NI MATRIXx™

Getting Started Guide

April 2007
371148B-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 (0) 9 725 72511, France 33 (0) 1 48 14 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

Chapter 3
Xmath

Chapter 4
SystemBuild

Chapter 5
AutoCode

Chapter 6
DocumentIt

Chapter 7
MATRIXx Help Resources

Appendix A
Technical Support and Professional Services

Glossary

Index

About This Manual

This manual contains information about getting started with MATRIXx. This manual introduces the MATRIXx development environment, the project system, Xmath, SystemBuild, AutoCode, and DocumentIt.

Conventions

The following conventions appear in this manual:

» The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

This icon denotes a note, which alerts you to important information.

This icon denotes a tip, which alerts you to advisory information.

bold Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

`monospace bold` Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

`monospace italic` Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Platform Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

1

Introduction to MATRIXx

MATRIXx provides a complete solution for mathematical analysis, modeling, control design, simulation, automatic code generation, and automatic documentation generation.

This chapter provides information about the MATRIXx products and navigating the MATRIXx Integrated Development Environment (IDE).

MATRIXx Product Family

The MATRIXx product family consists of the following products:

- Xmath provides an interactive command-based environment you use to analyze numbers and systems, visualize data, and script and program graphical interfaces.

  For more information about Xmath, refer to Chapter 3, *Xmath*.

- SystemBuild provides a visual modeling and simulation environment. You use this environment to construct models of dynamic systems, such as control loops and aerospace/automotive applications.

  You can also use the SystemBuild Simulator to simulate the non-real-time behavior of system models. The Simulator sends results to Xmath so you can verify, plot, and analyze data. You can also run the Simulator interactively to provide a graphical depiction of the simulation.

  For more information about SystemBuild, refer to Chapter 4, *SystemBuild*.

- AutoCode generates ANSI C or Ada code from a SystemBuild model. This code is suitable for a stand-alone simulation or for use in an embedded real-time system. AutoCode includes a Template Programming Language (TPL) that you can use to customize the generated code.

  For more information about AutoCode, refer to Chapter 5, *AutoCode*.

- DocumentIt generates documentation from a SystemBuild model. You can generate documentation in FrameMaker, Microsoft Word, and WordPerfect formats. Use this documentation to construct easy-to-read manuals and reports for a SystemBuild model. Like

AutoCode, DocumentIt includes a TPL for customizing the generated documentation.

For more information about DocumentIt, refer to Chapter 6, *DocumentIt*.

Two applications of the MATRIXx product family are rapid control prototyping (RCP) and hardware-in-the-loop (HIL) testing. In an RCP configuration, you use a non-real-time simulation to quickly verify alternative design choices and data sets. In an HIL configuration, you use hardware that executes the model in real time.

Figure 1-1 shows how you can use the MATRIXx product family to achieve these configurations.

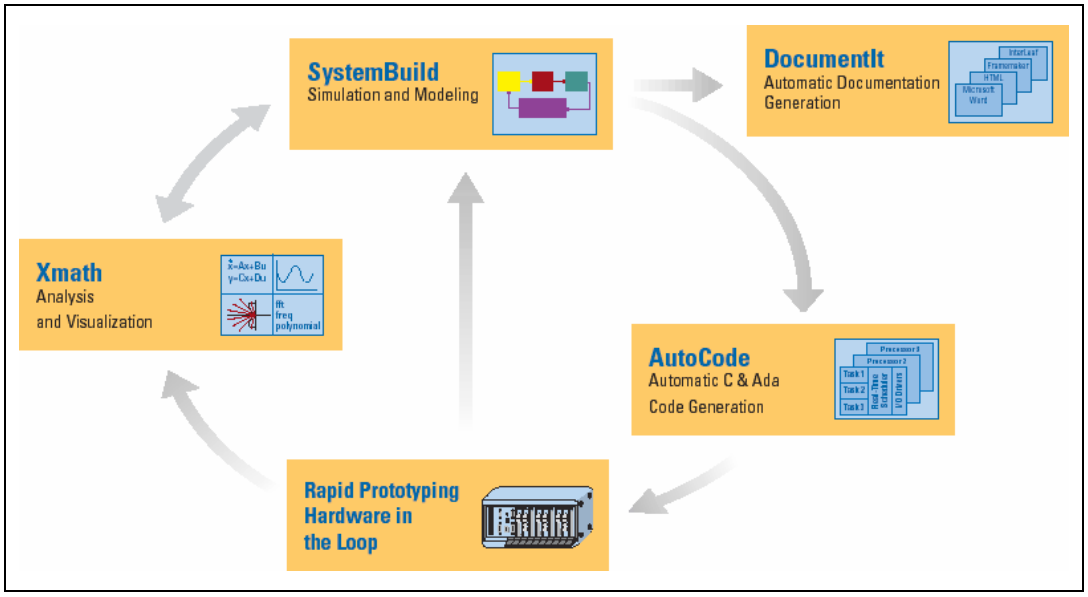

Figure 1-1. MATRIXx Product Family Overview

MATRIXx Integrated Development Environment

The MATRIXx IDE is the graphical framework you use to access the MATRIXx products, tools, and modules. The IDE consists of several views that you open and close to display relevant information. This section provides information about views and the IDE itself.

Views

A view is a dockable window that you can use to customize the look of the IDE. MATRIXx includes the following views:

- **Solution Explorer**—Displays the current solution and any included projects. For information about projects and solutions, refer to Chapter 2, *Projects and Solutions*.
- **Messages**—Displays any messages that MATRIXx applications send to the IDE.
- **Outline**—Displays a list of blocks in the open SuperBlock. This view also displays a list of bubbles in the open State Transition Diagram (STD).
- **Windows**—Displays a list of open applications.
- **Properties**—Displays properties that let you configure the item you selected in the **Solution Explorer** view.
- **Blocks**—Displays the palettes and blocks you use to build SuperBlocks. For information about SuperBlocks, refer to Chapter 4, *SystemBuild*.
- **Bubbles**—Displays the bubbles and SuperBubbles you use to build an STD. For information about STDs, refer to the *SystemBuild State Transition Diagram Block User Guide*, available by selecting **Help»MATRIXx Bookshelf**.
- **Tools**—Displays the tools you use to build SuperBlocks in SystemBuild.

You use the **View** menu to show and hide views.

Figure 1-2 shows a MATRIXx window that displays the **Solution Explorer** and **Blocks** views.

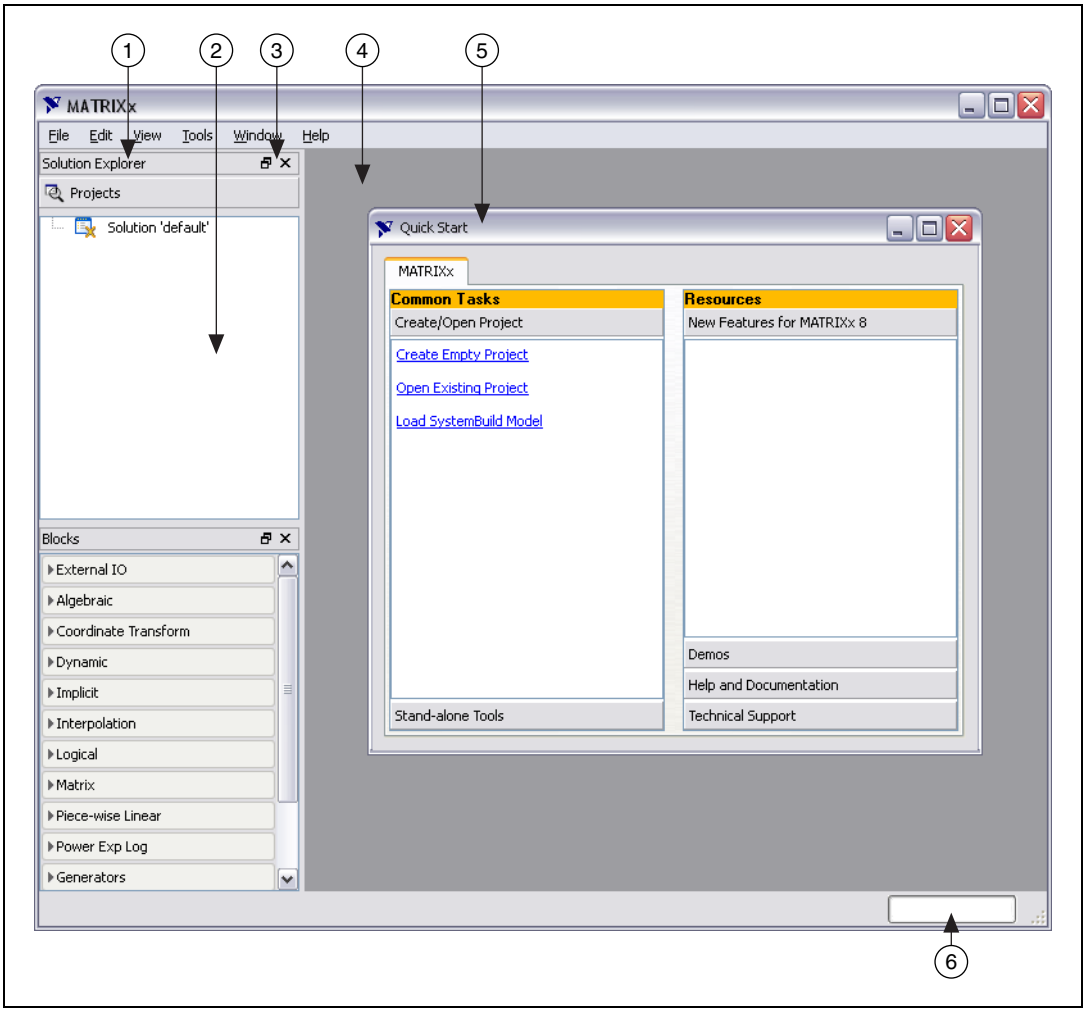

Figure 1-2. The MATRIXx Window

This window consists of the following components:

1. **View title bar**—Drag the title bar to reposition the view within the MATRIXx window. Right-click the title bar to display a list of other views you can select.
2. **View information display**—This is where MATRIXx displays the information that the view contains.

3. **View restore/close buttons**—Use these buttons to resize or close the view, respectively.
4. **Work area**—MATRIXx displays windows, such as Xmath or SystemBuild windows, here.
5. **Open window**—The **Quick Start** window is an example of an open window.
6. **Status bar**—Check this status bar to see the status of operations MATRIXx is performing. The status bar appears only when an application is open.

MATRIXx operates in multiple document interface (MDI) mode. In MDI mode, all application windows appear within a single MATRIXx window. For example, if you open two SystemBuild models, each model appears in a separate Editor window, but both Editor windows appear inside the larger MATRIXx window. To displays a list of all open windows, select **View»Windows**.

Displaying and Configuring Views

The views in Figure 1-2 are docked to the left side of the MATRIXx window. A docked view is a view that is attached to the side of the MATRIXx window.

Notice the pair of icons, shown at left, at the top-right corner of each view. Clicking the left icon undocks the view. After you undock a view, you can move the view anywhere on the screen, including outside the MATRIXx window.

Undocked views float on top of the MATRIXx window, so you can rearrange views as necessary. To dock the view in a different location, move the view against the top, left, bottom, or right edge of the MATRIXx window.

Clicking the right icon closes the view. You can open the view again by using the **View** menu or by right-clicking the title bar of another view.

You can resize docked and undocked views. To resize a view, move the mouse towards the top, left, or right borders until the cursor changes to the resizing handles. Then, click and drag the border to the size you want.

Using the Work Area

All application windows appear in the work area. For example, when you launch Xmath, the **Xmath Commands** window appears in the work area.

Figure 1-2 shows the **Quick Start** window open in the work area. When you move an application window outside the boundaries of the work area, scrollbars appear that you use to scroll the work area.

You can move, resize, minimize, and maximize applications. When you maximize an application, the minimize/restore/close buttons appear in the upper-right corner of the MATRIXx window, just below the minimize/restore/close buttons of the MATRIXx window itself.

Figure 1-3 shows this behavior.

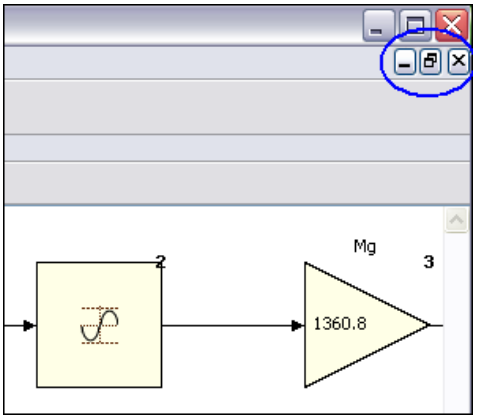

Figure 1-3. A Maximized Application

Figure 1-3 shows an Editor window that is maximized. Notice the minimize/restore/close buttons appear below the MATRIXx buttons.

Configuring MATRIXx

Select **Tools»Options** to launch the **Options** dialog box. You use this dialog box to configure the appearance and behavior of MATRIXx and the installed MATRIXx products, such as Xmath and SystemBuild. For information about these options, click the **?** button in the title bar of this dialog box.

Displaying Application Messages

MATRIXx applications return messages indicating the status of many operations, such as loading or saving files. To see these messages, select **View»Messages** to display the **Messages** view.

