SPARCworks/Ada User's Guide

SunSoft
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A

Part No.: 802-3473-10
Revision A, November 1995

2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

Contents

Figures

Tables

Preface

SPARCworks™/Ada is the OSF/Motif®, object-based, and window-based set of developer tools designed to enhance the programmer's productivity using SPARCompiler™ Ada. The *SPARCworks/Ada User's Guide* shows Ada developers how to use the suite of SPARCworks/Ada tools.

In this 3.0 release, SPARCworks/Ada consists of:

- AdaVision, a library browser and workplace
- AdaDebug, a GUI for the SPARCompiler Ada debugger
- LRMTool, a window-based, online version of the standard *Reference Manual for the Ada Programming Language*©[1]

SPARCompiler Ada and SPARCworks/Ada are collectively known as SPARCworks Professional Ada.

SPARCworks/Ada and SPARCompiler Ada

SPARCworks/Ada does not replace or take away any SPARCompiler Ada functionality. One of its major features is a graphical interface as an alternative to the Ada command-line interface. You can use the Ada command line interface interchangeably with SPARCworks/Ada.

1. Copyright 1983 U.S. Department of Defense

This guide is mainly for Ada developers. It assumes reader familiarity with Ada language development in general and SPARCompiler Ada development in particular. This guide concentrates on how to use SPARCworks/Ada to perform Ada operations, tasks, and commands; it does not attempt to explain in detail Ada development concepts or methods. For information about the underlying Ada toolset, see the SPARCompiler Ada documentation listed on page xix.

OpenWindows and OSF/Motif

All SPARCworks/Ada tools conform to OSF/Motif user interface standards. This *Guide* does not attempt to document the OSF/Motif GUI specification or the OSF/Motif workspace. For information about OSF/Motif, see the manuals listed under "Other Documentation" on page xix.

Software Requirements

Before you can run SPARCworks/Ada 3.0, the software configuration for your system must meet the following requirements:

- OSF/Motif
- SPARCompiler Ada 3.0
- Floating license server software

See the accompanying documentation for each of these items for instructions on how to install them. For information on operating system requirements, see the README file.

SPARCworks/Ada Installation

For instructions on how to install SPARCworks/Ada, see *Installing SunSoft Developer Products on Solaris.*

How This Book is Organized

Chapter 1, "Introducing SPARCworks/Ada," provides a conceptual overview of the SPARCworks/ Ada tools.

Chapter 2, "AdaVision Units," describes AdaVision units.

Chapter 3, "AdaVision Libraries," describes AdaVision libraries.

Chapter 4, "AdaDebug," explains AdaDebug functionalities.

Chapter 5, "LRMTool," discusses how to use LRMTool.

Notational Conventions

The following table describes the type changes and symbols used in this book.

Table P-1 Notational Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	Names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*`.`
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
Code samples are included in boxes and may display the following:		
%	UNIX C shell prompt	`system%`
$	UNIX Bourne and Korn shell prompt	`system$`
#	Superuser prompt, all shells	`root#`

In instructions for choosing items from submenus, an arrow points from the parent menu item to the child item. For example, the following means "choose the item Print from the File menu":

Choose File ➤ Print

Related Documentation

Listed in the following sections are other manuals and documents that pertain to SPARCworks/Ada.

Installation

To learn how to install SPARCworks/Ada, see *Installing SunSoft Developer Products on Solaris*.

*Online Release Notes (*README*)*

This release includes online release notes (called `README`) for:

- SPARCompiler Ada
- SPARCworks/Ada
- GXV-Ada, an optional component of SPARCompiler Ada

These `README` files list important known problems. They are located in:

```
/opt/SUNWspro/Ada3.0/README
/opt/SUNWspro/SWAda3.0/README
/opt/SUNWspro/GAda3.0/README
```

Manual Pages

This release includes online manual pages for all SPARCworks Professional Ada products. Refer to *Installing SunSoft Developer Products on Solaris* for instructions on installing these manual pages.

Online Help

Online context-sensitive help is available for SPARCworks/Ada by moving the pointer over a part of the GUI and pressing the Help key on SPARCstation™ keyboards.

SPARCompiler Ada Documentation

The following is a list of related SPARCompiler Ada documentation:

- *SPARCompiler Ada User's Guide*
- *SPARCompiler Ada Programmer's Guide*
- *SPARCompiler Ada Reference Guide*
- *SPARCompiler Ada Runtime System Guide*
- *GXV-Ada Programmer's Guide*
- *Multithreading Your Ada Applications*

Other Documentation

The following documentation is cited as reference:

- *SPARCworks/Ada Tutorial*
- *The Annotated Ada Reference Manual*, ANSI-MIL-STD-1815A-1983 (Annotated) 2nd Edition
- *OSF/Motif User's Guide*
- *DeskSet Environment Reference Guide*
- *Managing SPARCworks Tools*
- *Merging Source Files*
- *Solaris 2.x Developer Documentation*

Introducing SPARCworks/Ada 1

SPARCworks/Ada is the implementation of an Ada development environment. It applies the advantages of workstation technologies and object-based interfaces to large-scale Ada development.

SPARCworks/Ada consists of a suite of integrated object-based and window-based development tools designed for use with the SPARCompiler Ada compiler and toolset. In this 3.0 release, SPARCworks/Ada consists of:

- AdaVision
- AdaDebug
- LRMTool

In the unlikely event that a SPARCworks/Ada tool develops an internal inconsistency precluding further reliable processing, each of these three tools have been coded to dump core in `/tmp` to leave your applications core file intact..

1

1.1 AdaVision

AdaVision is the focal point for most SPARCworks/Ada development activities. AdaVision serves two main purposes. It functions as:

- An Ada library browser and manager
- The main workplace for editing, compiling, and debugging Ada units and programs

As the hub of SPARCworks/Ada tools, AdaVision allows you to start the other tools from within it. You can also start the other tools separately.

1.1.1 Unit-Based Programming

AdaVision is primarily a unit-based tool. Although the Ada units that make up an Ada library have source files, AdaVision displays Ada units, not files.

Using the Ada `a.make` command on a multiple-unit file, you recompile *all* of the units in that file even if only one unit has been changed. In practice, only those units that have been changed, as well as those which depend on these units, need to be recompiled. This latter step can best be accomplished with one Ada unit per file.

1.1.2 AdaVision as a Library and Unit Browser

You can use the AdaVision Unit View to browse existing Ada units or build new units, to view dependencies among units, and to locate and select the units with which to work.

You can select the libraries you want to browse or work with in the AdaVision Library View.

1.1.3 AdaVision as a Programming Workplace

AdaVision is the main workplace for performing both library building and management tasks, as well as tasks related to writing source code and compiling it into Ada units. You edit new or existing units and set options for the compiler in AdaVision windows. You can also run make, compile, and link jobs, as well as execute main programs.

When you edit a unit, AdaVision opens a separate text editor window and loads the file containing the unit into your editor of choice.

1.1.4 New Features and Changes from AdaVision 2.1

The following is a list of changes and new features in this 3.0 release of AdaVision:

- The Library Unit (LU) and Compilation Unit (CU) modes have been replaced by the Unit View and the Library View windows, which offer separate views of units and libraries. For more information, see "Unit View" on page 30 and "Opening the Library View" on page 60.
- You can now view multiple libraries in AdaVision.
- You can view and edit source files in your editor of choice: vi, emacs, or xemacs.
- Both WITH and spec-body dependencies appear in a single graph.
- Jobs are now displayed in a scrolling list. For more information, see "Managing AdaVision Jobs" on page 46.
- All SPARCompiler Ada commands and options are now supported.
- User-defined actions are now supported.

1.2 AdaDebug

The SPARCompiler Ada debugger, AdaDebug, is an object-based and window-based interface to `a.db`. AdaDebug supports most of the functionality of `a.db` in an easy-to-use graphical environment. It also provides a command line from which you can issue `a.db` commands directly.

As in AdaVision, AdaDebug as a debugging tool makes extensive use of windows, multiple views, icons, and other graphics. AdaDebug displays the program being debugged in a Program View window. By choosing menu items, you can set or clear breakpoints, step through the program, examine the stack, and perform other complex debugging tasks.

Using AdaDebug for multitasking programs, you can view the code for each task in separate Task View windows. Commands issued in a Task View window affect only the task associated with that window. In addition, the tool provides a task browser that presents a global view of the program tasks and a status window that displays information about the status of a task.

1.3 LRMTool

LRMTool is an online version of the *Ada Language Reference Manual*. Topics are extensively cross-referenced. The mouse-driven interface lets you move from one topic to another with ease. LRMTool has a history feature to help you navigate the document.

1.4 Building Libraries from Existing Code

When you start AdaVision in an Ada library directory, AdaVision automatically imports the Ada library to its library list, along with the names of libraries on the library's ADAPATH.

To create a new library:

1. **Start AdaVision and open the Library View by selecting View ➤ Libraries.**
2. **Choose File ➤ New Library in the Library View.**

If you have existing Ada source code, but no associated `ada.lib` file, create an Ada library for the code and import the code into the new library.

When you choose Actions ➤ Import in the Library View, AdaVision uses the Ada `a.make` facility to bring the units up-to-date by compiling the units you specify in the correct order. Once the source files are imported, link each main program unit. When the link job completes successfully, the program is ready to be run.

See Chapter 3, "AdaVision Libraries," for a more detailed description of building libraries.

1.5 General Features and Facilities

This section presents an overview of the SPARCworks/Ada features and facilities available across the suite of tools.

1.5.1 Icons

In an object-based GUI, you first select those objects you want to view or work with. The GUI relies on its knowledge of different types of objects to present you with the range of features and functions appropriate to the selected type.

Each icon in the AdaVision Unit View represents anAda unit object. The icons for each type of unit—package, subprogram, and so forth—have their own distinctive appearance. Each icon identifies an instance of a particular type of Ada object. See Figure 2-2 on page 33 for a picture and description of each kind of icon.

Icons convey other information, too. For example, the Sun logo in a subprogram body icon indicates that it is an executable main program.

1.5.2 Object-Based Interface

The Ada language is especially suited for an object-based interface. It is built from a set of formally defined units that translate readily and usefully into software objects.

Each type of object supports a range of functionality tailored specifically to it. Menu items and control fields change dynamically, depending on the object you have selected to work with or your place in a sequence of actions. Menu items or operations that are inappropriate for use with a given object at a particular time are either absent from the controls or inactive.

