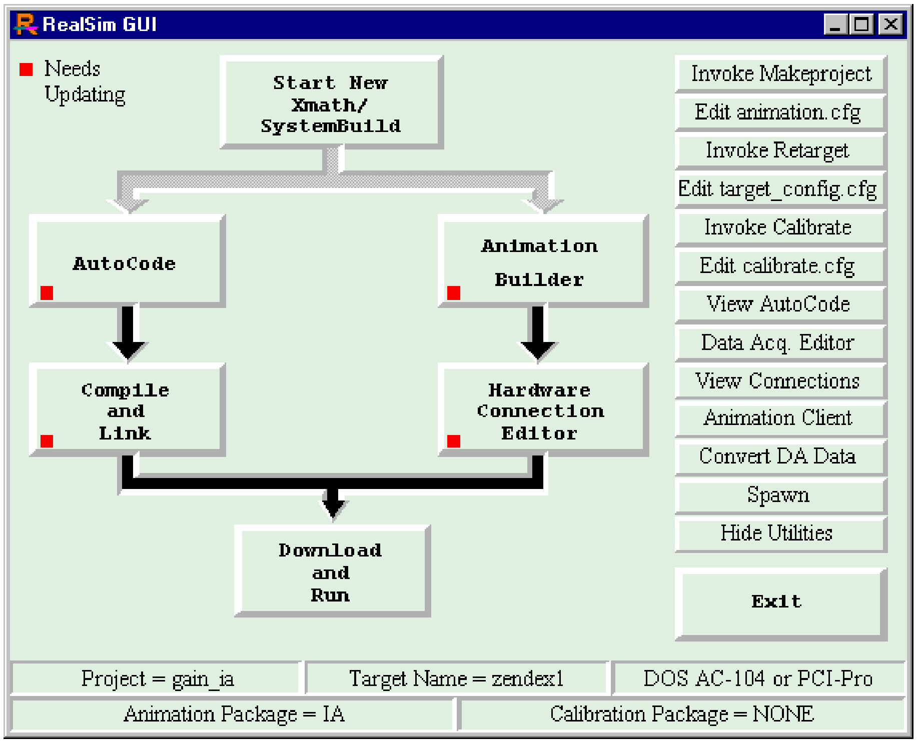
c. In the Generate Real-Time Code dialog, change the Language option to be **RTF only**.

d. Click OK to generate the RTF file.

You can watch the progress of the RTF file generation process in the log area of the Xmath window.

At this point, we have made a new project and have just completed building and generating the RTF file for our model. The RealSim GUI follows the progress of creating, building, and executing a model on the controller. Compare Figure 7-13 to Figure 7-10, p.145 to see the progression.

Figure 7-13 **RealSim GUI After Model and RTF Creation**



Project = gain_ia | Target Name = zendex1 | DOS AC-104 or PCI-Pro
Animation Package = IA | Calibration Package = NONE