Quick Start

This section provides information about getting started quickly with MATRIXx.

Using the Quick Start Window

By default, MATRIXx displays the **Quick Start** window upon launch. You can see this window in Figure 1-2.

This window provides quick access to common tasks, such as creating an empty project, opening an example SystemBuild model, or reading the MATRIXx documentation.

Each section of the window has several panes. For example, the **Resources** section contains the **Help and Documentation**, **Demos**, and **Technical Support** panes. Click a pane to view the contents of that pane.

Bypassing the Project System

MATRIXx features a project system that you use to organize SystemBuild models, Xmath scripts, and so on. For information about this system, refer to Chapter 2, *Projects and Solutions*.

However, you can load models and scripts from previous MATRIXx versions without configuring a project. This section provides information about using SystemBuild and Xmath without using the project system.

Opening SystemBuild Models from Previous Versions

MATRIXx supports SystemBuild models saved in MATRIXx 5.0 or later. Complete the following steps to open a SystemBuild model.

1. Select **File»Open»SystemBuild Model**. MATRIXx prompts you for a model to load. After you select the model file, MATRIXx places this model file in an empty project.

2. Locate the **Solution Explorer** view in the MATRIXx window. If you do not see this view, select **View»Solution Explorer** to display this view.

   By default, this view displays the **Projects** pane, which shows the hierarchy of the current project. Each project has a **Diagrams** folder, which contains SuperBlocks and State Transition Diagrams (STDs) associated with that project.

   You can display DataStores and UserTypes by clicking the **Data Dictionaries** pane in the **Solution Explorer** view.
3. Use the **Solution Explorer** view to navigate to the appropriate model file.
4. Double-click a SuperBlock or STD to launch the Editor window for that diagram.
5. Use the **Blocks** view to place blocks on the block diagram. If you do not see this view, select **View»Blocks** to display this view.

For more information about using SystemBuild, refer to Chapter 4, *SystemBuild*.

Using Xmath

To launch the **Xmath Commands** window, select **Tools»Open Main Xmath Window**. Open MathScripts by selecting **File»Open Script**.

For more information about using Xmath, refer to Chapter 3, *Xmath*.

2

Projects and Solutions

In MATRIXx, you use projects and solutions to organize groups of related files. A project is a collection of files, such as Xmath scripts, SystemBuild diagrams, Microsoft Word documents, HTML files, and so on. A diagram is a SuperBlock or a State Transition Diagram (STD).

A solution is a collection of projects. You use the **Solution Explorer** view to show the structure of a solution and any projects it contains.

If you are using the project system, you typically complete the following steps:

1. Open a solution.
2. Create a new project inside that solution.
3. Configure the project.
4. Add and configure folders to organize SuperBlocks and STDs.
5. Add and edit diagrams.
6. Save the project and the solution.

This chapter introduces you to basic project and solution functionality. However, this chapter does not describe adding and editing diagrams. For information about adding and editing diagrams, refer to Chapter 4, *SystemBuild*.

Introduction to Projects and Solutions

Figure 2-1 shows a sample solution that contains one project named **Discrete Cruise Control**.

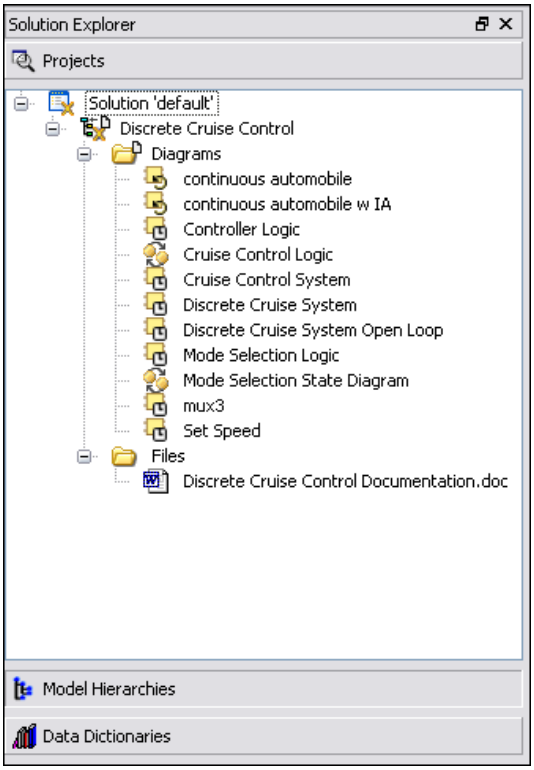

Figure 2-1. A Solution that Contains a Project

This project consists of several SuperBlocks, STDs, and a Microsoft Word document.

Solutions

You place all projects within a solution. In Figure 2-1, the solution is named **default**. Solutions provide a way to display and configure multiple projects together.

You can display only one solution at a time. You never have to create a solution because MATRIXx always loads one. You can also save and load solutions.

Savefiles and Folders

A savefile is a file on disk that contains MATRIXx data. A solution savefile contains the definition of the solution and has an extension of `.sln-mtx`. A project savefile contains the definition of the project and has an extension of `.proj-mtx`. In Figure 2-1, notice the **Discrete Cruise Control** project icon has a small file icon in the upper-right corner. This file icon indicates the project is configured and associated with a savefile.

A diagram savefile contains SuperBlock and/or STD data and has an extension of `.sbd-mtx`. The following section provides more information about diagrams and savefiles.

Diagrams Folder

Within a project, the **Diagrams** folder contains all the SystemBuild diagrams of the project. This folder and its subfolders are called *diagram folders*. All diagrams are contained in a diagram folder.

You save SuperBlocks and STDs by associating the diagram folder with a `.sbd-mtx` savefile. After you update a diagram in this folder, the diagram data is stored in this savefile.

For example, refer to Figure 2-1. In this figure, notice the **Diagrams** folder has a small file icon in the upper-right corner. This file icon indicates the **Diagrams** folder is associated with a single savefile. This setting means all diagrams in this folder are stored in a single savefile. You can also choose to save each diagram into a separate savefile.

To organize the project better, you can create subfolders under the top-level **Diagrams** folder. For example, you can put continuous SuperBlocks into a folder separate from discrete SuperBlocks. You can configure each subfolder to have a separate savefile or to use the savefile of its parent folder. During project configuration, you can specify the default savefile behavior of subfolders. For information about configuring a project, refer to the *Configuring the Project* section of this chapter.

For example, if you created separate subfolders for continuous and discrete SuperBlocks, you could configure each subfolder to have a separate savefile. You might name these savefiles `continuous.sbd-mtx` and `discrete.sbd-mtx`. In this way, you can divide a single project into multiple parts.

Note Project folders and subfolders are virtual and exist only within the `.proj-mtx` savefile. These folders have no relation to hard drive directories.

Files Folder

You use the **Files** folder to include non-diagram information with the project. For example, you could add documentation, Xmath scripts, and graphics to this folder. MATRIXx does not store these files within the `.proj-mtx` file. MATRIXx stores only references to these files. Double-clicking the icon from the **Solution Explorer** view launches the default editor for that file type. Similar to the **Diagrams** folder, you can create subfolders under the **Files** folder.

Note The concept of savefiles does not apply to **Files** folders.

Configured Projects

Configuring a project involves associating a savefile with the project and setting other options. National Instruments recommends you configure a project to take full advantage of the project system in MATRIXx. Also, you cannot save a project until you configure it. If you try to save an unconfigured project, MATRIXx might prompt you to configure the project.

Project Panes

The **Solution Explorer** view always displays the **Projects** pane. After you create or open a project, this view also displays the **Model Hierarchies** and **Data Dictionaries** panes. Each pane corresponds to a different way of displaying the project organization and information.

Note A data dictionary is a collection of UserTypes and DataStores. For information about these items, refer to the *SystemBuild User Guide,* available by selecting **Help» MATRIXx Bookshelf**.

Learning the Project System

This section describes how to create, configure, modify, save, and load a project and solution.

Before you begin, if you do not see the **Solution Explorer** view, select **View»Solution Explorer** to display this view.

Figure 2-2 shows the empty solution that MATRIXx loads at startup.

Figure 2-2. A Solution with No Projects

Creating a New Project

Complete the following steps to create a project.

1. Select **File»New»Project** to launch the **Add New Project** dialog box. You can also right-click the **Solution** item and select **New Project**.

2. Use the **Add New Project** dialog box to specify a project name of `Demo MtxProject`. Then, specify the location on disk and click **OK** to launch the **Configure MATRIXx Project** wizard, shown in Figure 2-3.

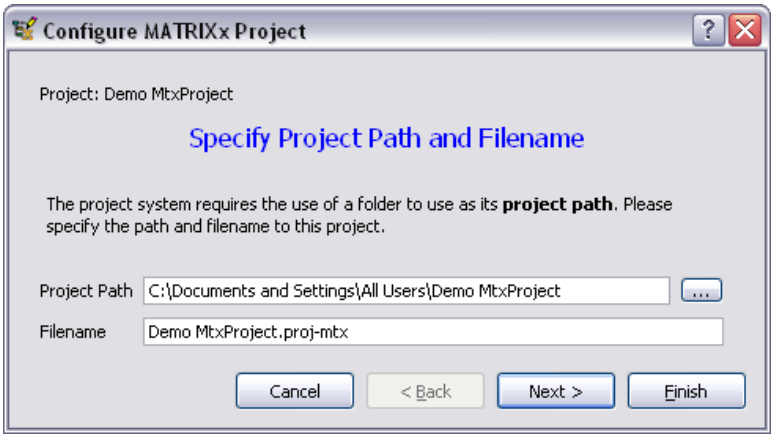

Figure 2-3. Configuring a New Project

The next step is configuring the project. The following section provides information about this process.

Configuring the Project

Configuring a project involves specifying certain options for the project and associating a savefile with the project.

Note National Instruments recommends you configure projects, because you cannot save unconfigured projects.

Complete the following steps to configure the project:

1. On the **Specify Project Path and Filename** page, specify the path and name of the `.proj-mtx` project savefile.
2. Click **Next**.
3. On the **Specify Default Save Option for Diagram Folders** page, select the **Inherited configuration of parent folder** option.

   This option specifies that, by default, diagram subfolders inherit the savefile of the parent folder. For information about savefiles, refer to the *Savefiles and Folders* section of this chapter.
4. Click **Next**.

5. On the **Specify Startup and Shutdown Scripts** page, ensure there are no checkmarks in any checkboxes. This project does not require any Xmath scripts to execute at startup or shutdown.
6. Click **Next**.
7. On the **Summary** page, review the options and click **Finish**.

After you click **Finish**, MATRIXx creates the project. The **Solution Explorer** view now resembles Figure 2-4.

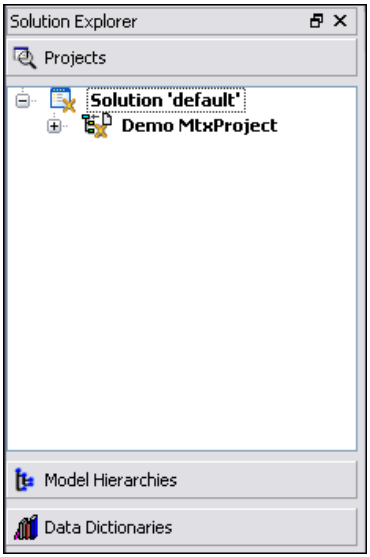

Figure 2-4. A Solution that Contains One Project

Figure 2-4 shows the following changes compared to Figure 2-2.

- The solution appears in bold font. Bold font indicates that the solution, project, or folder contains unsaved items. In this case, the solution has an unsaved project, which is also displayed in bold font.
- The **Solution Explorer** view now displays the **Model Hierarchies** and **Data Dictionaries** panes.

In Figure 2-4, also notice that the **Demo MtxProject** icon has a small file icon in the upper-right corner. This icon indicates the project is configured and associated with a savefile. Unconfigured projects do not have this small file icon.

Creating a new project also displays an **Xmath Commands** window automatically. You use this window to interact with Xmath.

For now, close or minimize this window. For information about Xmath, refer to Chapter 3, *Xmath*.

Adding a File to the Project

Complete the following steps to add a file to the project.

1. Expand the **Demo MtxProject** item in the **Solution Explorer** view. Notice the **Diagrams** and **Files** folders.
2. Right-click the **Files** folder and select **Add Existing Files**.
3. Navigate to `matrixx\mx_xx\version\license.txt`, where *xx* is the MATRIXx version number, and click **OK**. The **Files** folder now appears in bold.
4. Expand the **Files** folder. The **Solution Explorer** view now resembles Figure 2-5.

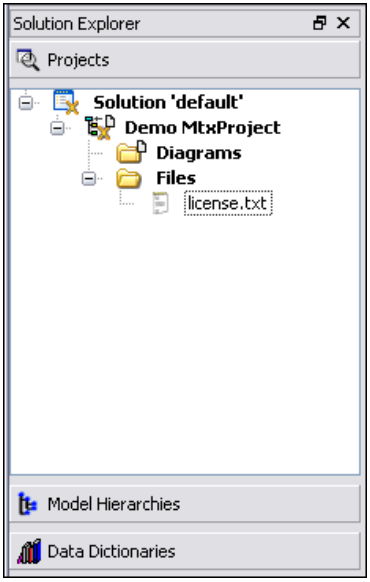

Figure 2-5. A Project with File Added

Adding a Diagram Subfolder to the Project

As described in the *Savefiles and Folders* section of this chapter, you can add subfolders under the **Diagrams** folder. You use these subfolders to organize diagrams within a project. You can also configure how these subfolders save diagrams.