1.5.3 AdaVision Scrolling Lists

In AdaVision, some items appear in a scrolling list, sorted by columns of characteristics and rows of values. When a row is selected, its values appear in the appropriate text fields beneath the list. The text fields can show the entire value of any selection whose value is truncated in the list and, if the list is not

read-only, you can edit the information in the rows. The Update, Append, Move Up, Move Down, and Delete buttons edit values and alter the position in the list of a selected row.

1.5.4 AdaVision Job Information Windows

SPARCworks/Ada takes further advantage of object-based design and multitasking with Job Information windows.

Choosing Status ➤ Jobs in AdaVision opens a list of in-progress and completed jobs. Pressing the Information button then shows the command line used and any compiler or error messages of a selected job. In addition, pressing the Redo button starts the compile job again after any errors have been fixed.

1.5.5 Context-Sensitive Help

Online context-sensitive help is available for SPARCworks/Ada by moving the pointer over a part of the GUI and pressing the Help key on SPARCstation keyboards.

1.5.6 Remote Display

As with any other OSF/Motif application you can run a SPARCworks/Ada tool on another SPARC system and direct the window's display to your machine. In this case, your machine need not be a SPARC system; it just needs to be an X client, or an X11/NeWS or OSF/Motif client.

To set up remote display:

1. **Enter the `xhost` command at the system prompt on the display machine.** The `xhost` command is effective until the display machine is rebooted.
2. **Log on to a system on which SPARCworks/Ada has been installed.**
3. **Set the environment variable `DISPLAY` to the display machine.**

For information on executing remote jobs, see "Executing Remote Jobs" on page 32.

1.5.7 Cross-Development

In cross-development, developers use one system (host) to develop an application for another type of system (target). The host provides all the necessary development and test tools for that target. Only the final integration, testing, and debugging take place on the target machine.

SPARC-hosted VADScross® products, available from Rational Corporation, are a line of cross-development tools. Parts of the VADScross product line are compatible with SPARCworks/Ada. The use of SPARCworks/Ada and VADScross gives you a graphical interface to develop Ada applications for both SPARC and non-SPARC targets, and SPARCworks/Ada provides the same GUI for both self-target and cross-development. You can run self-target and cross-development projects at the same time from AdaVision.

After you have installed a SPARCworks/Ada-compatible VADScross product in your SPARC system, your `/etc/VADS` file includes the cross-development release(s) now available to you. You can then create a new library using the parent release of your choice.

Remote display, remote execution, and cross-development are compatible with each other. To take full advantage of these features, you can actually have four networked machines set up at the same time as follows:

- A *display_machine* displaying the windows of the SPARCworks/Ada tools
- A *swada_host* running the SPARCworks/Ada tools
- An *ada_host* running the VADScross debugger by remote execution
- A *target* machine performing a debugging session of the application

AdaVision Units 2

AdaVision serves both as an Ada unit browser and as a program development workplace for editing, compiling, and linking Ada units into executable programs. It is also the hub for SPARCworks/Ada, providing access to AdaDebug and LRMTool.

By manipulating SPARCworks/Ada objects from within AdaVision, you can carry out tasks associated with developing an Ada application.The Unit View gives you access to all of the units in the opened libraries, displayed as separate icons.

2.1 Starting and Exiting AdaVision

You can start AdaVision from anywhere in your file system. If you don't specify an Ada library path name, AdaVision loads the Ada library in the current directory.

AdaVision depends on the SPARCompiler Ada `a.header` command for unit information. It depends on `sup/legal.all` and `a.info` for library directive information.

To start AdaVision from a command line:

♦ **Type** `adavision [-full] [`*library pathname*`] &`
AdaVision opens the Unit View window. The optional parameters cause AdaVision to perform in the following ways:

`-full`: AdaVision displays and supports the full set of SPARCompiler Ada commands in the Actions menu and Options dialog. Without this parameter, AdaVision supports a subset of these commands.

library pathname: AdaVision loads the indicated library at startup. If you do not enter a library path name, AdaVision loads the current directory.

You can also add AdaVision to your OSF/Motif Workspace Programs menu. See OSF/Motif documentation for instructions.

AdaVision accepts all the command line arguments of a Motif/X application.

To exit AdaVision from the AdaVision menu bar:

♦ **Choose Unit ➤ Exit AdaVision.**

Upon startup, AdaVision displays the Unit View window, which shows icons of all the units in the loaded library and allows you to manage and perform actions on those units.

From the Unit View you can open a Library View window, which allows you to perform actions such as creating, opening, closing, deleting, and cleaning Ada libraries.

The Library View is described further in Chapter 3, “AdaVision Libraries.”

2.2 Unit View

The Unit View window shows icons of all the units in the opened libraries and is the top-level display window for AdaVision.

From the Unit View, you can:

- Edit code in a unit or write code for new units
- Make, compile, and link units

- Sort and filter units
- Print files or screens

Figure 2-1 shows the Unit View window in icon mode.

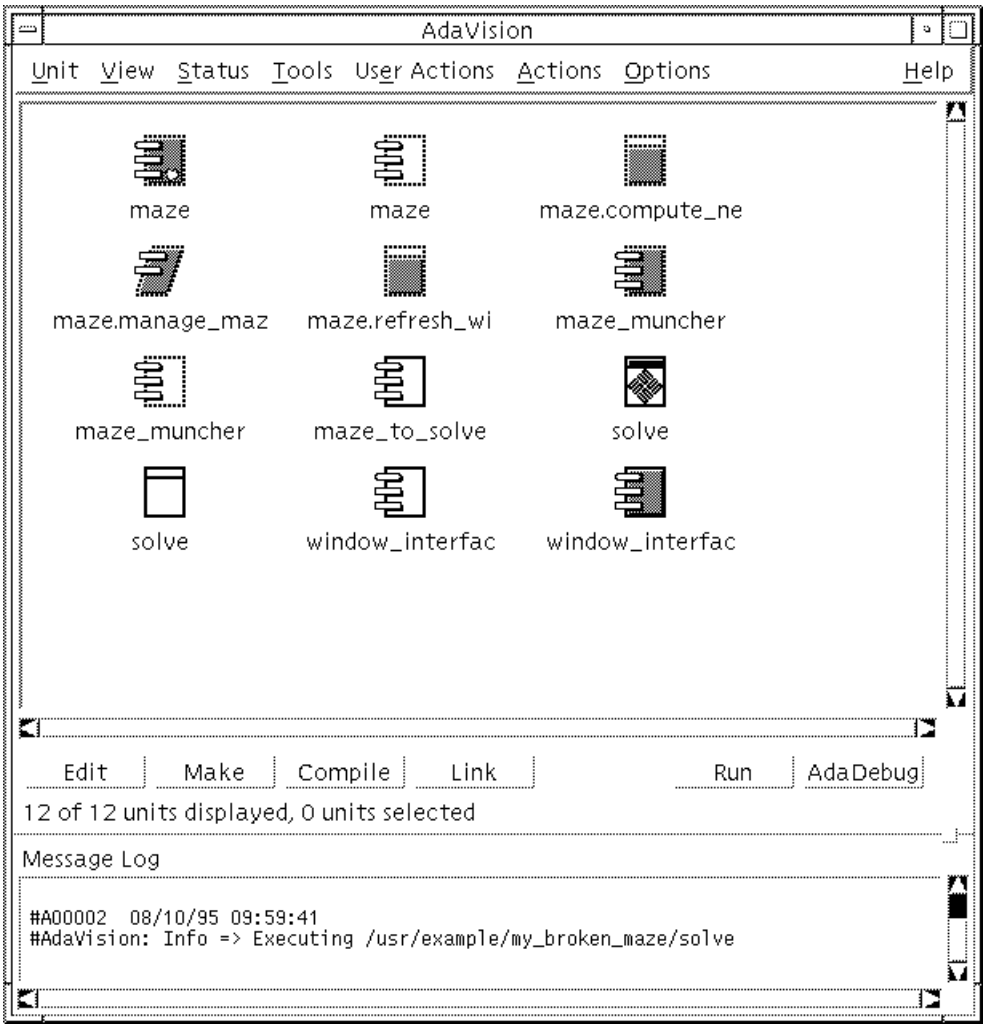


Figure 2-1 AdaVision Unit View in Icon Mode

AdaVision uses the Message Log to display information about the objects and the operations you perform, including error messages. The Unit View has a scrollbar for scrolling through large lists of units.

AdaVision icons are divided into three types: subprograms, packages, and tasks. They are either generic or nongeneric.

Specs and bodies for subprograms and packages are displayed as separate icons. Subunit bodies are displayed separately, and subunit names always contain a period (.) to distinguish them from other bodies.

A unit is displayed as a “broken” icon under one or more of the following conditions:

- The unit’s source file has been modified.
- The unit has been compiled for dependencies only (`-d`).
- The unit depends upon another unit that has been recompiled.

The broken state alerts you that the unit is obsolete and has to be recompiled.

Each unit image is shown both whole and broken here in both its larger icon and smaller glyph form.

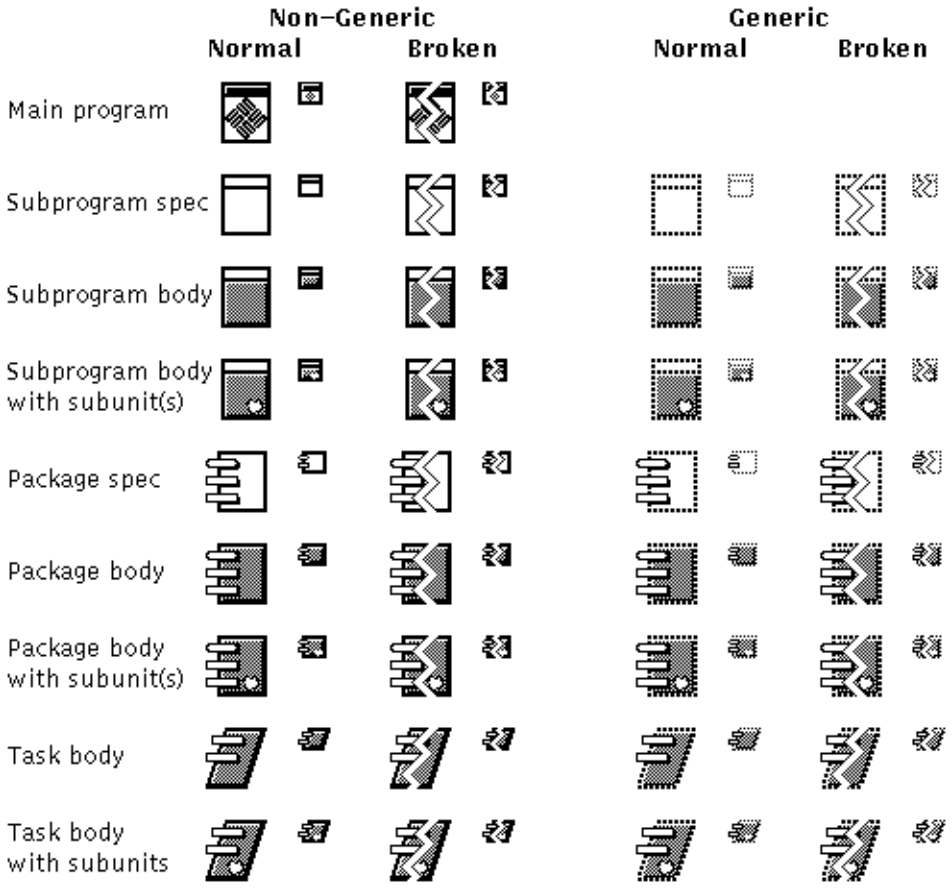


Figure 2-2 AdaVision Icons and Glyphs

You can view unit icons in icon mode, list mode, and graph mode.