## Building the Interactive Animation Panel

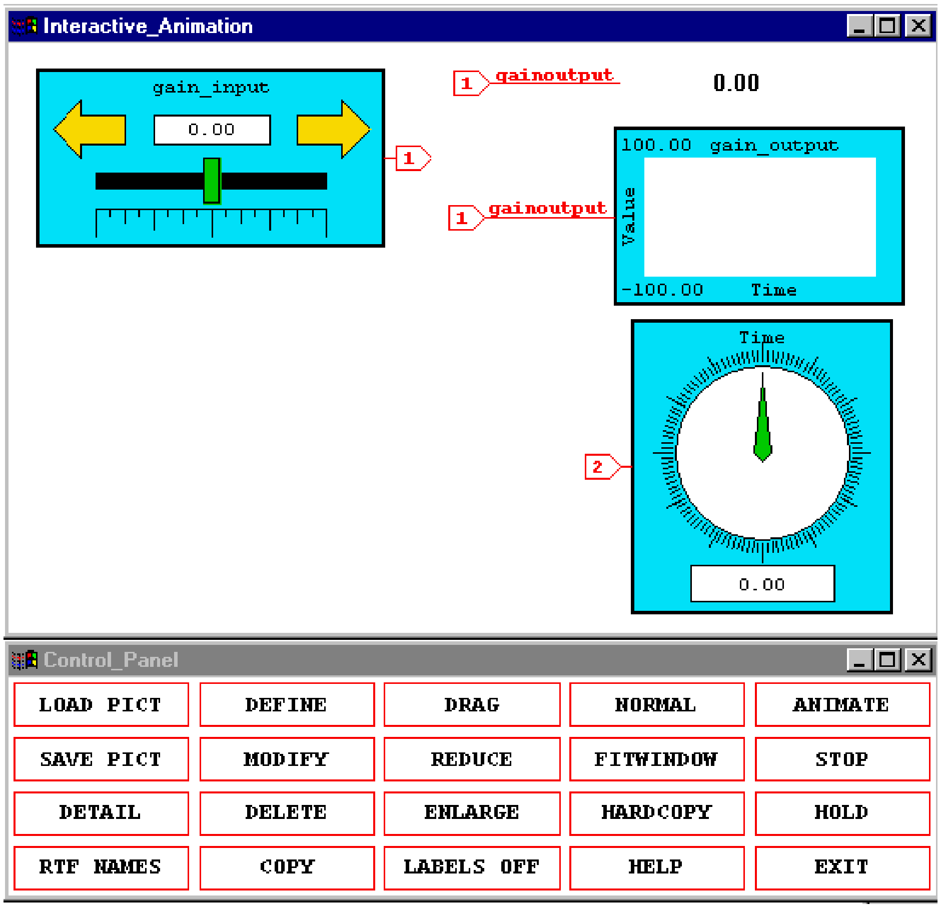
The Interactive Animation graphics that you create below are the graphical representation of the User Code Block labeled **ia** in the gain SuperBlock.

To create graphics for Interactive Animation:

1. Click the Animation Builder button on the RealSim GUI to launch the Interactive Animation builder tool.

   The Interactive_Animation picture window and the Control_Panel window appear. Figure 7-14 shows these two windows with the picture that we are going to build.

Figure 7-14 **Completed Interactive Animation Picture**



2. Build the Slide icon shown in Figure 7-14 as follows:

   a. Click DEFINE in the Control_Panel window.

      The Animation Palette appears. This palette contains six sets of icons, each with two pages. Each set is indicated by two letters in the second row at the bottom of the palette; for example, MI indicates Monitor Animation Icons. The name is expanded in the middle cell on the bottom row of the palette. The top row has two cells numbered 1 and 2; these represent the pages of the set. The current set and page have asterisks on either side of the mnemonic and page number. You can change either by clicking in the appropriate cell.

   b. Click CI and then 1 at the bottom of the window to bring up page 1 of the Control Animation Icons

**NOTE:** There are several Slide icons that are very similar in this palette; if you do not use the one from the CI set, your model will not work correctly.

   c. Click the Slide icon to place it in the Interactive_Animation window.

      The Animation Palette disappears.

**NOTE:** This Slide icon contains an initial value of 0, which shows on the icon.

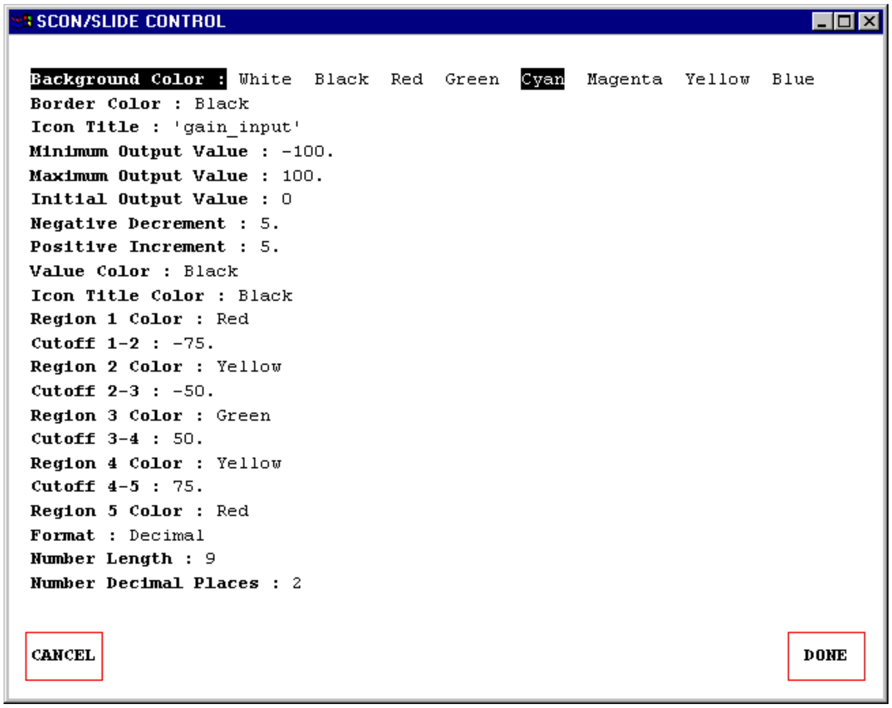
   d. Double-click the Slide icon in the Interactive_Animation window to bring up the SCON/SLIDE CONTROL dialog (see Figure 7-15).