Note You can add diagram subfolders only to projects that you have configured.

Complete the following steps to add a subfolder to this project.

1. Right-click the **Diagrams** folder and select **Add Diagram Folder**.
2. Expand the **Diagrams** folder to display the new subfolder, which is named New Diagrams, in the **Solution Explorer** view.
3. Right-click the New Diagrams subfolder and select **Configure Folder** to launch the **Folder Configuration** dialog box, shown in Figure 2-6.

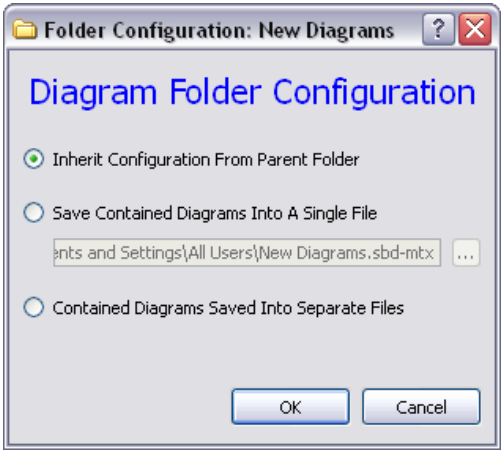

Figure 2-6. The Folder Configuration Dialog Box

You use this dialog box to specify how the subfolder uses savefiles.

Recall that earlier in this tutorial you specified the **Inherit configuration of parent folder** option. Therefore, this option is selected already in the **Folder Configuration** dialog box. Notice that you can choose other ways to handle saved diagrams.

Tip For more information about this dialog box, click the **?** button in the title bar of this dialog box. Most dialog boxes in MATRIXx have this button, which launches the *MATRIXx Help* for that dialog box. For information about the *MATRIXx Help*, refer to Chapter 7, *MATRIXx Help Resources*.

For now, do not make any changes in this dialog box.

4. Click **Cancel** to return to the **Solution Explorer** view.
5. Right-click the New Diagrams folder and select **Rename**.
6. Enter `Demo Subfolder` as the new subfolder name.

7. Press <Enter> to accept the new name.

Saving the Project

To save the project, right-click the **Demo MtxProject** item and select **Save**. MATRIXx saves the project information to the `.proj-mtx` file you specified in the *Configuring the Project* section of this chapter.

The **Demo MtxProject** item now displays in regular font, which indicates the project does not have any unsaved changes.

Note Saving a project does not save pending changes in open Editor windows. To save changes to diagrams, you must update those diagrams in their individual editor windows before saving the project. For more information about this process, refer to the *Adding the SuperBlock to the Project* section of Chapter 4, *SystemBuild*.

Saving the Solution

To save the solution, right-click the **Solution** item and select **Save**. MATRIXx prompts you for a location to save the solution as a `.sln-mtx` file. Save this file to a convenient location on disk. This location can be a different folder than where you saved the project savefile.

After you save the file, all items in the **Solution Explorer** view appear in regular font.

Note Saving a solution also saves any unsaved projects in that solution.

Renaming the Solution

Right-click the **Solution** item and select **Rename**. Enter `Sample Solution` as the new name. Press <Enter> to accept the change. Notice the solution appears in bold font, which indicates unsaved changes.

Save the solution again. The **Solution Explorer** view now resembles Figure 2-7.

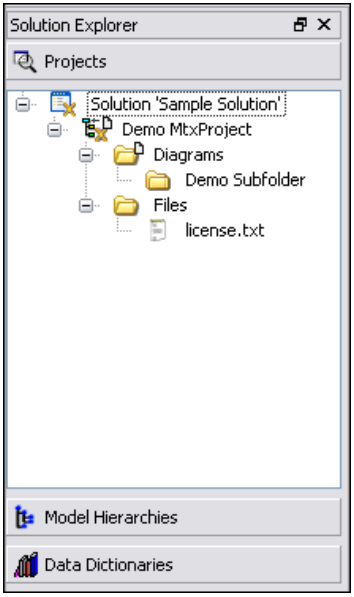

Figure 2-7. Sample Solution with All Items Saved

Adding a File Subfolder to the Project

To add a subfolder to the **Files** folder, right-click the **Files** folder and select **Add Folder**. MATRIXx displays a New Folder item in the project tree. Rename the folder to `File Subfolder` by clicking it in the **Solution Explorer** view and pressing <F2>.

Save the solution.

Where to Go from Here

This chapter introduced you to the project system and showed you how to create, modify, and save a project and solution. Chapter 4, *SystemBuild*, contains information about creating, editing, and managing SuperBlocks within the project system.

3

Xmath

Xmath provides tools for mathematical analysis. You can use Xmath to create, store, plot, and explore data. You can also define new functions, commands, and objects and also link in externally-compiled C or FORTRAN code.

This chapter introduces you to basic Xmath functionality. For a more detailed discussion of Xmath, refer to the *Xmath User Guide*, available by selecting **Help»MATRIXx Bookshelf**.

Introduction to Xmath

This section describes the basic capabilities of Xmath.

MathScript Functions and Commands

MathScript, the programming language of Xmath, defines statements, constructs, punctuation, functions, commands, and objects. You can use MathScript to create custom functions and commands, which are called scripts. Xmath includes over 700 pre-defined MathScript functions and commands, including mathematical functions and filter design functions. You can install addons for Xmath that contain additional MathScript functions and commands to address special uses, such as control design.

Note MathScript functions have the file extension `.msf`, MathScript commands have the file extension `.msc`, and MathScript objects have the file extension `.mso`.

MathScript incorporates an object-oriented structure that enables efficient numerical handling, including the overloading of operators. Xmath's hierarchical objects greatly reduce the amount of programming necessary to check data characteristics.

Xmath also provides plotting capabilities, one with an interactive graphics display, and the other integrated with a programmable GUI facility.

Graphical User Interface

Xmath includes a fully programmable graphical user interface (PGUI or GUI). Use this PGUI to create and manipulate windows, dialog boxes, and other user interface tools. For more information about the PGUI, refer to the *Xmath User Guide*.

External Code

In addition to the MathScript programming language, you can use Xmath to call external C and FORTRAN routines, and vice versa. The Linked External (LNX) facility uses an interprocess communication (IPC) mechanism to communicate between external routines and Xmath. These routines run as a separate process from Xmath. You can modify, debug, and recompile an external routine without leaving Xmath.

The User-Callable Interface (UCI) allows a C program to invoke Xmath as a computational engine. For more information about LNXs and UCIs, refer to the *Xmath User Guide*.

Data Handling

You can use MathScript 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.

Xmath Environment

This section provides information about the Xmath environment.

Xmath Commands Window

The **Xmath Commands** window is where you enter commands and view the results of these commands. Figure 3-1 shows the **Xmath Commands** window.

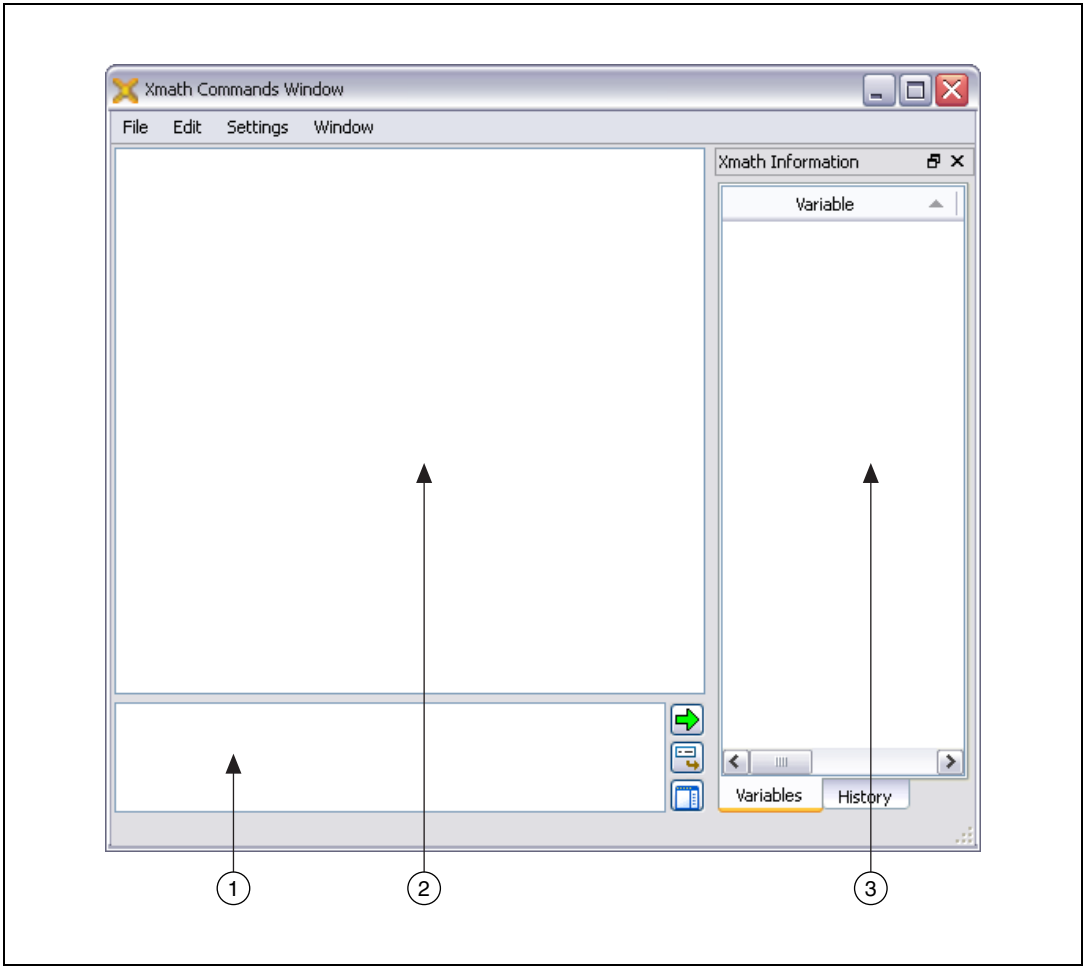

Figure 3-1. The Xmath Commands Window

The command area (1) is where you enter functions and commands. MATRIXx displays the results of these commands in the log area (2). You use the **Xmath Information** view (3) to display partitions, variable information, and a history of commands you entered during the current session.

The buttons to the right of the command area, listed from top to bottom, perform the following functions:

- **Evaluate**—Executes the current command(s) in the command area. You can also press <Enter> to perform this action.
- **Multiline**—Toggles multiline mode on and off. You can also press <Shift-Enter> to perform this action. For information about multiline mode, refer to the *Executing Multiple Commands at Once* section of this chapter.
- **Information**—Toggles the **Xmath Information** view on and off.

Environment Variables

Environment variables provide a quick way of referencing specific hard drive directories. MATRIXx defines several environment variables. For example, `%SYSBLD%` refers to the directory in which SystemBuild is located. These environment variables are known only within the MATRIXx application.

For information about environment variables, refer to the *Xmath User Guide*.

Working Directory

The working directory is the default location on the hard drive where Xmath saves and loads data. To display the current working directory, enter `show directory` in the **Xmath Commands** window. To change the current working directory, enter `set directory "dir"`, where *dir* is the new working directory. You can also select **Settings»Directory**.

After you load a configured project, MATRIXx automatically sets the project directory as the current working directory.

Lookup Path

The lookup path, or path, is the set of directories Xmath searches for functions, commands, and scripts. Xmath displays an error if the function or command is not in any of these directories.

For example, if you create a custom script `foo.msc` and save this script to `C:\bar\`, Xmath returns an error if you try to execute the script by simply calling `foo( )`. This error occurs because `C:\bar\` is not in the lookup path, so Xmath cannot find `foo.msc`. To avoid this error, you must add `C:\bar\` to the lookup path.

To display or modify the lookup path, use the **Path Editor** dialog box, availably by selecting **Settings»Path**.

Partitions

Partitions are named, non-hierarchical directories that contain Xmath data. The default partition is `main`. You can address data in other partitions by using the `.` symbol to separate the variable and partition name. For example, the following command assigns a value of `3` to the variable named `foo` that is located in the `bar` partition.

```
bar.foo=3
```

If you are addressing a variable in the current partition, you do not need to specify the partition. You can perform the following operations related to partitions.

- **Display the current partition**—Enter `show partition` in the command area. You can also right-click in the **Xmath Information** view and select **Update**.
- **Create a partition**—Enter `new partition` *`name`*, where *`name`* is the name of the new partition. You can also right-click in the **Xmath Information** view and select **Create Partition**.
- **Switch partitions**—Enter `set partition` *`name`*, where *`name`* is the partition you want to switch to. You can also right-click the partition in the **Xmath Information** view and select **Set as Current**.
- **Delete a partition**—Enter `delete` *`name`*, where *`name`* is the partition you want to delete. You can also right-click the partition in the **Xmath Information** view and select **Delete**.

Learning Xmath

This section provides information about accomplishing basic tasks in Xmath. You can use several different methods to accomplish many of these tasks. However, to simplify the presentation, these sections describe only one method.