2.2.1 Icon Mode

In icon mode, AdaVision units are shown as large images (icons) whose appearances describe the type and compilation state of the unit. See Figure 2-1 on page 31 for a picture of AdaVision in icon mode.

You can sort units in the icon mode by name or by type. For more information, see “Sorting Units for Display” on page 36.

2.2.2 List Mode

In list mode, AdaVision units are presented in a scrolling list as small images (glyphs) whose appearances describe the type and compilation state of the unit. The glyphs appear in a list sorted by name or type. List mode also displays a type description, modify date, compile date, and disk usage.

.

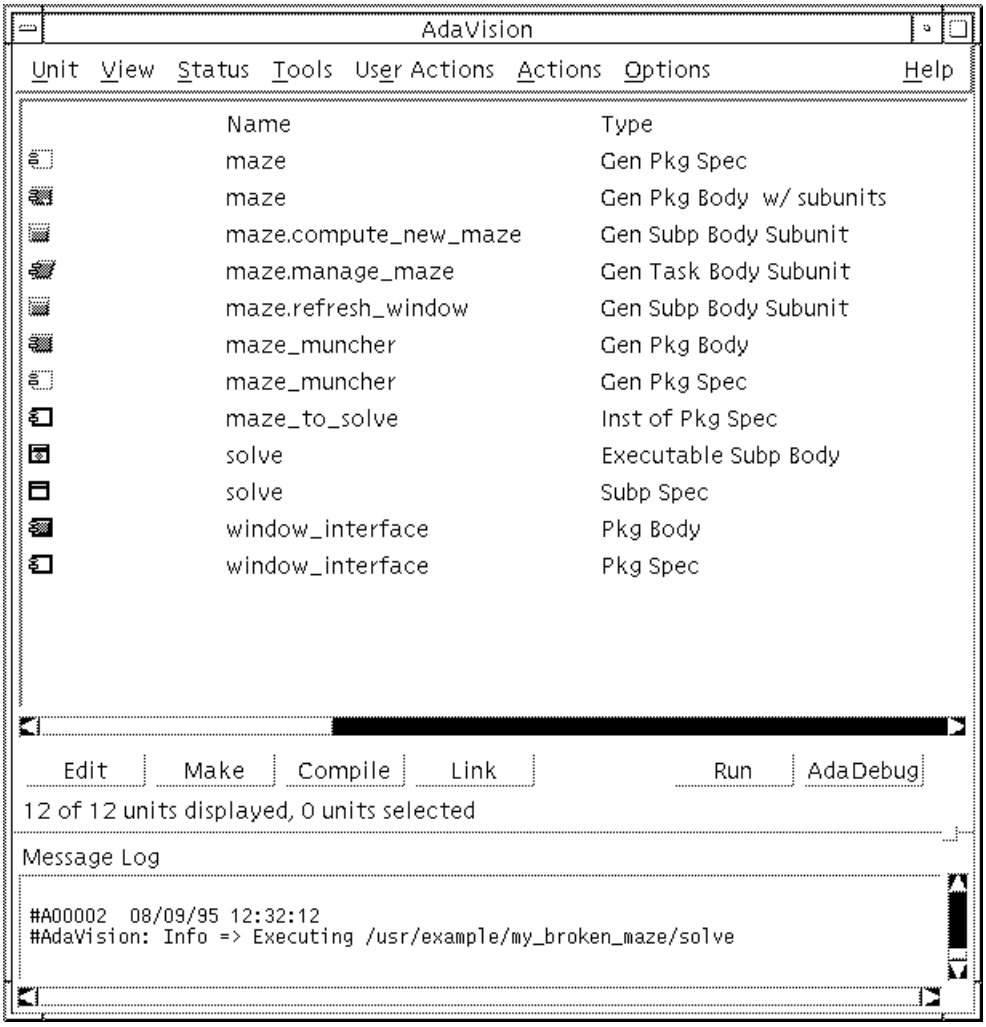


Figure 2-3 List Mode View

You can control which columns are displayed in list mode by choosing Options ➤ Filters. For more information see “Sorting Units for Display” on page 36.

2.2.3 Graph Mode

Graph mode displays a graph of unit dependencies. Spec-body-subunit dependencies appear as solid lines. WITH, inline, and generic body dependencies appear as dashed lines with an arrowhead indicating the direction of the dependence. You can show the transitive closure of WITHs (or the inverse relationship Is_WITHed_By) for selected units by using the View ➤ Dependencies submenu. See “Printing Files and Screens” on page 56 for information on printing the graph view.

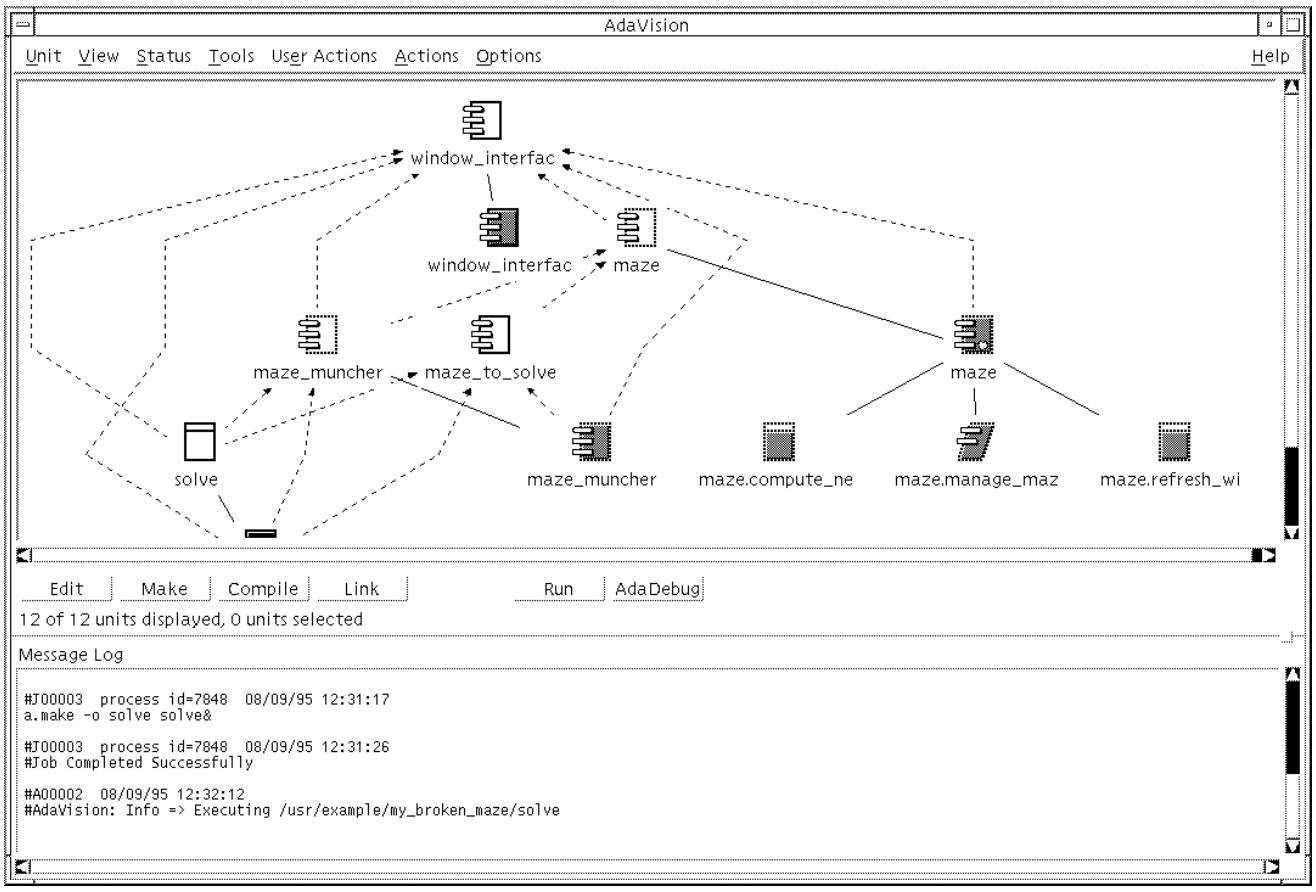


Figure 2-4 Graph Mode View

2.2.4 Sorting Units for Display

You can sort units in the icon mode and list mode in two ways:

- Alphabetically by unit name
- Alphabetically by unit type—generic package, generic subprogram, package, subprogram, tasks

To sort units:

1. **Choose Options ➤ Filters to open the Filter Options window.**
2. **Select the desired values for Sort By.**
3. **Select the value columns you want shown in list mode.**
4. **Click OK.**

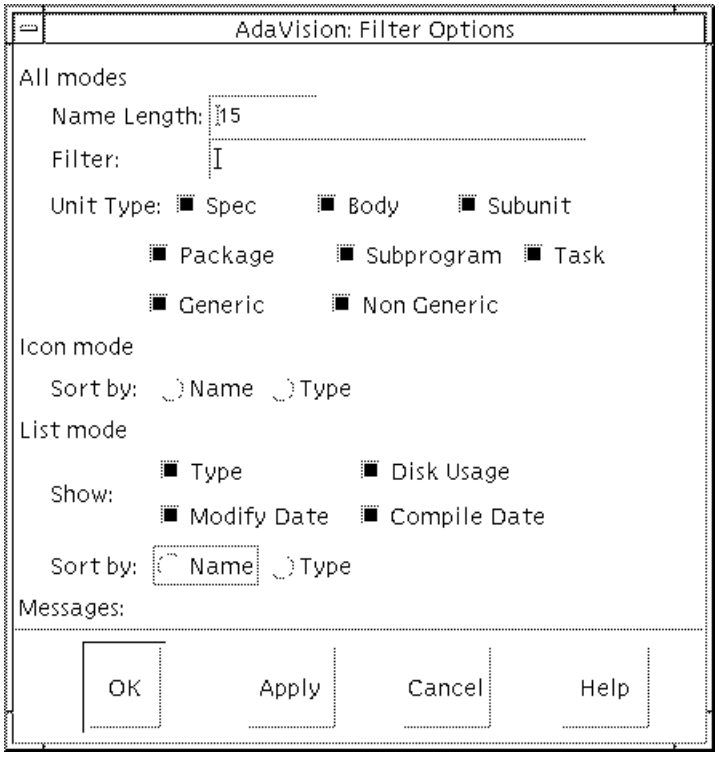


Figure 2-5 Filter Options Window

2.2.5 Filtering Units for Display

Filtering allows you to control which types of units AdaVision displays in the Unit View and how long the names of those units are. You can also direct AdaVision to display only those units with a specified pattern in their names.