   e. Set the Icon Title to **gain_input**.

   f. Ensure that the Maximum Gain Value is **100** and the Minimum Gain Value is **-100**.

   g. Click DONE when complete.

Figure 7-15 **SCON/SLIDE CONTROL Dialog**



```
Background Color : White  Black  Red  Green  Cyan  Magenta  Yellow  Blue
Border Color : Black
Icon Title : 'gain_input'
Minimum Output Value : -100.
Maximum Output Value : 100.
Initial Output Value : 0
Negative Decrement : 5.
Positive Increment : 5.
Value Color : Black
Icon Title Color : Black
Region 1 Color : Red
Cutoff 1-2 : -75.
Region 2 Color : Yellow
Cutoff 2-3 : -50.
Region 3 Color : Green
Cutoff 3-4 : 50.
Region 4 Color : Yellow
Cutoff 4-5 : 75.
Region 5 Color : Red
Format : Decimal
Number Length : 9
Number Decimal Places : 2
```

3. Build the Numerical icon shown in Figure 7-14, p.152 as follows:

   a. Click DEFINE in the Control_Panel window.

   b. Click MI and then 2 at the bottom of the window to bring up page 2 of the Monitor Animation Icons.

   c. Click any one of the numbers that you see to place it in the Interactive_Animation window (example uses **4.00**).

      The numbers themselves mean nothing; they simply illustrate the font and font size that will appear in your animation.

   d. Double-click the Numerical icon in the picture window to bring up the DV/DISPLAY VALUE dialog.

   e. Examine these values, and make any desired changes.

      The example uses the defaults.

   f. Click DONE when complete.

4. Build the Single Line icon shown in Figure 7-14, p.152 as follows:
   a. Click DEFINE in the Control_Panel window.
   b. Click MI and then 1 at the bottom of the window to bring up page 1 of the Monitor Animation Icons.
   c. Click the Single Line icon to place it in the picture window.

      The Animation palette disappears.
   d. Double-click the Single Line icon in the picture window to bring up the SC1/STRIP CHART dialog.
   e. Ensure that the Maximum Value is **100** and the Minimum Value is **-100**.
   f. Change the Icon Title to be **gain_output** (program encloses in single quotes).
   g. Set the X Axis Label to be **Time**.
   h. Set the Y Axis Label to be **Value**.
   i. Change the First Threshold Color to **Blue** for greater visibility.
   j. Click DONE when complete.
5. Build the Altimeter icon shown in Figure 7-14, p.152 as follows:
   a. Click DEFINE in the Control_Panel window.
   b. Click MI and then 1 at the bottom of the window to bring up page 1 of the Monitor Animation Icons.
   c. Click the Altimeter icon to place it in the picture window.

      The Animation palette disappears.
   d. Double-click the Altimeter icon in the picture window to bring up the SC1/STRIP CHART dialog.
   e. Change the Icon Title to **Time**.
   f. Set One Rotation Value to **60**.
   g. Set Rotation Ratio to **12**.
   h. Set Number of Major Tick Marks to **12**.
   i. Click DONE to dismiss the dialog.

6. Click RTF NAMES in the Control_Panel window to load the labels and names from the generated RTF file.

   The name of your generated file appears in the dialog.

7. Accept the default RTF file by clicking DONE.

8. Connect the icons to the inputs and outputs of the model as outlined below.

   This is just like connecting a block within a SystemBuild Editor window except that the outputs of an icon connect to inputs, and the inputs of an icon connect from outputs.

   a. Click LABELS ON in the Control_Panel window to show the connections.