Launching Xmath

To launch Xmath, select **Tools»Open Main Xmath Window**. The **Xmath Commands** window appears.

Entering and Executing MathScript Commands

Execute MathScript commands by entering text in the command area and pressing <Enter>. You view the results of these commands in the log area.

For example, enter `a = 3` in the command area, and press <Enter>. Xmath displays the following text in the log area.

```
a (a scalar) =   3
```

Executing Multiple Commands at Once

The command area has two command modes: single-line and multiline. Use multiline mode to execute multiple commands at once. In single-line mode, pressing <Enter> executes the command you entered. In multiline mode, pressing <Enter> creates a new line in the command area.

To enter multiline mode, click **Multiline Mode**, shown at left. This button is to the right of the command area.

For example, complete the following steps to execute three commands at once.

1. Press **Multiline Mode**. The command area now is in multiline mode. Notice the button changes to reflect the new mode.
2. Click in the command area, and type `a=-5`.
3. Press <Enter> to create a new line.
4. Type `b=3`.
5. Press <Enter> to create a new line.
6. Type `c=abs(a+b)?`.
7. Press **Multiline Mode** to switch to single-line mode.
8. Click in the command area and press <Enter> to execute all three commands at once. The log area displays the following output:

```
> a = -5
> b = 3
> c = abs(a+b)?
a (a scalar) =   -5
b (a scalar) =   3
c (a scalar) =   2
```

Note The `?` symbol forces the log area or **Xmath Graphics** window to display the command output. This symbol also separates multiple commands you enter on the same line.

Accessing Xmath Help

You can access help on any Xmath function or command by entering `help` *`function`* in the command area. For example, the previous section uses the `abs( )` function. To receive more information about the inputs, outputs, and operation of this function, enter `help abs` in the command area. The *MATRIXx Help* appears and automatically displays the help topic for the `abs( )` function.

Quitting Xmath

You can quit Xmath by using one of the following methods:

- Enter `quit` in the command area.
- Select **File»Quit** from the **Xmath Commands** window.
- Close the **Xmath Commands** window.

If you quit while working with unsaved Xmath data, Xmath launches a dialog box that prompts you to save the workspace before quitting. The workspace is the contents of all Xmath partitions, including defined variables and objects. If you click **Save** in this dialog box, Xmath saves the workspace to the file `save.xmd` in the working directory.

To stop an Xmath operation, press <Ctrl-Break>. However, this key combination cannot stop a process that is communicating with the operating system, such as loading or saving a file or displaying a window.

To abort Xmath if the program stops responding, follow the operating system procedure for handling non-responsive programs.

Creating Data and Objects

The first step in mathematical analysis is usually creating data on which you operate. Enter the following commands in the command area to create MathScript data. Press <Enter> after each command to display the output in the log area.

```
a=[1,2,2^2,3^3]

b=[1:.1:5]

c=sin(b)
```

These commands create two different types of numeric objects. To display the object type of the variable `a`, enter `whatis a`. The following text appears in the log area.

```
a is a row vector
```

Entering `whatis b` displays the following text:

```
b is a regular vector
```

Note For information about vector objects in Xmath, enter `help vector` in the command area.

A vector is a numeric class. Nonnumeric objects, also known as complex objects, are strings or combinations of strings and numeric objects. Polynomial equations fall into this category. Enter the following commands to create two polynomial equations.

```
d=makepoly(a,"d")

e=polynomial(1:3,"d")
```

Note For more information about these commands, enter `help makepoly` or `help polynomial`.

Another type of object you can create in Xmath is a system model. Enter the following command to create a system model in transfer function form.

```
sys=system(d,e)
```

Some MathScript functions accept only a certain type object and return another type object. For example, the `freq( )` function accepts a system and returns a parameter-dependent matrix (PDM). A PDM object stores matrices in relation to an independent parameter or domain. This independent parameter is typically a vector of time or frequencies.

For example, enter the following command to analyze the frequency response of the `sys` system.

```
f=freq(sys,b);
```

In the previous example, the `freq( )` function uses the `b` vector as the time points at which to calculate the frequency response of the `sys` system.

Note The `;` symbol suppresses output to the log area. Like the `?` symbol, the `;` symbol separates multiple commands entered on the same line.

Enter the following command to let the `freq( )` function determine the points at which to calculate the frequency response of the system.

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

The `f` and `g` variables are PDMs that contain the frequency responses of the `sys` system model at different time points. You can plot these two variables to compare them. The following section provides information about plotting variables.

Plotting and Printing Data

Enter the following command to plot `f`.

```
plot(f)
```

After you press <Enter>, Xmath displays the **Xmath Graphics** window that contains a plot of the values `f` contains. Because you did not specify an output of the plot, Xmath saves the plot output to a variable named `ans`. Notice the title bar of the **Xmath Graphics** window displays **main.ans**, which denotes the variable `ans` in the `main` partition.

To save the plot as a named object, enter the following command:

```
graph1=plot(f)
```

This command assigns the plot to an object `graph1`.

Enter the following command to plot `f` and `g` in the same **Xmath Graphics** window for comparison purposes.

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

The previous example creates two graph objects `graph1` and `graph2`, each with different frequency response information. The `rows` keyword specifies that Xmath displays an **Xmath Graphics** window with room for two plots. The `row` keyword specifies that `graph2` appears on the second row of this window.

Note In addition to the `plot( )` function, Xmath provides the `uiPlot( )` and `plot2d( )` commands, which feature advanced capabilities. Enter `help uiPlot` or `help plot2d` for information about these commands.

Binding Graphs to Variables

If you plot data but forget to save the plot to a variable, you can later bind the graph to a variable. For example, enter the following command to calculate the Nyquist response of the `sys` system model.

```
nyquist(sys)?
```

This command displays the Nyquist response data in the log area as well as the **Xmath Graphics** window. However, this data is not saved to a variable. Notice the title bar of the **Xmath Graphics** window reads **Unbound Graph Object**.

Save the Nyquist response data to the variable `graph3` by using one of the following methods:

- Display the **Xmath Graphics** window, select **File»Bind to Variable**, specify the variable name `graph3`, and click the **OK** button.
- Display the **Xmath Commands** window and enter `graph3=plot( )` in the command area.

After you bind the graph to `graph3`, the title bar of the **Xmath Graphics** window displays **main.graph3**. This display indicates the binding was successful.

Printing Graphs

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

- Display the **Xmath Graphics** window and select **File»Print**.
- If the printer is set as an environment variable, display the **Xmath Commands** window and enter `hardcopy {color=0}` in the command area. The option `color=0` ensures the printout is in black and white, whereas the default is a color printout. If the printer is not set as an environment variable, Xmath displays an error.

Note To use the `hardcopy` command, you must define the environment variable `%XMATH_PRINT%`. For more information about viewing and setting environment variables, refer to the *Xmath User Guide*.

You can also use the `hardcopy` command to save graphics to a PostScript (`.ps`) file. Enter the following command to save the `graph3` object as a PostScript file.

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

For information about PDMs or the `plot( )` command, enter `help pdm` or `help plot`, respectively. For information about plotting data and using the **Xmath Graphics** window, refer to Chapter 4, *Graphics,* of the *Xmath User Guide*. Access this guide by selecting **Help»MATRIXx Bookshelf** and clicking the *Xmath User Guide* link.

Saving and Loading Data

Xmath provides commands to view, manipulate, save, load, and print all or some of the objects you create. For example, enter `who` to display a list of the variables you created in the previous sections. Xmath displays the names and dimensions of the `a`, `b`, `c`, `d`, `e`, `f`, `g`, `sys`, `graph1`, `graph2`, and `ans` variables. For information about these dimensions, enter `help who` in the command area.

You can also use the **Xmath Information** view to display this list of variables. Complete the following steps to show this list.

1. If you do not see the **Xmath Information** view, click the **Information** button, located to the right of the command area, to display this view.
2. Click the **Variables** tab at the bottom of the view.
3. Right-click in the view and select **Update**. MATRIXx displays a list of all partitions and variables.
4. Expand the **main** partition in this list.

This view now resembles Figure 3-2.

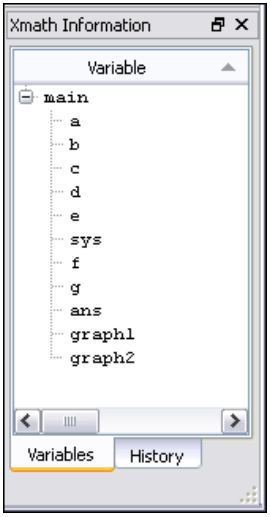

Figure 3-2. The Xmath Information View with All Variables Displayed

You can resize this view to display more information. Complete the following steps to resize this view.

1. Move the mouse to the left of the **Xmath Information** view until the cursor changes to a splitter bar.
2. Drag the border of the view to the left. The view expands to display additional columns, including **Type**, **Value**, **Dimension**, and **Comment**. You can click a column heading to sort the view by that column.

Saving Data

To save these variables and their values to an Xmath data (`.xmd`) file, enter the `save` command. Xmath saves all data to the file `save.xmd`.

You can customize the filename by entering a different one. For example, entering `save "try.xmd"` saves all data to a file named `try.xmd`.

Note If you do not specify a directory when using the `save` command, Xmath saves data in the current working directory. For information about the working directory, refer to the *Working Directory* section of this chapter.

You can also right-click in the **Xmath Information** view and select **Save All**.

You can also use wildcards to save only some variables to a file. To see this behavior, enter the following command:

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

The previous example saves `g`, `graph1`, `graph2`, and `sys` to the file `try_2.xmd`. The `*` character is a wildcard. For more information about wildcards, enter `help wildcards`.

In the previous examples, Xmath saved `try.xmd` and `try_2.xmd` in the working directory. To display the path to the working directory, enter the following command:

```
show directory
```

Use the command `oscmd( )` command to send a command to the operating system. To display the two `.xmd` files you created, enter the following command:

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

Now that you have confirmed these two files exist, you can delete the variables currently in memory in Xmath. To delete these variables, enter the following command:

```
delete *
```

You can also right-click in the **Xmath Information** view and select **Delete All**. MATRIXx prompts you to delete only variables or both partitions and variables. Click **Variables Only** to delete only the variables.

To confirm that MATRIXx deleted the variables, enter the following command:

```
who
```

This command displays nothing in the log area. However, notice that the **Xmath Information** view still displays the variables. You must refresh this view to display any changes you make to variables or partitions. To refresh this view, right-click in this view and select **Update**.

Loading Data

To load the data you saved in the previous section, enter the following command:

```
load "try_2"
```

This command loads the data from the `try_2.xmd` file in the working directory. You can also select **File»Load** from the **Xmath Commands** window pull-down menu.

To confirm the data loaded properly, enter the following command:

```
who
```

Remember that `try_2.xmd` contains only the `g`, `graph1`, `graph2`, and `sys` variables.

Creating Functions, Commands, and Objects

This section provides information about creating and debugging custom scripts.

Creating Scripts

To create a new script, select **File»New Script** from the **Xmath Commands** window. Enter the script shown in Example 3-1.

Note This file is also available in the `matrixx\mx_xx\xmath\examples\cdown\` directory, where *xx* is the version of MATRIXx you installed.

Example 3-1 cdown.msf

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

if is(c,{integer, min=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 this script by selecting **File»Save**. Name the file `cdown.msf` and place the script in the current working directory, or any directory in the path, and return to Xmath. For information about the working directory or path, refer to the *Working Directory* section of this chapter or the *Lookup Path* section of this chapter, respectively.

Running Scripts

To test this function, enter the following command in the command area.

```
cdown(5)
```

The log area displays the following text:

```
************

SQRT(5) = 2.23607

SQRT(4) = 2

SQRT(3) = 1.73205

SQRT(2) = 1.41421

SQRT(1) = 1

ans (a row vector) =   2.23607   2   1.73205   1.41421
  1
```

Notice the output of this function is assigned to a variable `ans`. This variable is the default variable when you do not specify an output variable. For example, you can specify an output variable by entering the following command:

```
foo=cdown(5)
```

Test the function with an invalid argument by entering the following command:

```
cdown(-5)
```

Xmath displays the error message you provided in the script.

Debugging Scripts

Use the **Xmath Debugger** window to debug MathScript files. This section provides information about using the **Xmath Debugger** window.

Entering Debug Mode

Xmath enters debug mode in any of the following three situations:

- You enter the command `debug` *`script_name`*, where *`script_name`* is the name of the script you want to debug.
- You execute a script that contains a syntax error, such as a missing brace or parentheses.
- You execute a script that encounters a run-time error, which occurs when Xmath cannot process a script instruction.

  For example, the following statement uses correct syntax but causes a run-time error. The + operator cannot accept an integer and a string.

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

Normally, when Xmath detects an error in the script, Xmath automatically displays the error in the **Xmath Debugger** window and sets the interpreter to debug mode. To prevent the interpreter from going into debug mode, enter the following command:

```
set debugonerror off
```

Xmath still displays the **Xmath Debugger** window, but the interpreter does not go into debugging mode.

Interactive vs. Textual Debugging

You can debug scripts interactively by clicking buttons in the **Xmath Debugger** window and textually by entering commands in the command area. These two methods are equivalent. For example, clicking the **End Debug** button is equivalent to entering `abort` in the command area.