To filter units for all modes of the Unit View:

1. **Choose Options ➤ Filters to open the Filter Options window.**
2. **Enter the maximum number of letters for unit name display in the Name Length text field.**
3. **If you want AdaVision to display only units matching a pattern, enter the pattern in the Filter text field.**
4. **Select the unit types you want displayed.**
5. **Click OK.**

2.3 Creating New Units

To create a new unit:

1. **Choose Unit ➤ New.**

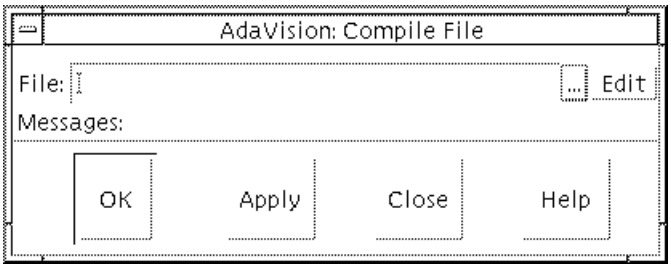


Figure 2-6 Compile File Window

2. **Enter the name of the source file for the new unit.**
   Clicking the ellipsis (. . .) button allows you to select the file name using a file chooser.

3. **Click the Edit button.**
   The Edit button opens your editor of choice, in which you can edit the new unit.

4. **Click OK to compile the unit.**
   When the unit compiles successfully, an icon for it appears in the Unit View.

2.4 Running a Program

The AdaVision Run action does not directly correspond to the SPARCompiler Ada `a.run` command. SPARCompiler Ada supports `a.run` for cross-targets only, but the AdaVision Run action applies to self-targets as well.

If the program makes use of standard input (`stdin`), standard output (`stdout`), or standard error (`stderr`), it should run in a Shell Tool window. The default for this control is True. You can also enter options to be passed to your program in the Options text field.

If you choose not to run a program in a Shell Tool window, the program output, if any, is written to the Shell Tool window from which you invoked AdaVision.

To run a program from the AdaVision Unit View in AdaDebug:

1. **Select the main program icon.**

2. **Choose Tools ➤ AdaDebug or click the AdaDebug button.**

2.5 Editing Units

When you open a unit, AdaVision loads the source file containing the unit into a window running your editor. You can open more than one unit object at a time.

To open a unit for editing:

- **Double-click on the unit's icon or select the unit you want to edit and click the Edit button.**

AdaVision opens a window for each selected object, using the editor of choice specified in the AdaVision Global Options window. For more information on the Global Options window, see "Defining Global Options" on page 54.

Each window editor opened from within AdaVision operates independently of AdaVision. You can move, resize, and close the window to an icon.

2.6 Viewing Unit Options

To view information about a selected unit:

♦ **Choose Options ➤ Unit to open the Unit Options window.**

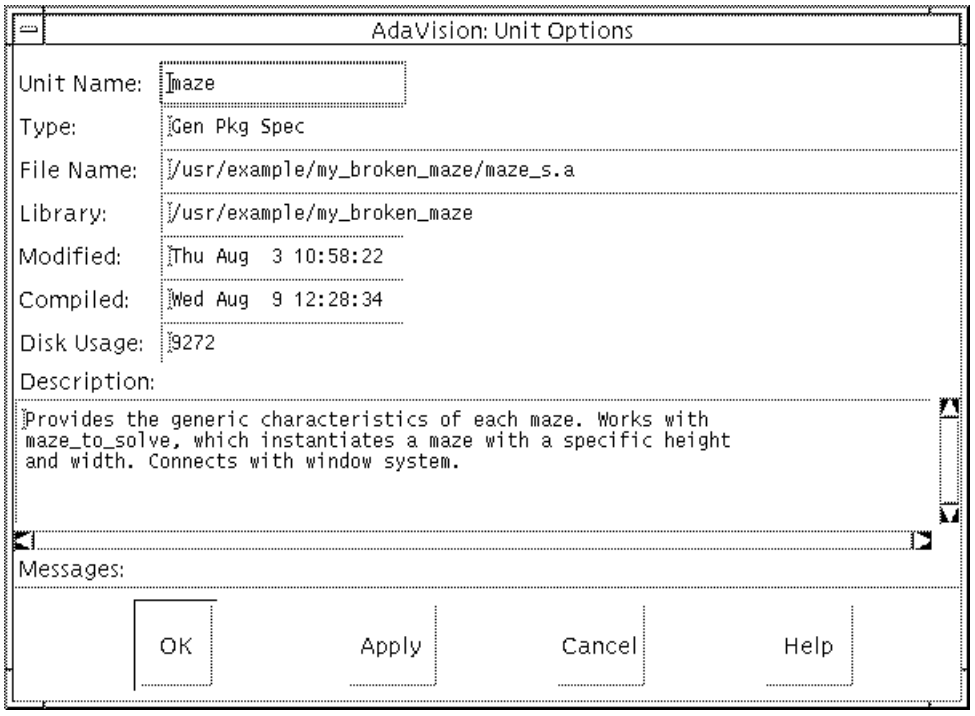


Figure 2-7 Unit Options Window

The Unit Options window displays the following information about the selected unit:

- Name
- Type
- File Name
- Library
- Last Date Modified
- Last Date Compiled
- Disk Usage
- Description (if applicable)

Once the Unit Options window is open, it automatically updates the displayed information to match any unit you select from the Unit View.

2.7 Modifying Action Options and Arguments

AdaVision allows you to modify the options and arguments of the actions listed in the Unit View Actions menu. The interface does not differentiate between valid and invalid entries, so you cannot confirm that the changes you make to an action's options or arguments are correct until after the action is executed.

To modify an action option or argument:

1. **Choose Options ➤ Actions to open the Action Options dialog.**

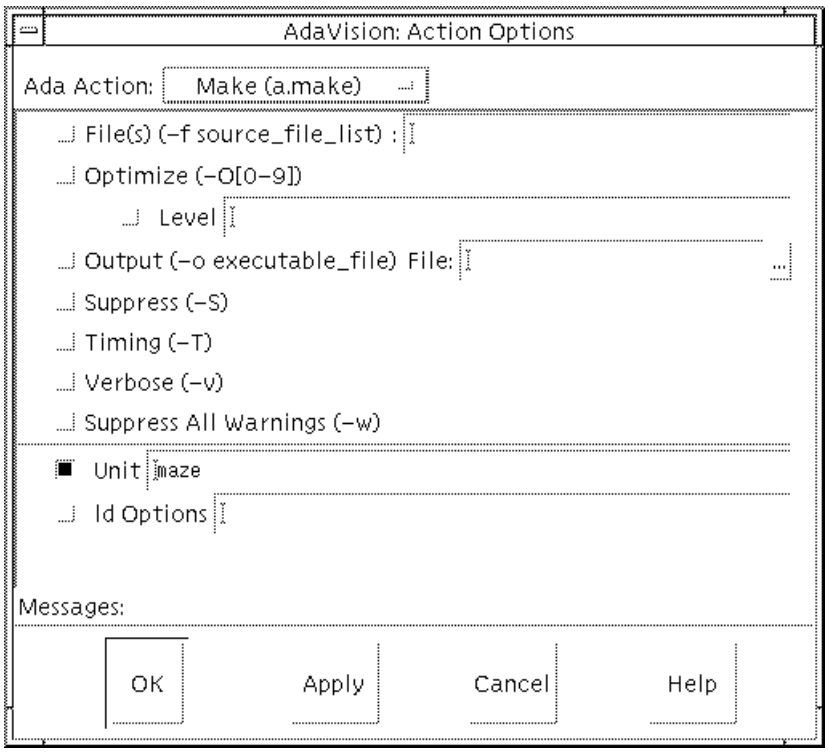


Figure 2-8 Action Options Window

2. **Select an action from the Action list.**
3. **Enter the appropriate information, as necessary.**
4. **Click Apply or OK.**

Once you have changed and applied a setting, it is saved for the rest of the current session and remains in effect for your next session of AdaVision.

2.7.1 Setting Options for Unit-Specific Actions

The following actions are unit-specific:

- Make
- Compile
- Link
- Debug
- Run

For these actions, AdaVision displays and saves options separately for each unit.

If no units are selected, AdaVision displays and saves the default options. Default options apply to new units and to units whose options have not been modified before. Actions that are not unit-specific have only default options regardless of the number of units selected.

If more than one unit is selected for a unit-specific action, AdaVision displays a composite of all unit options. If an option differs between any two units, the multi-value glyph appears, and the option displays as empty. When saving, all selected units are updated; any options still showing a multi-value glyph are not updated, and the option remains unchanged in each unit.

2.7.2 Pseudo-Variables

Action options and arguments can use pseudo-variables, placeholders that correspond to internal AdaVision data. The pseudo-variable is replaced by its actual value when it appears in the Action command string.

Since many actions have arguments based on selections of units or libraries, the argument fields of the Action Options dialog have been preset with pseudo-variables to use the proper selections.

Table 2-1 lists the defined pseudo-variables for AdaVision.