      Initially, there are none, but small, red indications appear.

   b. Connect the output of the Slide icon named **gain_input** to the first **gain_ia** input by clicking MB2 inside the Slide icon and then clicking MB2 in an empty region of the picture window. In the Connection Editor dialog, click in the circle representing the slide control output and then the circle representing the **gain_ia** input; then click Done.

**NOTE:** Clicking Ctrl MB1 is equivalent to clicking MB2.

**NOTE:** If the Connection Editor window does not appear, look behind your other windows.

   c. Connect the Numerical icon to the first **gain_ia** output by clicking MB2 in an empty region of the picture window and then inside the Numerical icon. In the Connection Editor dialog, click in the circle representing the **first gain_ia** output and then the circle representing the display value; then click Done.

   d. Connect the Single Line icon, now named **gain_output**, to the first **gain_ia** output by clicking MB2 in an empty region of the picture window and then clicking MB2 inside the Single Line icon. In the Connection Editor dialog, click in the circle representing the first **gain_ia** output and then the circle representing the display value; then click Done.

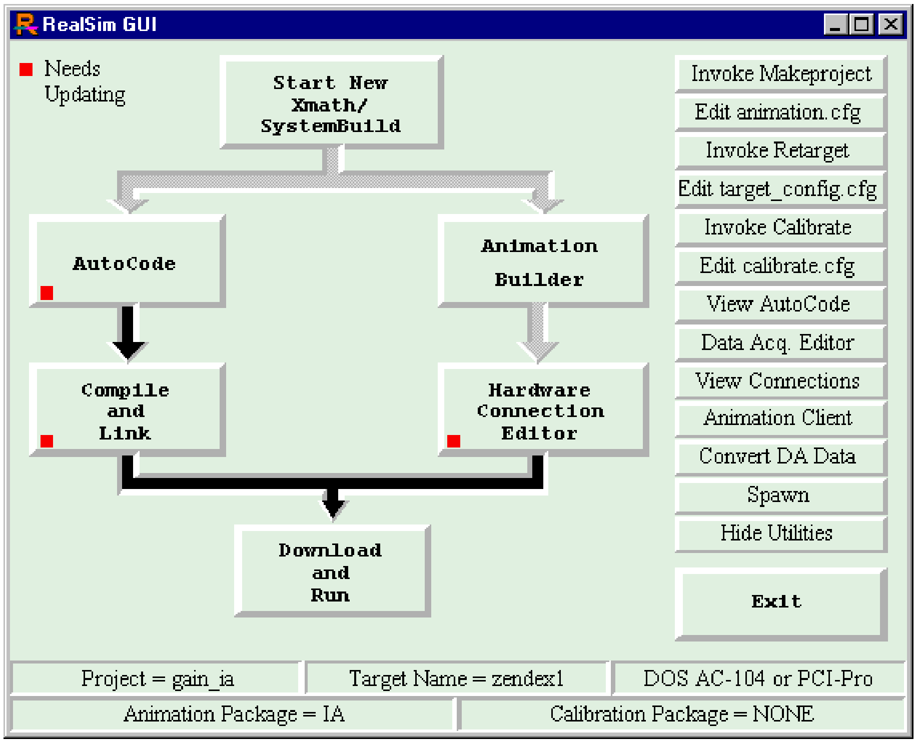
      You have now hooked up two icons to the first output of **gain_ia**.

   e. Connect the Altimeter icon, now named **Time**, to the second **gain_ia** output by clicking MB2 in an empty region of the picture window and then clicking MB2 inside the Altimeter icon. In the Connection Editor dialog, click in the circle representing the second **gain_ia** output and then the circle representing the altimeter; then click Done.

9. Save the picture file by clicking SAVE PICT in the Control_Panel window.
10. Click DONE in the subsequent dialog to accept the default picture filename, **gain_ia.pic**.
11. Click EXIT to close the interactive animation builder too.

    Notice that the RealSim GUI now indicates that the Animation Builder is complete, as shown in Figure 7-16.

Figure 7-16 **RealSim GUI after Animation Builder Completed**



Project = gain_ia
Target Name = zendex1
DOS AC-104 or PCI-Pro
Animation Package = IA
Calibration Package = NONE

### *Simulating the Model in SystemBuild*

To simulate the model with the SystemBuild simulator using interactive animation:

1. From the Xmath command area, enter the following value:

   ```
   t = [0:0.1:1000]';
   ```

2. Make the SystemBuild Catalog Browser window the active window.

   If you closed that tool after you created the model, you need to start it again and load the **gain_top.cat** file that you saved in Creating the SystemBuild Model and an RTF File of the RealSim Top-Level SuperBlock, p.146.