Starting the Debugger

To enter debug mode, enter the following command:

```
debug cdown
```

This command sets a breakpoint on the first executable line of the `cdown` script. A breakpoint is a point at which Xmath pauses the script execution. The next time you run the `cdown` script, the **Xmath Debugger** window appears.

To test this behavior, enter `cdown(2)` in the command area. MATRIXx launches the **Xmath Debugger** window, shown in Figure 3-3.

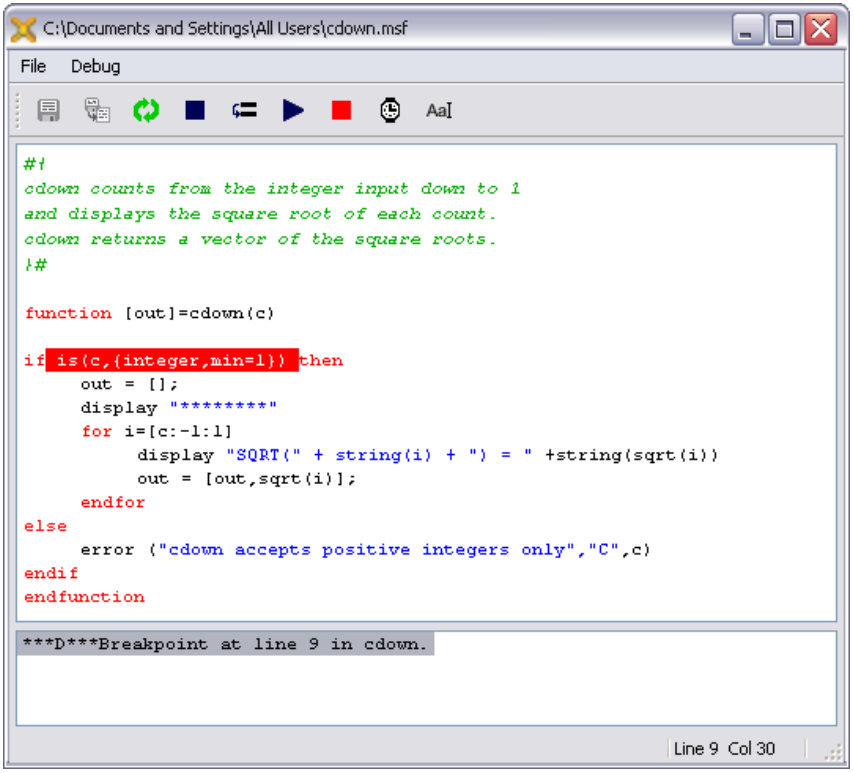

Figure 3-3. The Xmath Debugger Window

If you switch back to the **Xmath Commands** window, notice the words **DEBUG MODE** appear in the status bar at the bottom left of this window, along with the name of the script you are debugging.

Stepping through the Script

Display the **Xmath Debugger** window and click the **Next** button, shown at left, three times. You can also enter `next` three times in the command area of the **Xmath Commands** window.

MATRIXx highlights the following line:

```
for i=[c:-1:1]
```

Watching a Variable

With the line **`for i=[c:-1:1]`** highlighted, click the **Set Watch** button, shown at left, and enter `i` in the text box that appears. Then click OK.

This action sets a watch on the variable `i`. You can also enter the following command:

```
set watch i
```

Click the **Next** button. Notice the following message appears in the **Xmath Debugger** window.

```
Value of watched variable i has been modified.
```

Display the **Xmath Commands** window and enter the following commands using multiline mode:

```
who
i?
```

Note For information about multiline mode, refer to the *Executing Multiple Commands at Once* section of this chapter.

The `who` command lists the names and dimensions of all variables in memory. The `i` command displays the current value of the variable `i`.

Enter `remove watch i` to stop watching `i`.

Stopping the Debugger

Click the **Next** button until you reach the end of the script. You can also click the **Go** button, shown at left. Clicking this button runs the script until the next breakpoint, watch, or error, or until the script finishes.

When you reach the end of the script, Xmath automatically exits debug mode. You can also click the **End Debug** button, shown at left, or enter `abort` in the command area.

Correcting Errors during Debugging

While you are debugging a script, you can click the **Allow Edit** button, shown at left, and fix script errors. You can then click **Save** to save the updated script and rerun the debugger on the updated script. You can also click **Revert** to erase all changes you made since clicking the **Allow Edit** button.

Where to Go from Here

This chapter introduced you to Xmath and some basic functionality of the software. The following documents contain more information about Xmath and programming in MathScript.

- The *MATRIXx Help* contains topics about using Xmath and MathScript. Access these topics by launching Xmath and entering `help xmath` or `help mathscript`.
- The *Xmath User Guide* contains detailed information about concepts such as objects, plotting, and data handling. This user guide also contains information about accomplishing specific tasks using Xmath and MathScript. Access this user guide by selecting **Help»MATRIXx Bookshelf**.
- Chapter 2 of the *Xmath User Guide* provides a more detailed tutorial of Xmath functionality.
- Xmath provides several interactive demonstrations that showcase certain functionality. Access these demos by launching Xmath and entering `demo` in the command area.

4

SystemBuild

SystemBuild is a graphical programming environment that you use to construct and simulate models of linear and nonlinear dynamic systems. You use the MATRIXx project system to create, edit, and manage SystemBuild models. The exercises in this chapter assume you are familiar with the project system. For more information about the project system, refer to Chapter 2, *Projects and Solutions*.

This chapter introduces you to basic SystemBuild functionality. For a more detailed discussion of SystemBuild, refer to the *SystemBuild User Guide*, available by selecting **Help»MATRIXx Bookshelf**.

Introduction to SystemBuild

This section introduces fundamental SystemBuild concepts and tools.

Models, SuperBlocks, and Blocks

A model is a collection of SuperBlocks, which you build by using blocks. SuperBlocks define the properties, such as the integration algorithm and timing information, that SystemBuild uses to simulate the model. Blocks receive input signals, perform calculations based on those input signals, and produce output signals based on this calculation. For example, the Integrator block integrates an input signal. You build a dynamic system model by placing blocks inside a SuperBlock.

SuperBlocks are hierarchical; that is, you can place a SuperBlock within another SuperBlock to create a subsystem. Subsystems provide a way to modularize and encapsulate pieces of the model. By combining several blocks into a subsystem, you reduce the amount of space needed on the block diagram, making the simulation easier to navigate visually. Subsystems also are useful for validating, distributing, and reusing portions of the block diagram.

SystemBuild includes over 80 pre-defined blocks. Most of these blocks have user-definable properties that affect the calculation of the block. You can also create custom blocks.

To view detailed information about a block, launch Xmath and enter `help block_name` in the command area, where *block_name* is the name of the block whose help topic you want to view. To view a full list of SystemBuild blocks, enter `help blocks`.

Blocks View and Palettes

Use the **Blocks** view, available by selecting **View»Blocks**, to define a dynamic system model by using SystemBuild blocks. SystemBuild uses palettes to organize the blocks into groups. Figure 4-1 shows the **Blocks** view with the **Algebraic** palette displayed.

Figure 4-1. Blocks View with the Algebraic Palette Displayed

Block Diagram and the Editor Window

The block diagram is where you place all SuperBlocks and blocks. As you place blocks on the block diagram, you build the dynamic system model by connecting outputs from one block to inputs of other blocks. A group of connected blocks can collectively define a SuperBlock at any hierarchy level.

The SuperBlock Editor, also known as the Editor window, is what you use to edit the block diagram and connections. Figure 4-2 shows an example Editor window.

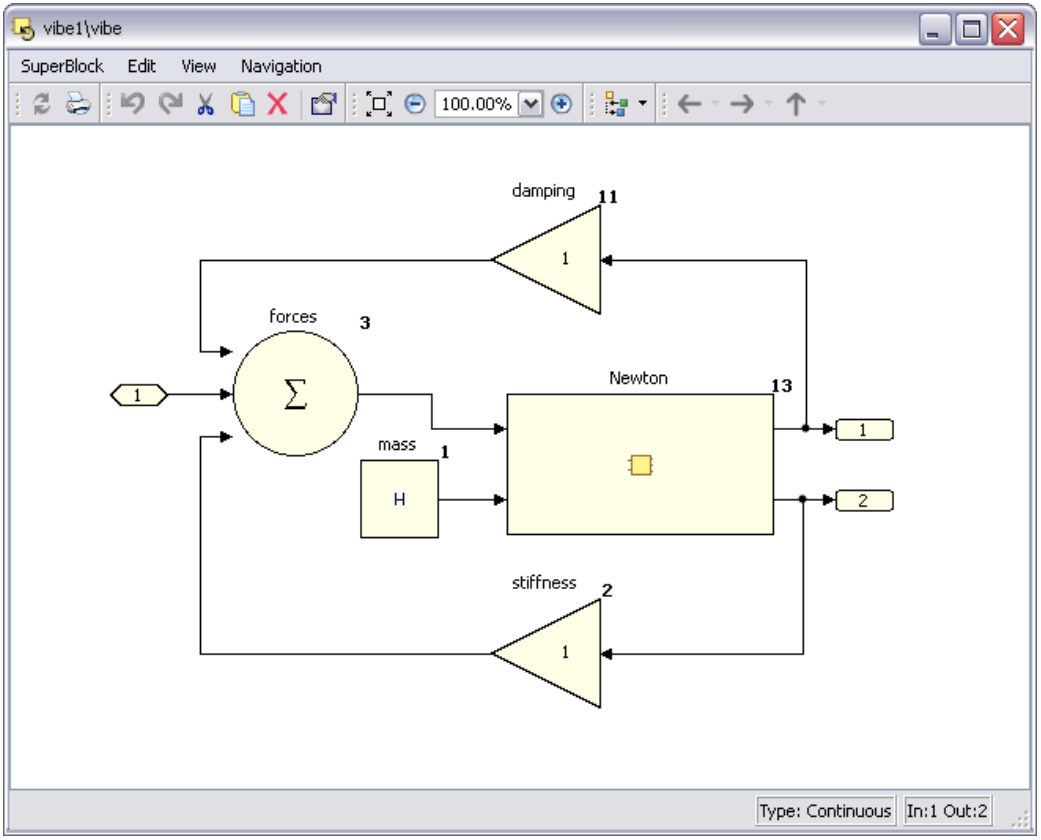

Figure 4-2. A SuperBlock in the SystemBuild Editor Window

Figure 4-2 shows a SuperBlock that contains several SystemBuild blocks and another SuperBlock, labeled `Newton`. In this way, Figure 4-2 shows you can encapsulate block diagram code using hierarchical levels of SuperBlocks.

You simulate a SuperBlock by specifying each of its input signals with respect to some time vector. For more information about simulating a SuperBlock, refer to the *SystemBuild Simulator* section of this chapter.

You can display multiple Editor windows simultaneously.

SystemBuild Tools

MATRIXx provides the following SystemBuild tools that you use to create and edit SuperBlocks.

- **Selection**—Use this tool to select, resize, move, and edit blocks and SuperBlocks.
- **Connection**—Use this tool to create connections between blocks and SuperBlocks.
- **Grab**—Use this tool to move the block diagram in any direction.

Figure 4-3 shows these tools, listed from top to bottom.

Figure 4-3. The SystemBuild Tools—Selection, Connection, and Grab

You can press <Tab> to cycle between these tools.

SystemBuild Simulator

Use the SystemBuild Simulator to simulate a block diagram model under conditions you define. You can define integration algorithms, data input methods, model timing, and other properties of the simulation. You can access and control the simulation by using the Editor window or by entering commands in the **Xmath Commands** window.

When you simulate a model in interactive mode, you can interact with the model and monitor the simulation outputs. You can also use interactive capabilities, such as executing blocks one at a time, to debug simulations.

During interactive simulation, you can change the values of some block parameters by using the Run-Time Variable Editor (RVE). For information about the RVE, enter `help rve` in the Xmath command area.

SystemBuild and Projects

The Editor window is always associated with a project. If you open a model without creating a project, MATRIXx automatically creates an unconfigured project. For more information about projects, refer to Chapter 2, *Projects and Solutions*.

Each project has a **Diagrams** folder. All SuperBlocks and State Transition Diagrams (STDs) reside in this folder or its subfolders.

When you configure a project, you choose how you want MATRIXx to save the SuperBlocks and STDs in the **Diagrams** folder. You can also configure each subfolder separately. For information on how SystemBuild saves diagrams, refer to the *Savefiles and Folders* section of Chapter 2, *Projects and Solutions*.

Data References

A data reference, also known as a % variable or parameter variable, is an Xmath variable that you use for the value of a SystemBuild block parameter. Using data references separates model data from the block diagram. This separation enables you to quickly change data values between simulations and to organize sets of data values.

For example, consider a Gain block on a block diagram. You can use the **Block Properties** dialog box in SystemBuild to configure the gain of this block. However, if you want to run the simulation three times with different gain values, you must open the Editor window and launch this dialog box two more times.

Alternatively, you can define a data reference *k* in SystemBuild. You then specify a value for *k* in Xmath before you run the simulation. Because you can easily change the value of *k* before each simulation, you can run the simulation repeatedly without ever opening the Editor window to configure the Gain block.

SystemBuild Access

SystemBuild Access (SBA) is a set of Xmath commands and functions that you can use to programmatically create, modify, and query any SystemBuild object. You can use SBA to operate on SuperBlocks as well as blocks within those SuperBlocks. SBA enables you to create scripts that automatically create and edit block diagrams.