Table 2-1 Pseudo-Variable Definitions

Pseudo-variable	Replaced by
`#UNITSOURCES`	names of the source files of all selected units
`#UNITSOURCE`	name of the source file of one selected unit
`#UNITMAINS`	names of the executables of all selected units
`#UNITMAIN`	name of the executable of one selected unit
`#UNITS`	names of all selected units
`#UNIT`	name of one selected unit
`#LIBRARIES`	names of all selected libraries
`#LIBRARY`	name of one selected library
`#CURR_DIR`	name of the current directory
`#DEFAULT_SRC_EXT`	value of the Ada library `INFO` directive
`#LOGFILE`	name of the AdaVision log file

A pseudo-variable can match either a single object or name or a list of objects or names. Each single object or name is assigned the next selected object. If there are too few selected objects, AdaVision generates an error message. If there are too many selected objects, AdaVision generates as many action command lines as needed to exhaust the selected objects.

Pseudo-variables that match a list generate a string of names separated by blank characters. Any duplicate names are eliminated.

2.8 Creating User-Defined Actions

To define your own actions and save them in the User Actions menu:

1. **Choose User Actions ➤ Customize in the Unit View window to open the Customize User Actions dialog.**
   The dialog displays the current menu items and their properties.

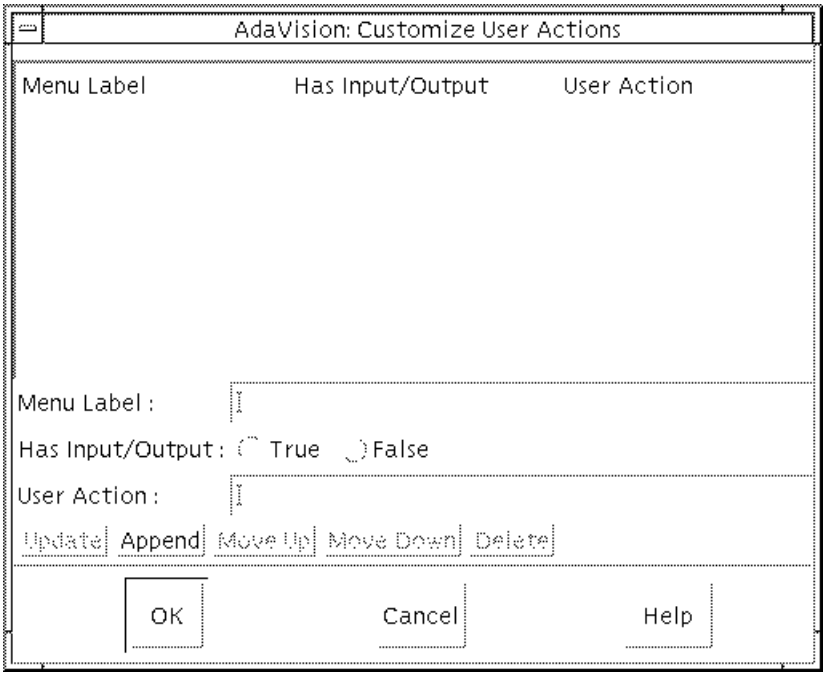


Figure 2-9 Customize User Actions Dialog

2. **Define your action in the User Action field.**
3. **In the Menu Label field, enter the name that will appear for this action in the User Actions menu.**
4. **Indicate if your action has input and/or output.**

You can insert the new button in any position in the User Actions menu by using the Move Up and Move Down buttons. The Update and Append buttons allow you to alter the definition of your action.

Once you have changed and applied a setting, it is saved for the rest of the current session and remains in effect for your next session of AdaVision.

2.9 Managing AdaVision Jobs

Actions that are also jobs—make, compile, and link—run in the background and are executed in a Job Status window. The Job Status window can also be opened independently to display information about completed jobs and jobs in progress.

2.9.1 Viewing Information on Jobs

To view information on jobs:

- **From the Unit View window, select Status ➤ Jobs to open the Status of Jobs window.**

Figure 2-10 Status of Jobs Window

To obtain information about a selected job, click the Information button. The Job Information window is displayed with the following information about the job:

- Job status
- Process ID
- Date and time started
- Date and time completed
- Command with options and arguments
- Job output

If a compile was unsuccessful, you can recompile from the Job Information window using the Redo Command button.

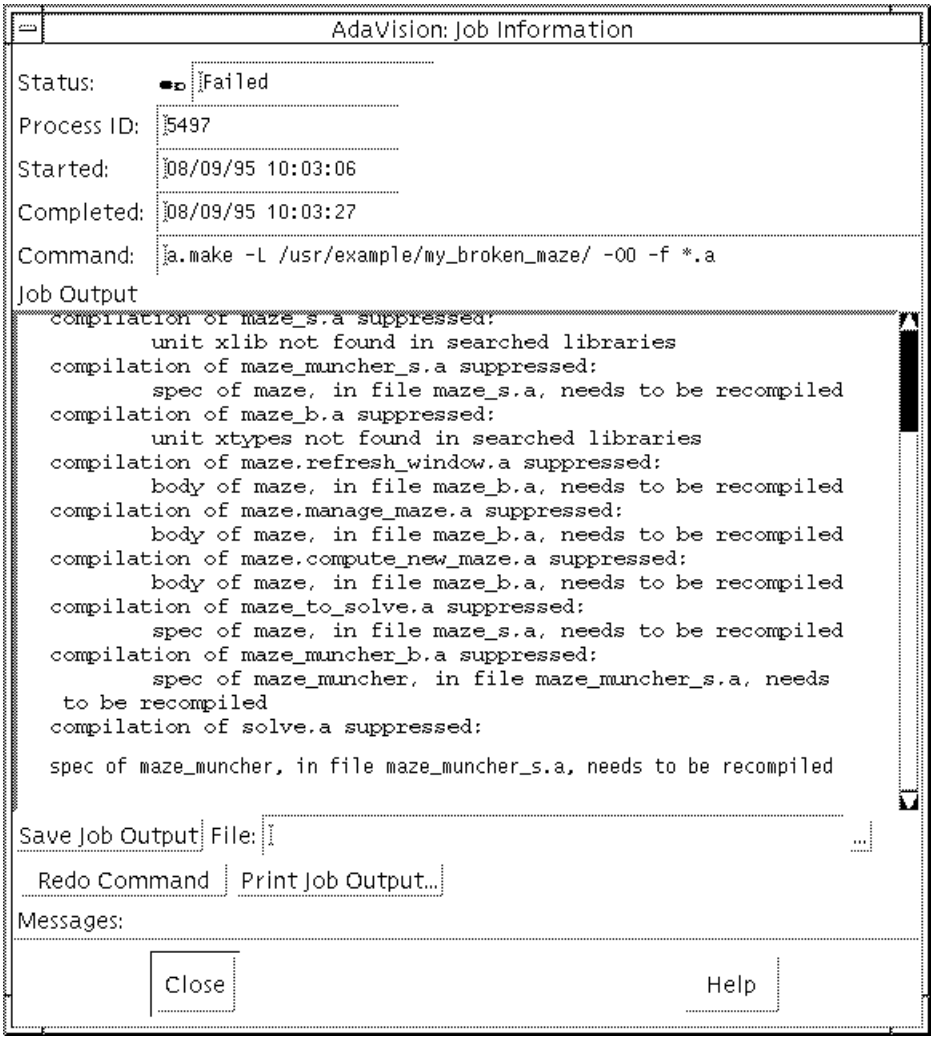


Figure 2-11 Job Information Window

2.9.2 Browsing Errors

If the Job Information window shows that a job has completed unsuccessfully, you can browse the source to get more information about the errors.

To browse errors:

1. **From the Job Status window, select an unsuccessfully completed job and click Information to open the Job Information window.**
   The source text appears in the Job Output window with errors highlighted.

2. **In the Job Output window, click on the text formatted**
   filename`: line` *x*`, char` *y*`:error`
   in the line containing the error you want to browse.
   Your editor opens with the selected error highlighted and marked with a glyph. An error description appears at the bottom of the editor window.

Figure 2-12 Error Browsing

Some error messages contain links to relevant sections of LRMTool, the online Ada language reference manual. The links are characterized by the text `RM` followed by the appropriate reference manual section number.

To link from an error message in the Job Output window to an LRMTool entry:

♦ **Click** `RM` *section number***.**

See Chapter 5, “LRMTool,” for more information on using LRMTool.

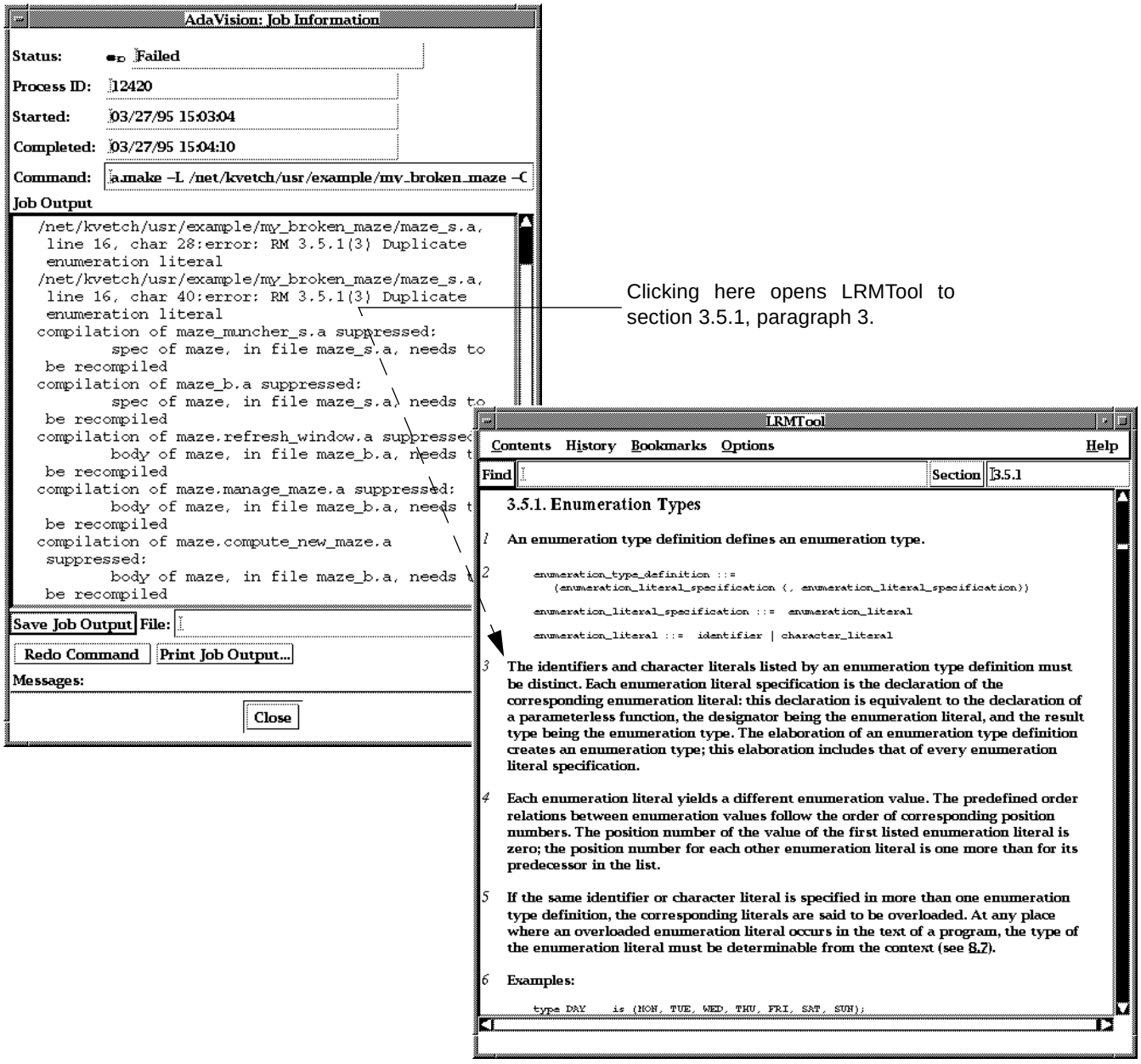


Figure 2-13 Opening LRMTool From the Job Information Window

2.10 Defining Global Options

To define global options for the AdaVision environment:

1. **From either the Unit View or the Library View, select Options ➤ Global to open the AdaVision Global Options window.**
2. **Enter the appropriate information for each option.**
3. **Click Apply or OK.**

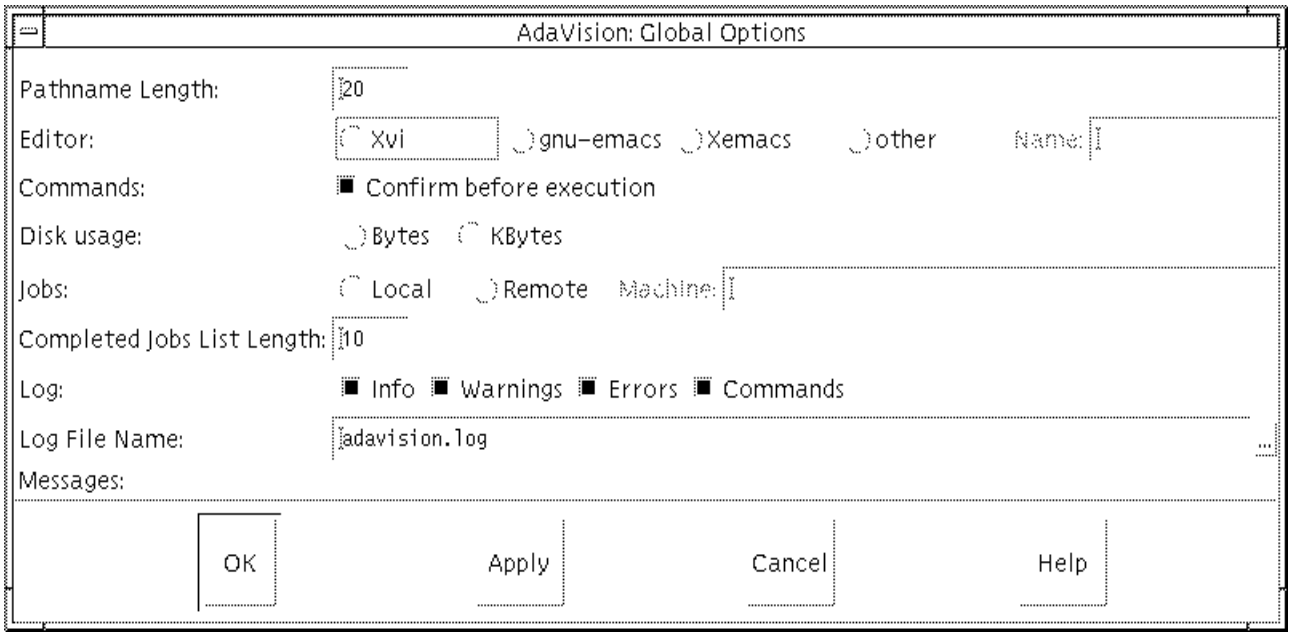


Figure 2-14 Global Options Window

In the Global Options Window you can set:

Pathname Length

Use this option to determine the maximum length of text fields in AdaVision dialogs.

Editor

Select Xvi, gnu-emacs, or Xemacs. AdaVision also allows you to specify your own editor by clicking the Other button and entering the path name of your editor in the Name text field.

Command Confirmation

If Confirm before execution is set when you invoke an action, AdaVision displays a dialog with options allowing you to make changes before the action starts.

Disk Usage Information

Choose to display your disk usage information in either bytes or kilobytes.

Job Execution Location

Set job execution to local or remote. If you choose remote, enter the name of the machine on which you execute jobs. For local execution, AdaVision uses the catalog of available releases in the file `/etc/VADS` on *swada_host*. If you switch to remote execution, AdaVision uses `/etc/VADS` on *ada_host*.

Completed Jobs List Length

Enter the number of completed jobs to appear in the completed job list.

Message Log Customization

You can choose to have any combination of info, warning, error, or command messages saved in the message log file.

Message Log File

AdaVision saves the Message Log in the file shown in the log file name field. Clicking the ellipsis (. . .) button allows you to determine the log file name using a file chooser. Options for Show Log File and Print Log File are in the Status menu.

Once you have changed and applied a setting, it is saved for the rest of the current session and remains in effect for your next session of AdaVision.

2.11 Executing Remote Jobs

You can execute jobs remotely using the `on` command. You must enable the remote execution daemon (`rexd`) in the file `/etc/inetd.conf` on each machine you intend to use remotely. You can only do so as superuser on each system. For details on the `on` command, as well as `inetd` and `inetd.conf`, see Solaris 2.4 developer documentation.

In addition, Ada libraries and source files must be mounted in Sun's distributed computing file system (NFS) on the same path name for both the local machine and the remote machine. Ada libraries should physically reside on the remote machine (compile server) for efficiency.

To enable remote jobs from AdaVision:

1. **Choose Options ➤ Global to open the Global Options window.**
2. **Set the Jobs control to Remote and enter the name of the remote machine in the text field.**
3. **Click Apply or OK.**

After you have set the Jobs control, each job you start in AdaVision will run on the chosen machine. If you subsequently change the Jobs control setting, all existing Job Status windows still keep the same setting they had when they were started.

2.12 Printing Files and Screens

AdaVision supports printing of source files of units. You can also print the entire AdaVision Unit View as it would appear in graph mode without scrollbars.

To print a source file:

1. **Select the unit whose source file you want to print.**

2. **Choose Unit ➤ Print Source to open the Print Options dialog.**

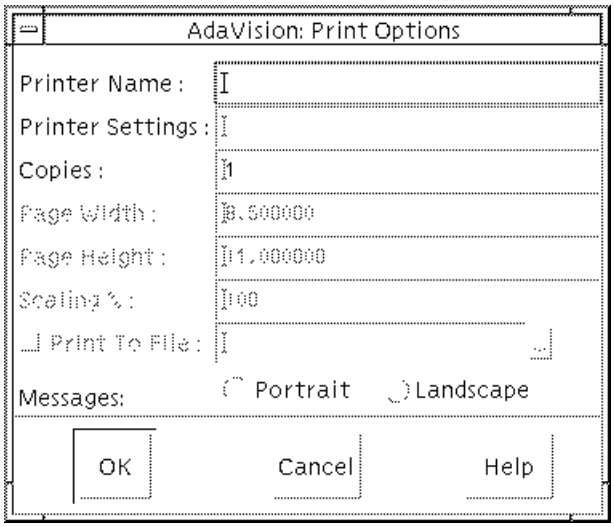


Figure 2-15 Print Options Dialog

3. **Enter the appropriate printer information in the text fields and select Portrait or Landscape.**

4. **If you want to print to a file, select Print to File and enter the file name in the text field.**
   Clicking the ellipsis (. . .) button allows you to select the file name using a file chooser.

5. **Click OK to print.**

To print the contents of the entire Unit View in graph mode:

1. **Choose Unit ➤ Print Graph to open the Print Options window**

2. **Enter the appropriate printer information in the text fields and select Portrait or Landscape.**

3. **If you want to print to a file, select Print to File and enter the file name in the text field.**
   Clicking the ellipsis (. . .) button allows you to select the file name using a file chooser.

4. **Click OK to print.**

AdaVision Libraries 3

This chapter describes how to use AdaVision to browse and manage Ada libraries. AdaVision provides tools for understanding the structure of a selected library.

3

3.1 Opening the Library View

Using AdaVision's Library View, you can perform actions on Ada libraries, such as creating, opening, closing, deleting, and cleaning.

To open the Library View:

♦ **Choose View ➤ Libraries from the Unit View window.**

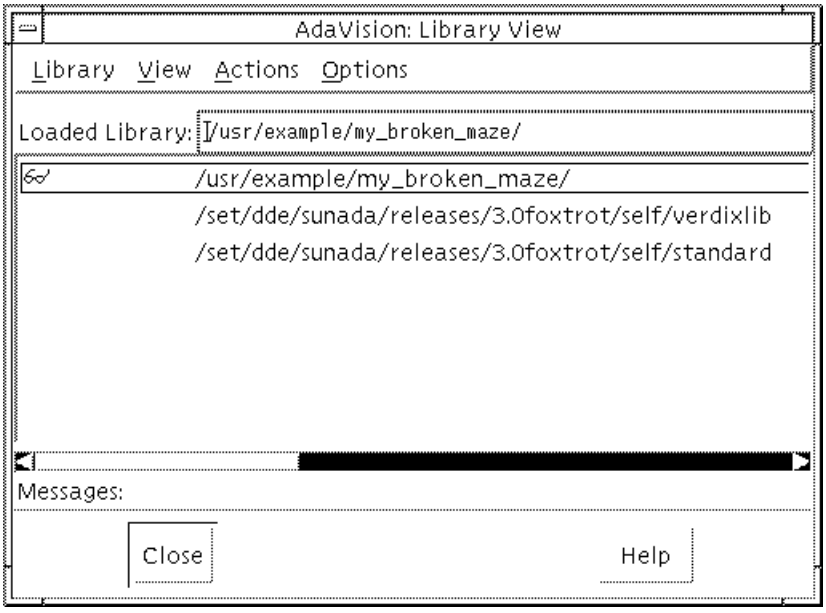


Figure 3-1 Library View Window

To close the Library View and return to the Unit View:

♦ **Choose Library ➤ Exit View from the Library View window.**

Libraries are shown by their full path names. To view library names without their paths behind them:

♦ **Choose View ➤ Short Names.**

To bring the Unit View window to the foreground if it is hidden behind other windows:

♦ **Choose View ➤ Units.**

Note – The View ➤ Units feature may not work with some settings on the Motif or Common Desktop Environment window managers.