3. From the Catalog Browser, open the **gain_top** SuperBlock in a SystemBuild Editor window (see *4.3.4 Opening a SuperBlock in the Editor*, p.58).

4. From the SystemBuild Editor window, select Tools→Simulate.

   The SystemBuild Simulation Parameters dialog appears.

5. On the Parameters tab, set the following parameters:

   a. Set Time Vector/Variable to **t**.

   b. Leave Input Data/Variable blank (no external inputs to SuperBlock).

   c. Enable the Plot Outputs checkbox.

   d. Enable the Interactive check box.

   e. Click OK.

      The Interactive Simulator (ISIM) and the Interactive_Animation_Display windows appear.

6. In the Interactive Simulator window, select Simulation→Resume to begin.

7. Click in the Interactive_Animation_Display window.

   During simulation, you can move the Slide icon's slider to vary the external input.

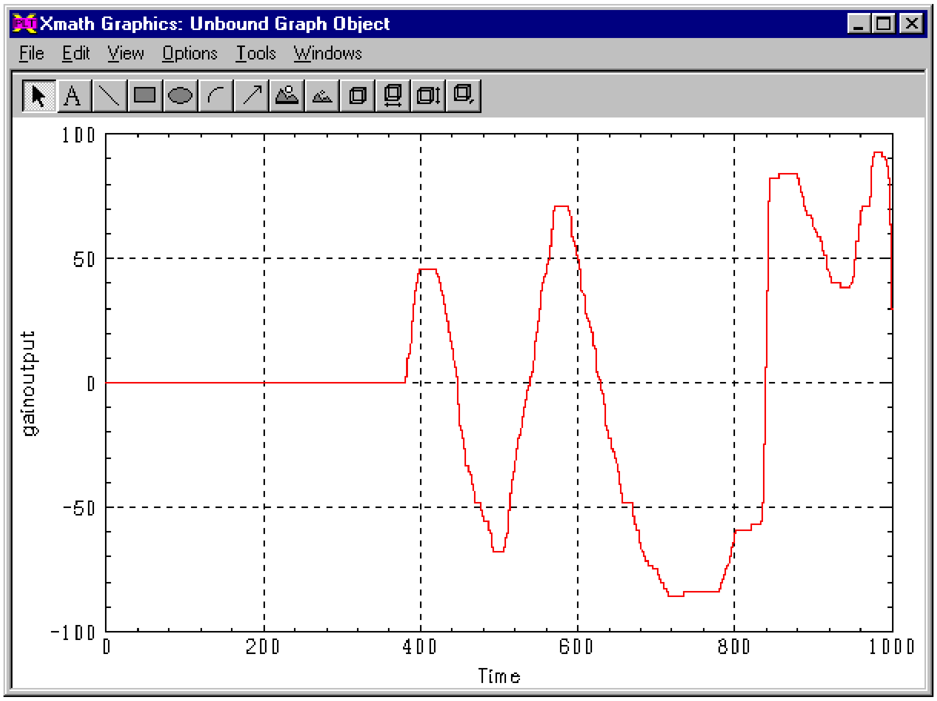
8. Move the slider by clicking and holding down MB1 on the rectangular shape just above the scale and below the value indicator; move the slider to the left and right.

   The slider tracks the mouse movements. Moving to the left decreases the slider's value, while moving to the right increases the value. Notice that the output graph changes as the slider changes.

9. When the simulation completes, select Simulation→Resume to run the simulation again, or select File→Exit to end the interactive simulation and return to SystemBuild.

   When the Time reaches 1000, the simulation completes and the Interactive_Animation_Display window closes. After you exit, the plot of your simulation appears. Figure 7-17 shows a simulation plot.

Figure 7-17 **Simulation Plot**


Xmath Graphics: Unbound Graph Object
File Edit View Options Tools Windows
100
50
0
-50
-100
gainoutput
0
200
400
600
800
1000
Time

10. Close the Plot window.
11. Exit SystemBuild by selecting File→Exit in the Catalog Browser.

    This closes all open windows in SystemBuild.
12. If you receive the prompt, **Save Changes?**, click NO since you have not made any changes to the model.
13. Exit Xmath by selecting File→Exit.