For more information about SBA, enter `help sba` in the Xmath command area.

Learning SystemBuild

This section describes common tasks involving SystemBuild models. In this section you will create and configure a project, add a SuperBlock to the project, and add blocks to define the functionality of the SuperBlock. This SuperBlock will generate a sine wave and multiply that sine wave by a fixed value.

You can use several different methods to accomplish many of these tasks; however, to simplify the presentation, this section describes only one method.

Creating a SuperBlock

Complete the following steps to create a SuperBlock.

1. Launch MATRIXx.
2. If you do not see the **Solution Explorer** view, select **View»Solution Explorer** to display this view.
3. Create a new project named `Demo Project`. For information about creating projects, refer to the *Creating a New Project* section of Chapter 2, *Projects and Solutions*.
4. Configure the project. For information about configuring projects, refer to the *Configuring the Project* section of Chapter 2, *Projects and Solutions*.
5. Expand the **Demo Project** item in the **Solution Explorer** view.
6. Right-click the **Diagrams** folder and select **New Diagram» SuperBlock**. MATRIXx displays an empty Editor window.

Notice the title of this window displays the project and SuperBlock name, which is currently **Demo Project\<Unnamed>**. Also notice that the SuperBlock does not yet appear in the **Diagrams** folder in the current project.

Adding the SuperBlock to the Project

The next step is to add the SuperBlock to the current project. To make this change, select **SuperBlock»Update As** from the Editor window pull-down menu. Enter `Sample SuperBlock` in the **Name** text box and click **Update**.

After you make these changes, the **Solution Explorer** updates to show the **Sample SuperBlock** item in the **Diagrams** folder of the **Demo Project** project.

The next step is to save the project. Right-click the **Demo Project** item and select **Save**. The **Solution Explorer** view now resembles Figure 4-4.

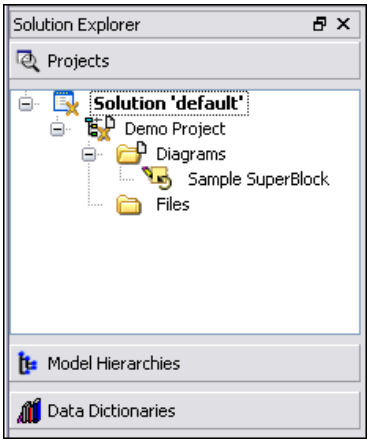

Figure 4-4. Project with New SuperBlock

In Figure 4-4, notice the pencil and circle icons on top of the **Sample SuperBlock** icon. The pencil indicates that the SuperBlock is open in an Editor window. The circle indicates that the SuperBlock is continuous. Different types of SuperBlocks have different icons in the **Solution Explorer** view.

Configuring the SuperBlock

Complete the following steps to configure this SuperBlock.

1. If you do not see the SystemBuild tools, select **View»Tools** to display these tools.
2. Click the **Selection** tool.
3. Double-click an empty area in the block diagram to launch the **SuperBlock Properties** dialog box. You use this dialog box to configure the properties of the SuperBlock, including the name, timing information, and number of inputs and outputs.

Figure 4-5 shows the **SuperBlock Properties** dialog box for this SuperBlock.

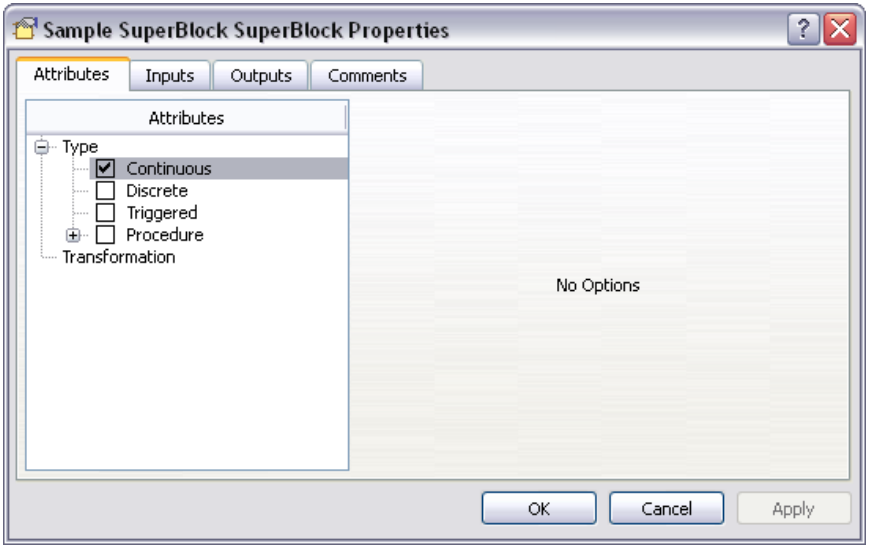

Figure 4-5. SuperBlock Properties Dialog Box

Complete the following steps to configure this SuperBlock.

1. On the **Attributes** tab, ensure the **Continuous** checkbox contains a checkmark.
2. Click the **Inputs** tab and ensure the **Inputs** text box has a value of `0`.
3. Click the **Outputs** tab and enter `1` in the **Outputs** text box.
4. Click **OK** to save changes and return to the block diagram. The Editor window displays the empty SuperBlock you configured. Notice the bottom-right of the Editor window displays the SuperBlock type and the number of inputs and outputs.

Adding Blocks to the SuperBlock

The next step is to add blocks to make the SuperBlock functional. Complete the following steps to add blocks to the SuperBlock.

1. If you do not see the **Blocks** view, select **View»Blocks** to display this view.
2. Click the **Generators** palette in this view.
3. Click the Sine Wave Generator block, and then click again in the Editor window to place the block on the block diagram.

4. Click the **Algebraic** palette.
5. Place a Gain block on the block diagram.
6. Move the Gain block to the right of the Sine Wave Generator block.

   The block diagram now resembles Figure 4-6.

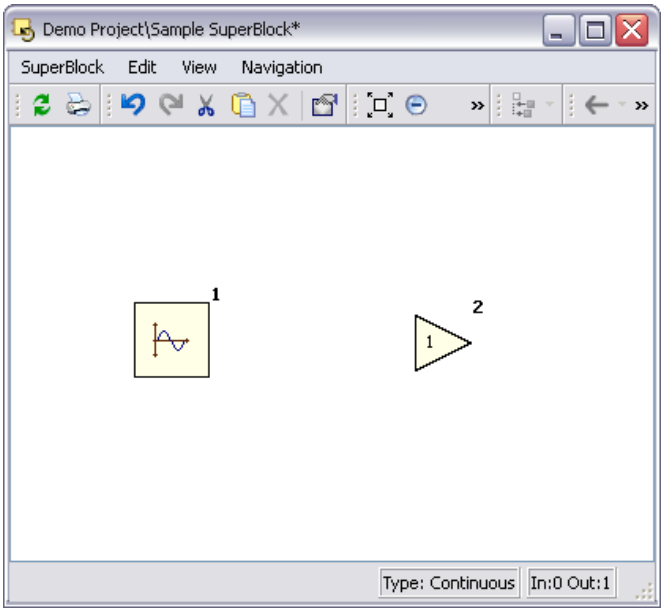

Figure 4-6. The Sine Wave Generator and Gain Blocks on the Block Diagram

7. Select **SuperBlock»Update into Project**. MATRIXx updates the SuperBlock into the project. After you update the SuperBlock, the **Solution Explorer** view displays several items in bold, because these items now have unsaved changes.

   The next step is to write the changes to disk, which you do by saving the project.
8. Save the project. For information about saving the project, refer to the *Saving the Project* section of Chapter 2, *Projects and Solutions*.

   After you save the project, MATRIXx displays the project and SuperBlock in regular font.

Connecting Blocks Together

Wires in SystemBuild represent the flow of data. You typically connect the output of one block to the input of another block to represent the flow of data.

Complete the following steps to connect the output of the Sine Wave Generator block to the input of the Gain block.

1. Click the **Connection** tool. You can also press <Tab> until this tool is highlighted.
2. Click the Sine Wave Generator block, and then click the Gain block. MATRIXx launches the **Connection Editor** dialog box, shown in Figure 4-7.

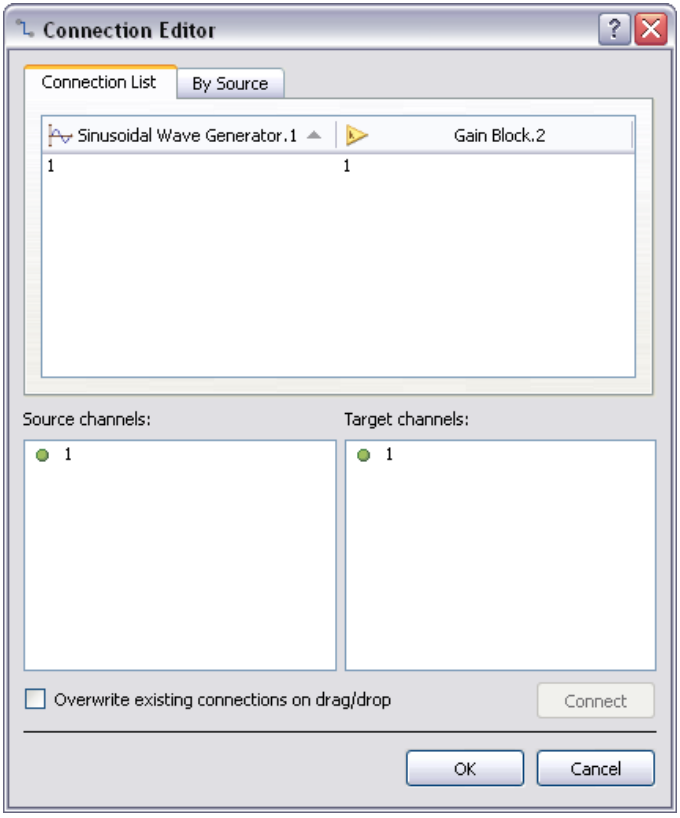

Figure 4-7. The SystemBuild Connection Editor Dialog Box

3. Drag the **1** from the **Source channels** list to the **1** of the **Target channels** list. This action creates a connection between the two blocks. MATRIXx updates the dialog box to display the existing connection.
4. Click **OK** to save changes and return to the block diagram. Notice the wire that now connects the two blocks. You can double-click this wire with the **Selection** tool to change the connection settings. Hovering over the wire also displays the connection information.

Tip MATRIXx automatically finds a route for the wire after you create it. To manually route an existing wire, you can move the blocks to which the wire is connected. You can also drag the handles that appear when you click the wire with the **Selection** tool.

Connecting a Block to the Output of the SuperBlock

To simulate this SuperBlock, you must connect the output of the Gain block to the output of the SuperBlock itself. You use this external connection to access the output of the SuperBlock in Xmath.

Complete the following steps to make this external connection.

1. Right-click the Gain block and select **Connection»To External Outputs**. MATRIXx launches the **Connection Editor** dialog box again.
2. Drag the **1** from the **Source channels** list to the **1** of the **Target channels** list. This action creates a connection between the output of the Gain block and the output of the SuperBlock.
3. Click **OK** to save changes and return to the block diagram. Notice the wire connecting the Gain block to the external output, which is represented by a horizontal pentagon. This pentagon displays a `1` to indicate it is the first external output of the SuperBlock.

The block diagram now resembles Figure 4-8.

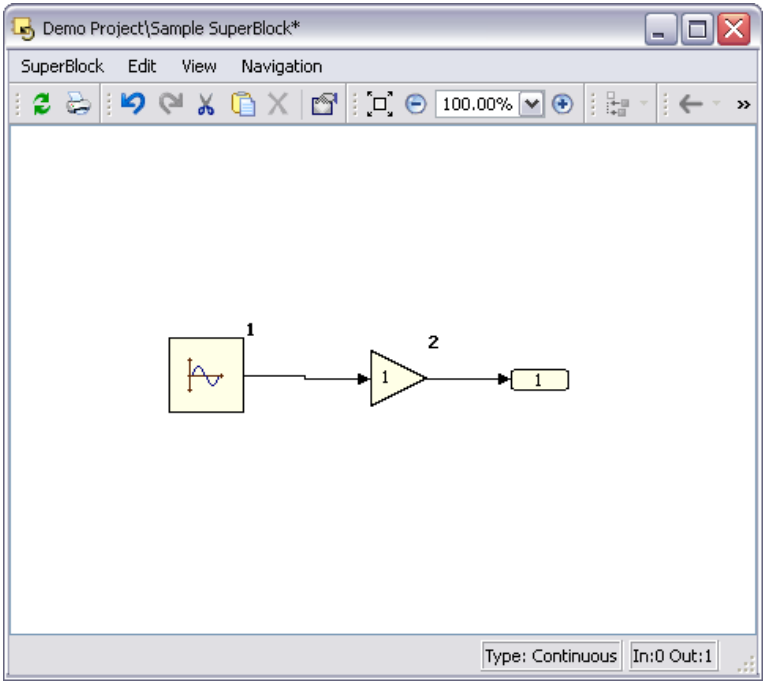

Figure 4-8. The Finished SuperBlock

4. Update the SuperBlock and save the project.

This SuperBlock now generates a sine wave and multiplies that sine wave by a scalar gain. SystemBuild sends this multiplied value to the output of the SuperBlock, so you can access this value in Xmath.