3.2 *Loading a Library*

AdaVision bases its operation on a single Ada library. Loading a library tells AdaVision which library to base its operation on.

To load an Ada library from AdaVision:

1. **From the Library View, choose Library ➤ Load to open the Load Library dialog.**

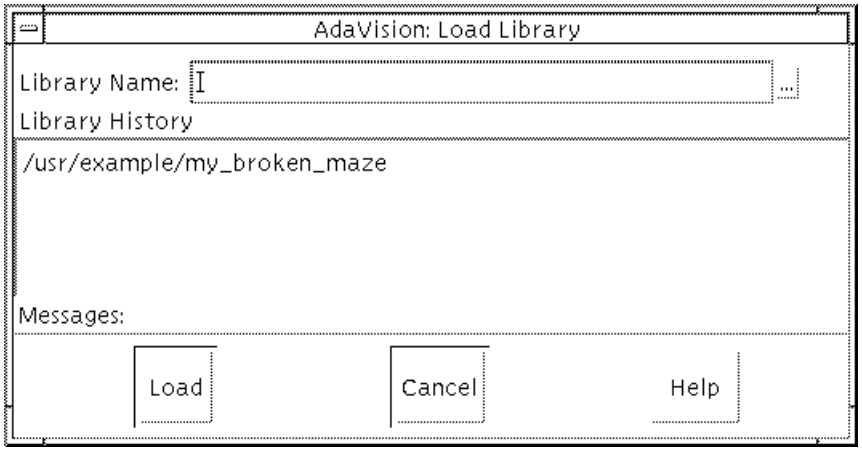


Figure 3-2 Load Library Dialog

2. **Enter the library name in the Library Name text field.**
   You can also use the file chooser by clicking the ellipsis (. . .) button.
3. **Click Load.**

When you load a library, AdaVision initializes the Library View window with the name of the loaded library followed by the names of all the libraries on its ADAPATH. AdaVision then opens the loaded library, displaying its units in the Unit View window.

3.3 Opening a Library

Opening a library allows you to browse and manipulate its units in the Unit View. To open one or more Ada libraries:

1. **Select one or more libraries from the Library View scrolling list.**
2. **Choose Library ➤ Open Selection.**
   A pair of eyeglasses is shown next to the newly opened libraries. The units in the selected libraries are added to the Unit View display.

3.4 Closing a Library

To close a library:

1. **Select one or more libraries from the Library View scrolling list.**
2. **Choose Library ➤ Close Selection.**

3.5 Creating a New Library

To create a new library:

1. **From the Library View, choose Library ➤ New.**

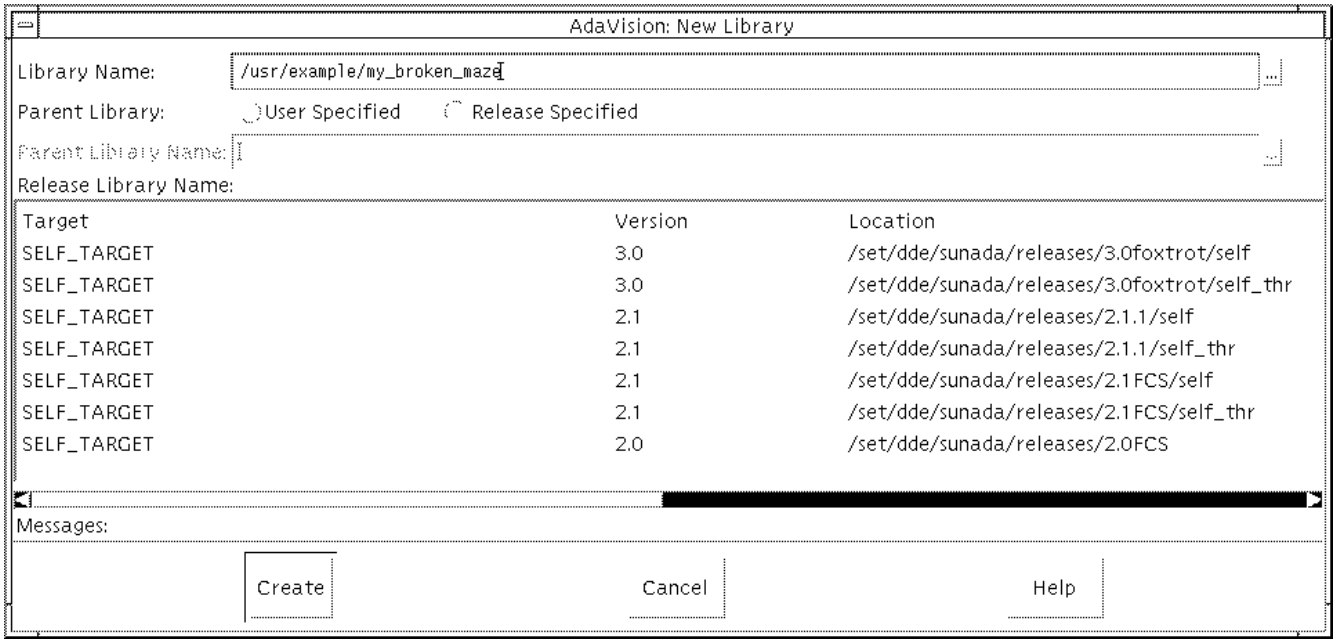


Figure 3-3 New Library Window

2. **Type the path name of the new library in the Library Name text field.**
   AdaVision provides the current directory name by default unless the directory already contains an Ada library.

3. **Choose the Parent Library type.**
   The library is either User Specified or Release Specified. If your parent library is User Specified, type the file name in the Parent Library Name text field or use the file chooser by clicking the ellipsis (. . .) button. If the present library is Release Specified, choose one of the releases shown.

4. **Click Create.**
   If no other library is already loaded, the new library is automatically loaded.

3.6 Removing a Library

When you delete a library, AdaVision performs an Ada `a.rmlib` command. This action is always accompanied by a confirm dialog, regardless of the Global Options setting on Confirm Before Execution.

To delete a selected library:

- **From the Library View, choose Actions ➤ Remove.**
  A confirm before execution dialog is displayed.

Caution – Be sure you want to delete the object(s) because there is no undo facility.

3.7 Cleaning a Library

Cleaning a library removes all compilation information contained in the selected Ada library—that is, the information in the `ada.lib`, `gnrx.lib`, and `GVAS_table`, as well as the contents of the `.imports`, `.nets`, `.lines`, and `.objects` directories. A confirm dialog appears regardless of the Global Options setting for Confirm Before Execution.

To clean a library from within AdaVision:

- **From the Library View, choose Actions ➤ Clean.**
  A confirm before execution dialog is displayed.

Caution – Be sure you want to delete the object(s) because there is no undo facility.

3.8 Importing Units to a Library

When you import units to a library, an `a.make -f *.a` command is performed on the files you specify in the Import window. The Import action compiles all named sources for dependencies with a confirm dialog appearing regardless of the Global Options setting for Confirm Before Execution. Changes you make are remembered and displayed the next time you import.

To import units to a library:

1. **From the Library View, choose Actions ➤ Import to open the Confirm Import dialog box.**

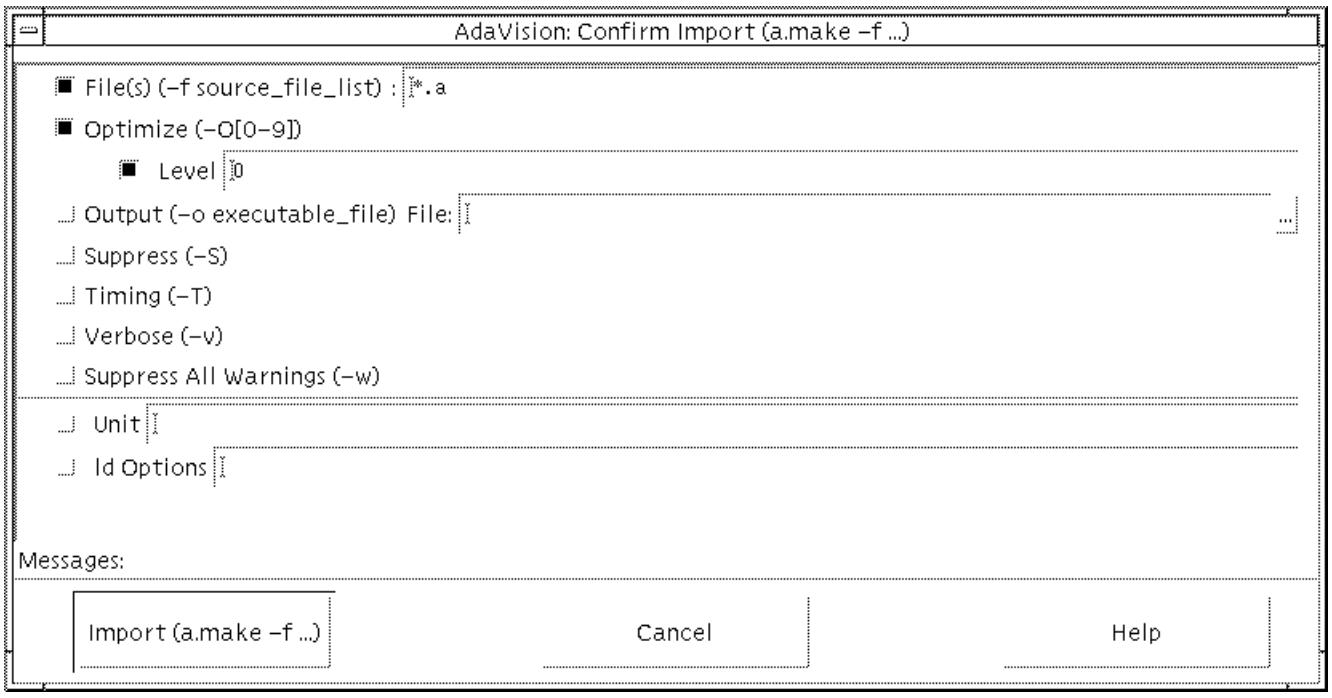


Figure 3-4 Confirm Import Dialog

2. **Modify options as needed.**
3. **Click Apply or OK.**

3.9 Updating a Library

When you update a library, AdaVision performs an `a.make` command to update all units in the library. A confirm dialog appearing regardless of the Global Options setting for Confirm Before Execution. Changes are remembered and displayed the next time you perform an update.