    If you are asked, there is no need to save any data.

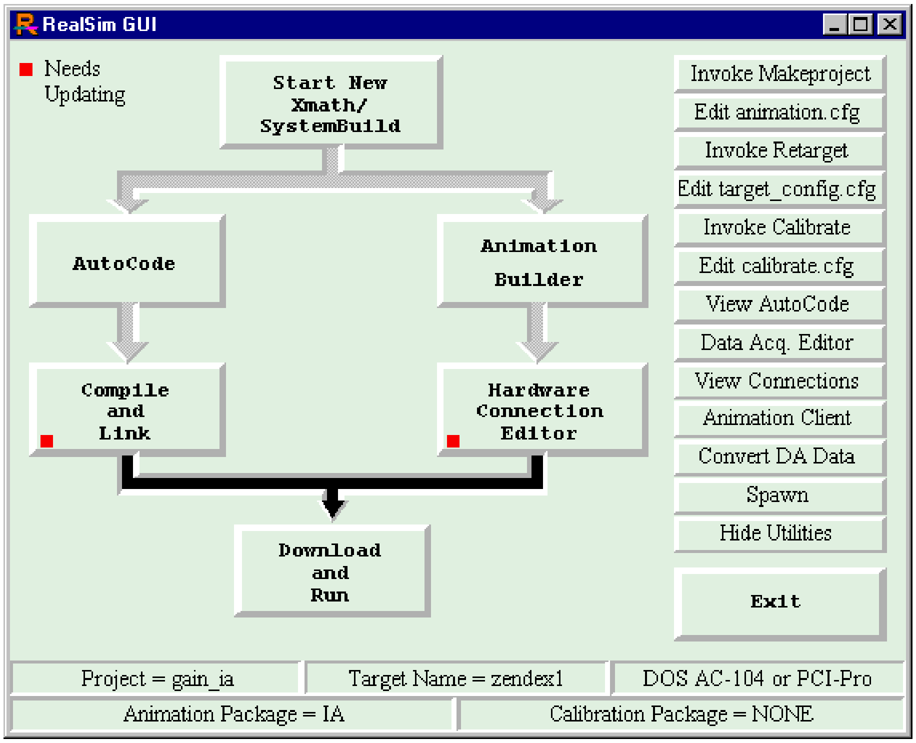
## Simulating the Model on the RealSim Controller

To simulate the model on the controller:

1. Click AutoCode on the RealSim GUI.

   Notice that the RealSim GUI continues to update your progress (see Figure 7-18).

Figure 7-18 **RealSim GUI After Running AutoCode**



Project = gain_ia | Target Name = zendex1 | DOS AC-104 or PCI-Pro
Animation Package = IA | Calibration Package = NONE

   In a companion Command Prompt window, AutoCode notifies you of the C code generation.

2. Click Hardware Connection Editor on the RealSim GUI.

The Hardware Connection Editor (HCE) window appears; it lists SB Inputs. All I/O types are **MONITOR_INPUT**.

a. No changes are needed, so click DONE.