Configuring the Blocks

By default, the Sine Wave Generator block generates a 1 Hz sine wave, and the Gain block multiplies this sine wave by a value of 1. You can configure these blocks to meet the needs of the model.

Complete the following steps to configure the Sine Wave Generator and Gain blocks.

1. Double-click the Sine Wave Generator block to launch its **Block Properties** dialog box, shown in Figure 4-9.

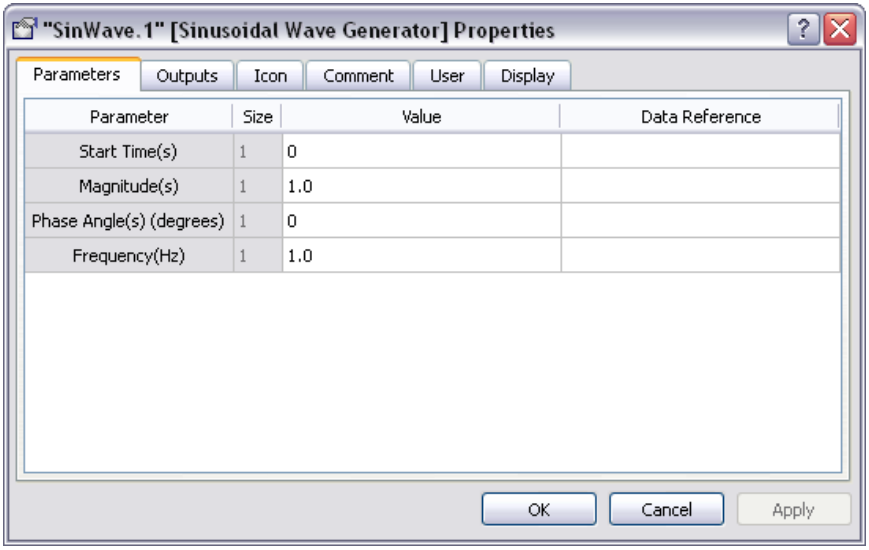

Figure 4-9. The Sine Wave Generator Block Properties Dialog Box

2. On the **Parameters** tab, locate the **Frequency(Hz)** row.
3. Enter `500` in the **Value** column of this row. This value specifies the block generates a sine wave at 500 Hz.
4. Click the **Outputs** tab.
5. Enter `sine wave` in the **Label** column of row 1.
6. Click the **Display** tab.
7. Enter `Generate Sine Wave` in the **Block Name** text box.
8. Expand the **Output Pins Attributes** item and place a checkmark in the **Show Labels** checkbox.
9. Click **OK** to save changes and return to the block diagram.

Notice the block is now labeled **`Generate Sine Wave`**.

Complete the following steps to configure the Gain block.

1. Double-click the Gain block to launch the **Block Properties** dialog box.

2. On the **Parameters** tab, enter `5` in the **Value** column of the **Gain(s)** row.
3. Click the **Inputs** tab. Because this input is connected to the labeled output of the Sine Wave Generator block, the input signal is labeled **sine wave**.
4. Click the **Display** tab.
5. Enter `Multiply by Gain` as the block name.
6. Click **OK** to save changes and return to the block diagram. Notice the Gain block displays **5** to reflect the change you made.
7. Update the SuperBlock.
8. In the **Solution Explorer** view, right-click the current solution and select **Save**. Name the solution savefile `Getting Started.sln-mtx`, and place this file in a convenient location. Notice how saving the solution automatically saves all projects.

This SuperBlock now generates a 500 Hz sine wave and multiplies this sine wave by 5.

Displaying All the Blocks in a SuperBlock

This SuperBlock has only two blocks. However, SuperBlocks with many blocks can be difficult to navigate. To display a simple list of all blocks in a SuperBlock, select **View»Outline** to display the **Outline** view. When you select a block from this view, the Editor window scrolls to locate that block on the block diagram.

Simulating the Model

Now that you have created a model, configured the blocks, and made the necessary connections, the next step is to simulate the block. You can simulate the block by using either Xmath or the Editor window. The following sections describe both these methods.

Simulating the Model in Xmath

Complete the following steps to simulate this model by using Xmath.

1. In the **Solution Explorer** view, right-click the **Demo Project** item and select **Show Xmath**. MATRIXx displays the **Xmath Commands** window associated with this project.
2. Enter the following command in the command area:

```
t = [0:0.1:5]';
```

This command defines a time vector *t* that MATRIXx uses to simulate the model. The `'` symbol transposes the row vector that would be created without this symbol. Time vectors must be column vectors.

3. Enter the following command in the command area:

```
y=sim("Sample SuperBlock",t,{graph});
```

This command simulates the model and graphs the results in an **Xmath Graphics** window.

Simulating the Model from the Editor Window

Complete the following steps to simulate this model from the Editor window.

1. In the **Solution Explorer** view, right-click the Sample SuperBlock item and select **Simulate**. MATRIXx launches the **Simulation** dialog box, shown in Figure 4-10.

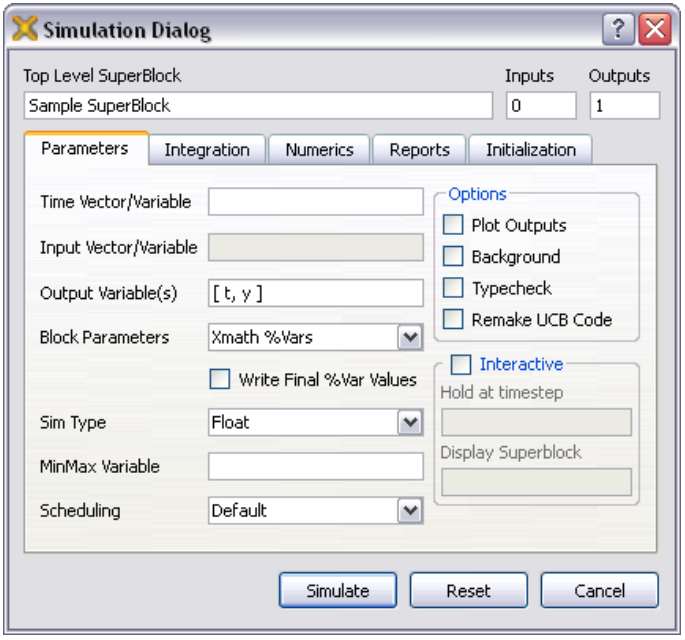

Figure 4-10. The Simulation Dialog Box

2. Enter `[0:0.1:5]'` in the **Time Vector/Variable** text box.

Note You can also enter defined Xmath variables in this dialog box. For example, because you defined *t* in the *Simulating the Model in Xmath* section of this chapter, you can enter `t` in the **Time Vector/Variable** text box.

3. Place a checkmark in the **Plot Outputs** checkbox.
4. Click **Simulate**. MATRIXx simulates the model and graphs the results.
5. Close the solution by selecting **File»Close Solution** from the MATRIXx window pull-down menu.

Navigating the Model Hierarchy

This section describes how to navigate a hierarchical model of an F14 jet. This model consists of several SuperBlocks.

Complete the following steps to open this model.

1. Select **File»Open»SystemBuild Model**.
2. Navigate to the `matrixx\mx_xx\sysbld\examples\f14\` directory, where *xx* is the version of MATRIXx you installed.
3. Select `f14new.sbd-mtx` and click **Open**.

After you open the model, MATRIXx executes the following steps:

- Creates a new, unconfigured project named **f14new**.
- Creates a subfolder in the **Diagrams** folder of this project. This subfolder is also named f14new.
- Associates a savefile with this subfolder. For information about savefiles, refer to the *Savefiles and Folders* section of Chapter 2, *Projects and Solutions*.
- Places all SuperBlocks into this subfolder.

To display a hierarchical list of the SuperBlocks in this model, click the **Model Hierarchies** pane in the **Solution Explorer** view. Navigate the model hierarchy by using this pane.

You can also navigate through the hierarchy from the Editor window. Complete the following steps to navigate the model using this method.

1. In the **Solution Explorer** view, double-click the sensor filtering SuperBlock. MATRIXx opens this SuperBlock in an Editor window.

2. In the Editor window, select **Navigation»Parents» f14new\Hierarchical Model**.

   MATRIXx opens the Hierarchical Model SuperBlock in the Editor window. Notice that this SuperBlock references the sensor filtering SuperBlock.

3. To move down in the model hierarchy, double-click the control SuperBlock on the Hierarchical Model block diagram. MATRIXx displays the control SuperBlock in the Editor window.

   The Editor window toolbar also contains buttons that duplicate the functionality of the **Navigation** pull-down menu.

4. Close the Editor window and the current solution.

Where to Go from Here

This chapter introduced you to SystemBuild and some basic functionality of the software. The following documents contain more information about SystemBuild.

- The *SystemBuild User Guide* contains detailed information about using SystemBuild to accomplish specific tasks. Access this user guide by launching MATRIXx and selecting **Help»MATRIXx Bookshelf**.
- Chapter 2 of the *SystemBuild User Guide* provides a more detailed tutorial of SystemBuild functionality.
- The *MATRIXx Help* contains topics about using SystemBuild. Access these topics by launching Xmath and entering `help systembuild`.
- SystemBuild provides several interactive demonstrations that showcase certain functionality. Access these demos by launching Xmath, entering `demo` in the command area, and selecting the **SystemBuild Demos** option. You can also access these demos by using the **Quick Start** window.

After you construct a model, you can use AutoCode to generate C or Ada code for that model. You can also use DocumentIt to generate documentation for that model. For more information about AutoCode, refer to Chapter 5, *AutoCode*. For more information about DocumentIt, refer to Chapter 6, *DocumentIt*.

5

AutoCode

Use AutoCode to generate ANSI C or Ada code from SystemBuild models. You can run this generated code on a computer or an actual controller in a rapid control prototyping (RCP) or hardware-in-the-loop (HIL) configuration.

AutoCode supports a template programming language (TPL) that you can use to customize many aspects of the generated code. This chapter provides information about generating both standard and customized code.

Generating Code from a SystemBuild Model

MATRIXx includes a discrete SystemBuild model of a cruise control system as a demonstration. Complete the following steps to generate code for this model.

1. If you are not displaying the **Solution Explorer** view, select **View»Solution Explorer** to display this view.
2. Select **File»Open»SystemBuild Model**.
3. Open `matrixx\mx_xx\sysbld\examples\auto\cruise_d.sbd-mtx`, where *xx* is the version of MATRIXx you installed. MATRIXx displays the associated SuperBlocks and State Transition Diagrams in the **Solution Explorer** view.
4. Right-click the continuous automobile SuperBlock and select **Generate Code**.

MATRIXx displays the **Code Generation** dialog box, shown in Figure 5-1.

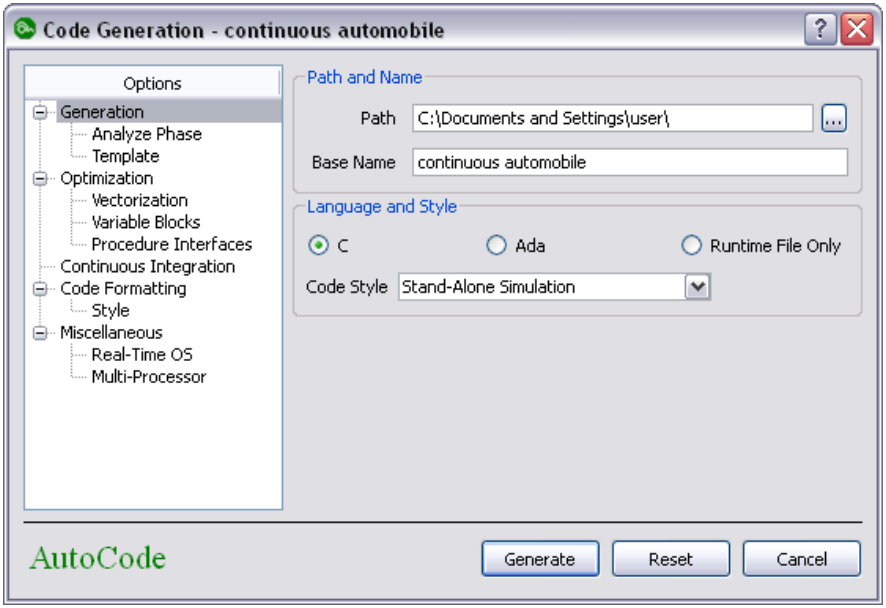

Figure 5-1. The Code Generation Dialog Box

5. Enter a filename in the **Base Name** text box. This filename serves as the filename for the generated code file and the intermediate runtime file. By default, MATRIXx provides the name of the SuperBlock as the filename.

   Notice the **Language and Style** section specifies that AutoCode generates **C** code for this model.

6. Click **Generate** to start the code generation process. You can display the **Xmath Commands** window to monitor this process.

After AutoCode finishes generating the code, Xmath displays the following text in the log area of the **Xmath Commands** window.

`Output generated in directory\filename.c.`

`Code generation complete.`

where ***`directory`*** is the current working directory and ***`filename`*** is the **Base Name** you specified in step 5.

The generated code is now present in the directory you specified.

Note You can generate code from only a top-level SuperBlock. For example, if you right-click the Set Speed SuperBlock, the shortcut menu does not display the **Generate Code** option. To view the SuperBlock hierarchy of the current model, click the **Model Hierarchies** pane in the **Solution Explorer** view.