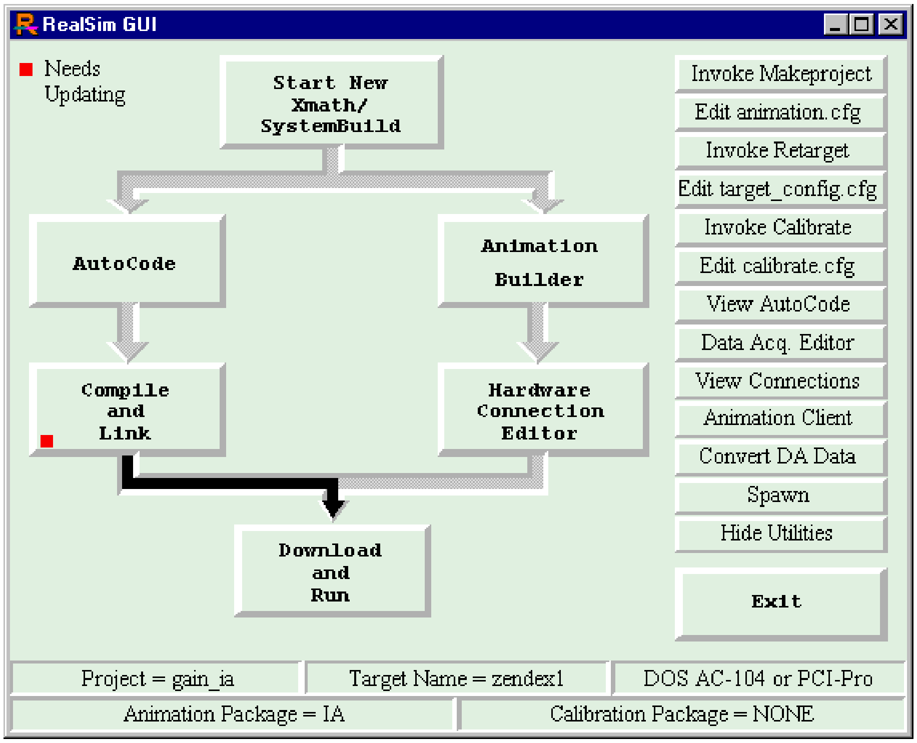
Another Hardware Connection Editor (HCE) window appears; it lists SB Outputs. All I/O types are **NO_DEVICE**.

b. No changes are needed, so click DONE.

The Hardware Connection Editor (HCE) window closes.

The RealSim GUI continues to be updated based on your progress (see Figure 7-19).

Figure 7-19 **RealSim GUI After Getting the Hardware Connection Editor**



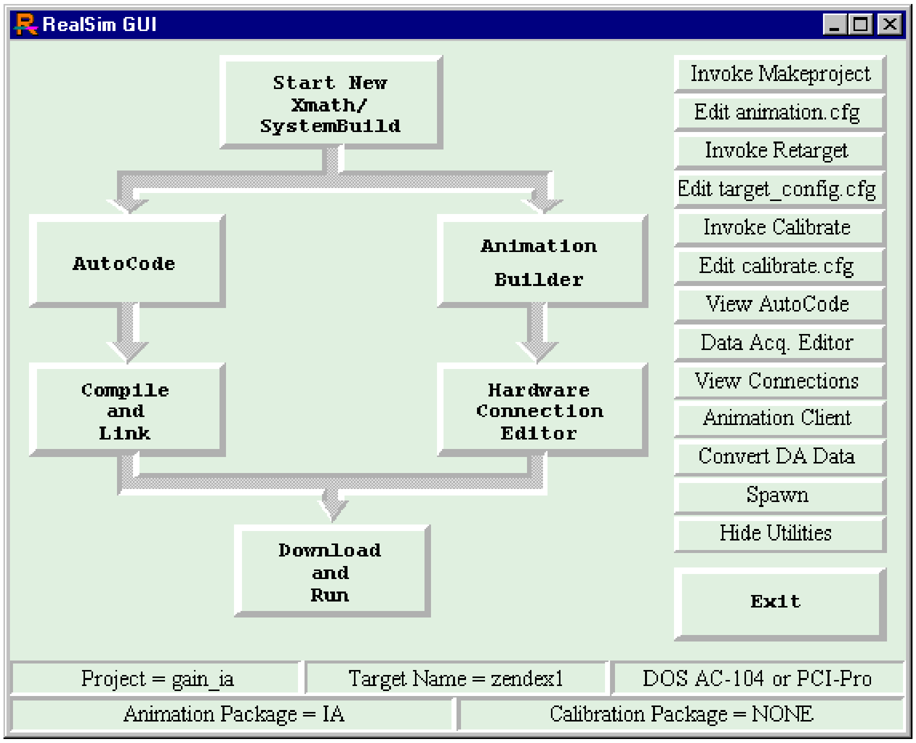
Project = gain_ia
Target Name = zendex1
DOS AC-104 or PCI-Pro
Animation Package = IA
Calibration Package = NONE

3. Click Compile and Link on the RealSim GUI.

   In the Command Prompt companion window, you are notified that the generated C code is compiled and linked.

   Notice that the RealSim GUI continues to update your progress (see Figure 7-20).

Figure 7-20 **RealSim GUI After Compiling and Linking**



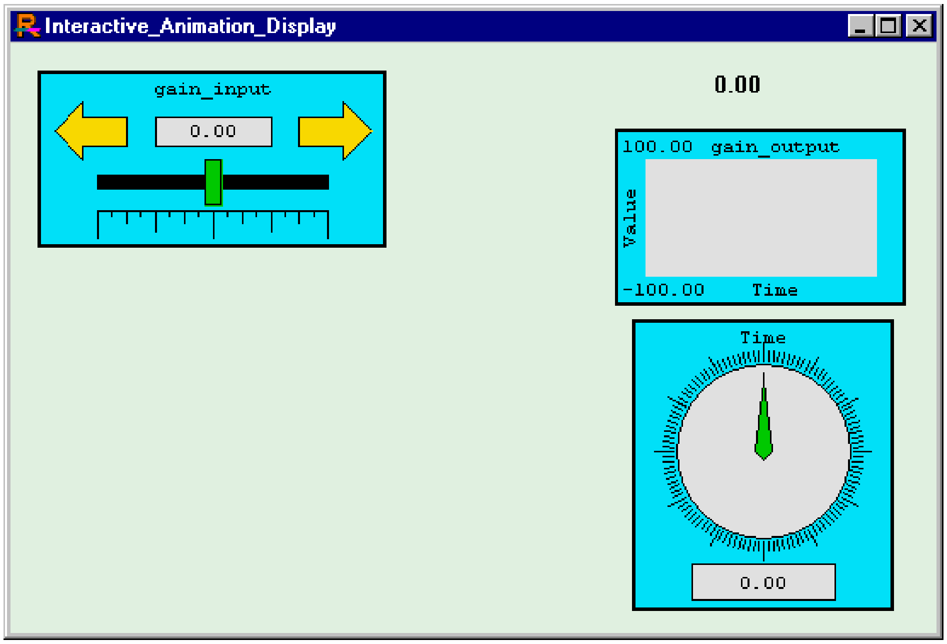
Project = gain_ia | Target Name = zendex1 | DOS AC-104 or PCI-Pro
Animation Package = IA | Calibration Package = NONE

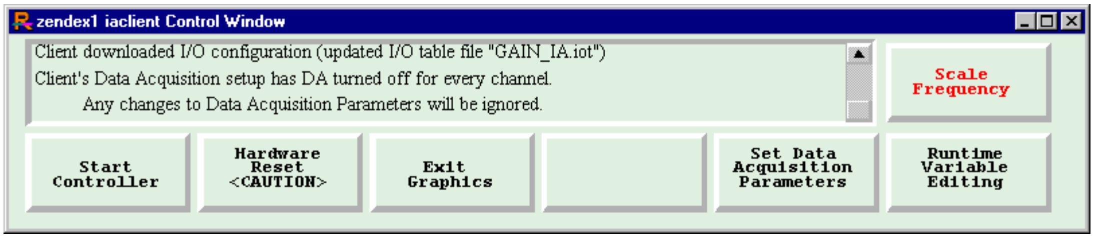
4. Click Download and Run on the RealSim GUI.

   In the Command Prompt companion window, you are notified that the model is downloaded to the controller.

   The iaclient Control window and the Interactive_Animation_Display window appear (see Figure 7-21).

Figure 7-21 **Interactive_Animation_Display and iaclient Control Windows**





Client downloaded I/O configuration (updated I/O table file "GAIN_IA.iot")
Client's Data Acquisition setup has DA turned off for every channel.
Any changes to Data Acquisition Parameters will be ignored.

5. Click START CONTROLLER in the iaclient Control window.

   The model is now running on the controller.

6. Move the slider during the simulation by clicking and holding down MB1 on the rectangular shape just above the scale and below the value indicator. While the mouse button is down, move the slider to the left and right.

   The slider tracks the mouse movements. Moving to the left decreases the slider's value, while moving to the right increases the value. Notice that the output graph changes as the slider changes.

7. Perform one or more of the following actions:
   a. Click STOP CONTROLLER in the iaclient Control window to stop the simulation.
   b. Click RESTART CONTROLLER to restart the simulation. Go to Step 6.
   c. Click EXIT GRAPHICS to return to the RealSim GUI.

      The iaclient Control window disappears, but the controller is still running.
   d. Click HARDWARE RESET (CAUTION) to stop the controller and to stop the graphics.
8. Click EXIT on the RealSim GUI.
9. To re-connect the controller, type **rtmain -r** in the Command Prompt window.

# *Index*