Customizing Generated Code

Notice the **Options** list on the left side of the **Code Generation** dialog box. This list contains numerous settings that affect the generated code. For example, you can specify a different integration algorithm, optimize the code for more than one processor, and format the generated code. For information about these options, click the **?** button in the title bar of this dialog box.

This list also includes a **Template** option that you use to select a template file. Use this page to include a `.tpl` file written in the TPL. This language is a powerful way of customizing the layout and behavior of the generated code. You can use the TPL to generate code many different ways without having to reconfigure AutoCode each time you generate code.

Where to Go from Here

This chapter introduced you to AutoCode and the TPL. The following documents contain more information about these topics.

- The *MATRIXx Help* contains topics about using AutoCode. Access these topics by launching Xmath and entering `help autocode`.
- The *AutoCode User Guide* and the *AutoCode Reference* contain detailed information about using AutoCode to accomplish specific tasks. Access these guides by selecting **Help»MATRIXx Bookshelf**.
- The *MATRIXx Template Programming Language User Guide* contains detailed information about the TPL.

6

DocumentIt

Use DocumentIt to generate documentation from SystemBuild models. You can generate documentation in ASCII text, Adobe FrameMaker, Microsoft Word, and WordPerfect formats.

DocumentIt supports a template programming language (TPL) that you can use to customize many aspects of the generated documentation. This chapter provides information about generating both standard and customized documentation.

Generating Documentation from a SystemBuild Model

MATRIXx includes a discrete SystemBuild model of a cruise control system as a demonstration. Complete the following steps to generate documentation for this model.

1. If you are not displaying the **Solution Explorer** view, select **View»Solution Explorer** to display this view.
2. Select **File»Open»SystemBuild Model**.
3. Open `matrixx\mx_xx\sysbld\examples\auto\cruise_d.sbd-mtx`, where *xx* is the version of MATRIXx you installed. MATRIXx displays the associated SuperBlocks and State Transition Diagrams in the **Solution Explorer** view.
4. Right-click the continuous automobile SuperBlock and select **Generate Docs**.

MATRIXx displays the **Document Generation** dialog box, shown in Figure 6-1.

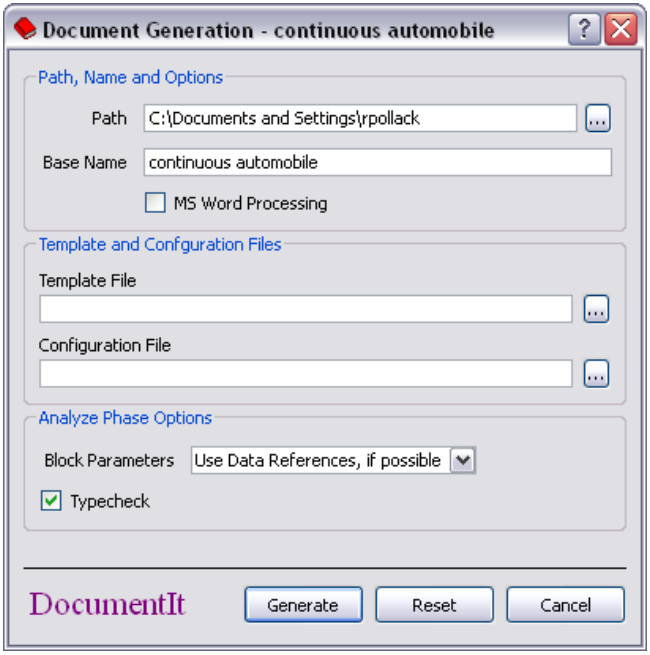

Figure 6-1. The Document Generation Dialog Box

5. Enter a filename in the **Base Name** text box. This filename serves as the filename for the generated documentation file and the intermediate runtime file. By default, MATRIXx provides the name of the SuperBlock as the filename.

Note You do not need to enter a file extension. DocumentIt supplies the default `.doc` extension for you.

6. Click **Generate** to start the documentation generation process. You can display the **Xmath Commands** window to monitor this process.

After DocumentIt finishes generating the documentation, Xmath displays the following text in the log area.

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

Note The `.doc` file is in ASCII format. The current defaults also produce a `.rtf` file, which contains Microsoft Word markup commands.

Customizing Generated Documentation

You can use the TPL to write template files that customize the documentation you generate. Template files are ASCII `.tpl` files that contain text and template command parameters, which specify the way DocumentIt generates documentation. You can also enter software markup commands in template files. For example, you can enter specific formatting information for Microsoft Word or Adobe Framemaker documents. These markup commands automatically format the document when you open that document with the corresponding software.

Where to Go from Here

This chapter introduced you to DocumentIt and the TPL. The following documents contain more information about these topics.

- The *MATRIXx Help* contains topics about using DocumentIt. Access these topics by launching Xmath and entering `help documentit`.
- The *MATRIXx DocumentIt User Guide* contains detailed information about using DocumentIt to accomplish specific tasks. Access this guide by selecting **Help»MATRIXx Bookshelf**.
- The *MATRIXx Template Programming Language User Guide* contains detailed information about the TPL.

7

MATRIXx Help Resources

The MATRIXx product family includes many ways to get help and technical support. This chapter provides information about all the ways in which you can access help while using the MATRIXx products.

Overview of the MATRIXx Help System

The MATRIXx help system consists of the *MATRIXx Bookshelf* and the *MATRIXx Help*. This section provides information about these two components.

MATRIXx Bookshelf

The *MATRIXx Bookshelf* is a collection of user manuals and installation guides in PDF format. These guides and manuals help you use MATRIXx to accomplish certain tasks.

Note You must install Adobe Acrobat Reader version 7.0 or later to use the *MATRIXx Bookshelf*.

Access the *MATRIXx Bookshelf* by using any of the following methods:

- Launch MATRIXx and select **Help»MATRIXx Bookshelf**.
- **(Windows)** Select **Start»All Programs»National Instruments» MATRIXx mx_*xx*»MATRIXx Bookshelf**, where ***xx*** is the version of MATRIXx you installed.
- Launch `matrixx.pdf`, which is located in the `matrixx\mx_xx\help\bookshelf\` directory.

MATRIXx Help

The *MATRIXx Help* is a collection of HTML files that provide detailed help on specific product topics, such as Xmath commands or SystemBuild blocks.

Access the *MATRIXx Help* by using any of the following methods:

- Launch MATRIXx and select **Help»MATRIXx Help**.
- Enter `help` in the Xmath command area.
- Enter `help` *`topic`* in the Xmath command area, where *`topic`* is the topic you want to open. For example, to receive help on the Xmath function `makepoly`, enter `help makepoly` in the Xmath command area.
- **(Windows)** Select **Start»All Programs»National Instruments» MATRIXx mx_*xx*»MATRIXx Help**.
- Enter `mtxhelp` on the operating system command line. You can use this method only if the MATRIXx environment variables are properly set and the default browser is in the system path. For information about environment variables, refer to the *Environment Variables* section of the *Xmath User Guide*.

Context-Sensitive Help

MATRIXx features context-sensitive help. Clicking the **?** button in the title bar of most dialog boxes launches the help for that dialog box. Some windows have **Help** buttons or **Help** pull-down menus that you can also use.

MATRIXx Help System Conventions

The *MATRIXx Help* and the *MATRIXx Bookshelf* use several formatting and terminology conventions, each with specific meaning. This section provides information about these conventions.

Formatting Conventions

Xmath input appears in `monospace` font, and Xmath output appears in **`monospace bold`** font immediately below the output. If the output is too long to convey in the help, continuation marks … or : indicate continuation. These marks also replace missing parts. Example 7-1 shows these conventions.

Example 7-1 Handling Xmath Input and Extended Xmath 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 ...

y=x'
y (a rectangular matrix) =

  0.827908     0.559594
      :            :
      :            :
  0.0568928    0.988541
```

Notice the Xmath input in Example 7-1 does not include a prompt. This convention indicates you are typing in the Xmath command area.

If the input is too long for a single line, subsequent lines are indented. Example 7-2 shows this convention.

Example 7-2 Handling Extended Xmath Input

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

Symbol Conventions

Table 7-1 shows symbols and their meaning in the MATRIXx help system.

Table 7-1. Symbol Conventions

Symbol	Meaning
%	**(UNIX)** Operating system prompt for C shell.
$	**(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. You generally do not type these brackets when entering the information. In Xmath, brackets represent a vector, matrix, or variable list.

Table 7-1. Symbol Conventions (Continued)

Symbol	Meaning
< >	Angle brackets indicate a key or keys you press.
|	A vertical bar separating two text items indicates that either item can be entered as a value.

MATRIXx Release Information

MATRIXx contains the following release documents:

- The *What's New in MATRIXx* document, available by selecting **Help» MATRIXx Bookshelf**, contains an overview of features in each new major release.
- The *MATRIXx Release Notes*, available by selecting **Help» Quick Start Window**, contains information specific to the latest point or maintenance release.

A

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Discussion Forums at ni.com/forums. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit ni.com/services or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

Symbols

% variable	*See* data reference.
?	Command delimiter in Xmath that forces output to display in the log area.
;	Command delimiter in Xmath that prevents the log area from displaying output.

B

block	A basic component of SystemBuild models. Blocks receive inputs and calculate outputs based on the properties you set for the block.
block diagram	The area of SystemBuild where you place blocks to define a model.

C

command area	The area in Xmath where you enter functions and commands.
connection	A link in SystemBuild that defines the path of a signal. Connections can be internal (between blocks inside a SuperBlock) or external (between SuperBlocks or between a SuperBlock and a real-world input/output).

D

data reference	A variable defined in both Xmath and SystemBuild that corresponds to the value of a SystemBuild block property.
diagram	A SuperBlock or a State Transition Diagram. You organize diagrams in the **Diagrams** project folder.
dynamic system model	Mathematical representation of a dynamic system, or a physical system you can describe with differential or difference equations. You use SystemBuild blocks to reconstruct this model in software.

E

Editor window	The component of SystemBuild that displays the contents of a diagram. You use the Editor window to edit diagrams.
environment variable	Shortcuts you use in Xmath to stand in for the location of MATRIXx products on the hard drive.

I

IDE	Integrated Development Environment. The graphical user interface you use to write Xmath scripts, create SystemBuild models, and perform related tasks in MATRIXx.
input vector	The vector that defines the external input to a SystemBuild model.
IPC	Interprocess Communication. The mechanism the LNX uses to link external C and FORTRAN code in Xmath scripts.

L

LNX	Linked External. The mechanism Xmath uses to link external C and FORTRAN code in Xmath scripts.
log area	The area of Xmath that displays the results of functions and commands you enter.
lookup path	The set of directories Xmath searches for functions and commands you enter.

M

`.msc` file	The file extension for a custom MathScript command.
`.msf` file	The file extension for a custom MathScript function.
`.mso` file	The file extension for a custom MathScript object.
MathScript	The language you use to define custom Xmath functions, commands, and objects.
model	*See* dynamic system model.

O

object	An Xmath data type, such as a vector, a polynomial equation, or matrix. Objects can be numeric or nonnumeric.

P

pane	A particular way of looking at projects in the **Solution Explorer** view. This view has three panes: **Projects**, **Model Hierarchies**, and **Data Dictionaries**.
parameter variable	*See* data reference.
partition	A named, non-hierarchical directories that contains Xmath data.
path	*See* lookup path.
PDM	Parameter-Dependant Matrix. An Xmath object that stores matrices in relation to an independent parameter or domain, such as time or frequency.
PGUI	Programmable Graphical User Interface. The capability of Xmath to support elements, such as dialog boxes and radio buttons, that provide user interface to MathScripts.
pin	An input or output of a SystemBuild block as displayed on the block diagram. You connect pins together to form connections.
project	A collection of SystemBuild model files, Xmath scripts, and related files.

S

savefile	A file on disk that references a solution, project, project folder, or individual SuperBlock. Each item has its own type of savefile. You can also configure how each item handles savefiles.
SBA	SystemBuild Access. The mechanism you use to perform SystemBuild tasks by using Xmath commands.
script	A text file, written in the MathScript language, that you call from Xmath.
signal	Data that flows through the SystemBuild blocks that comprise a model.

Simulator	The component of SystemBuild you use to simulate the behavior of a model.
solution	A collection of one or more projects.
Solution Explorer	The view that displays the current solution, any associated projects, and all project files in a hierarchical manner.
subsystem	A SuperBlock you can call from another SuperBlock.
SuperBlock	The basic hierarchical object in SystemBuild. SuperBlocks serve as a container for blocks and define the environment in which these blocks operate.

T

time vector	The vector that defines the points in time at which SystemBuild evaluates a model. The time vector must be a column vector.
TPL	Template Programming Language. The language you can use to customize generated code in AutoCode or generated documentation in DocumentIt. For more information about the TPL, refer to the *MATRIXx Template Programming Language User Guide*, available by selecting **Help»MATRIXx Bookshelf**.

U

UCI	User-Callable Interface. The mechanism a C program uses to invoke Xmath as a computational engine.

V

view	A dockable window that displays useful information inside the MATRIXx window.

W

work area	The area of the MATRIXx window where open applications, such as SystemBuild and Xmath, appear.
working directory	The first directory in which Xmath saves and looks for data.
workspace	The contents of all partitions, including defined variables, currently loaded in Xmath.

Index

K

M

N

P

R

S

T

V

W

X