To update a library:

1. **From the Library View, choose Actions ➤ Update Library to open the Confirm Update Library dialog box.**

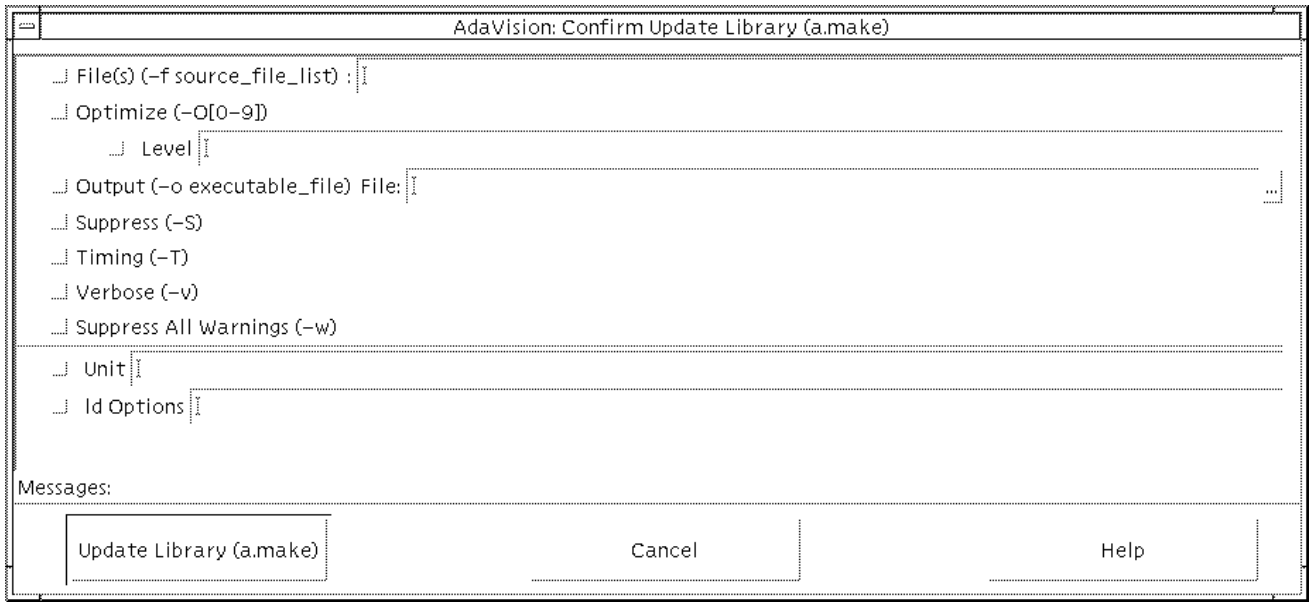


Figure 3-5 Confirm Update Library Dialog

2. **Modify options as needed in the confirmation window.**
3. **Click Update Library.**

3.10 Viewing Library Options

To view basic information about a selected Ada library:

- **From the Library View, choose Options ➤ Library to open the Library Options dialog.**
  The dialog shows the Name, Target, Version, VADS, and Category (ADAPATH, INFO Directive, LINK Directive, or DEFINE Directive) of the library.

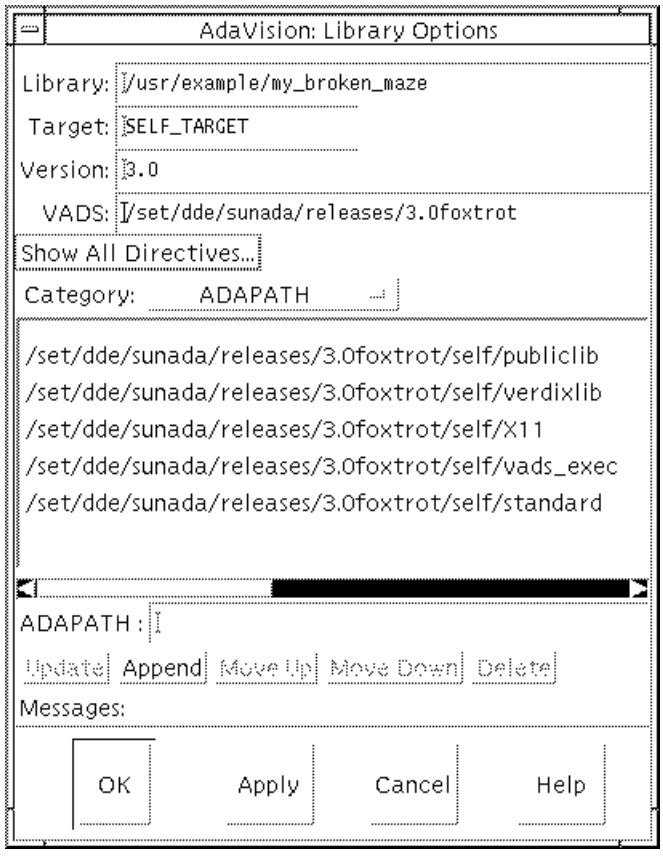


Figure 3-6 Library Options Window Showing the ADAPATH

3.11 Editing the ADAPATH

When you edit the ADAPATH for a selected library, you can add, delete, or reorder libraries in a scrolling list.

To edit the ADAPATH:

1. **From the Library View, choose Options ➤ Library to open the Library Options window.**
2. **Select ADAPATH from the Category menu.**
   The window contains a scrolling list showing the libraries in the ADAPATH.
3. **Use the buttons and the text field at the bottom of the window to edit the ADAPATH.**

3.12 Viewing and Altering Directives

Directives can exist in any Ada library on the ADAPATH. All directives are visible to the compiler or linker.

3.12.1 Viewing All Directives on the ADAPATH

To view all the directives that the compiler or linker can see:

1. **From the Library View, choose Options ➤ Library to open the Library Options window.**
2. **Click the Show ADAPATH Directives button to open the Compiler Directives window.**
   The Compiler Directives window contains a read-only composite of all directives set in all libraries on the ADAPATH.

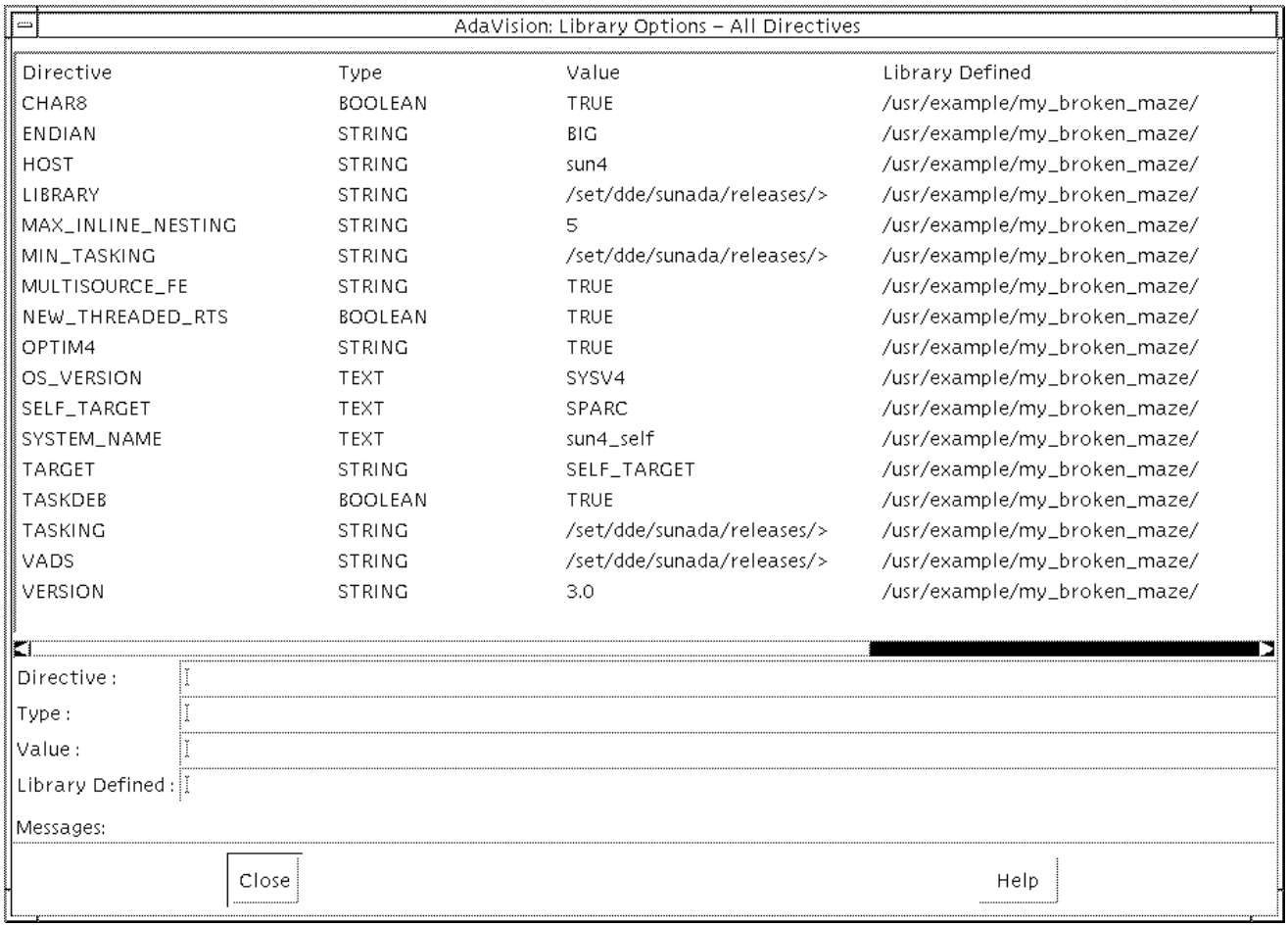


Figure 3-7 Compiler Directives Window

3.12.2 Viewing and Altering Directives of a Single Library

To view a list of INFO, LINK, or DEFINE directives defined in a selected library:

1. **From the Library View, choose Options ➤ Library to open the Library Options window.**

2. **Select INFO Directives, LINK Directives, or DEFINE Directives from the Category menu to open the appropriate list of directives.**

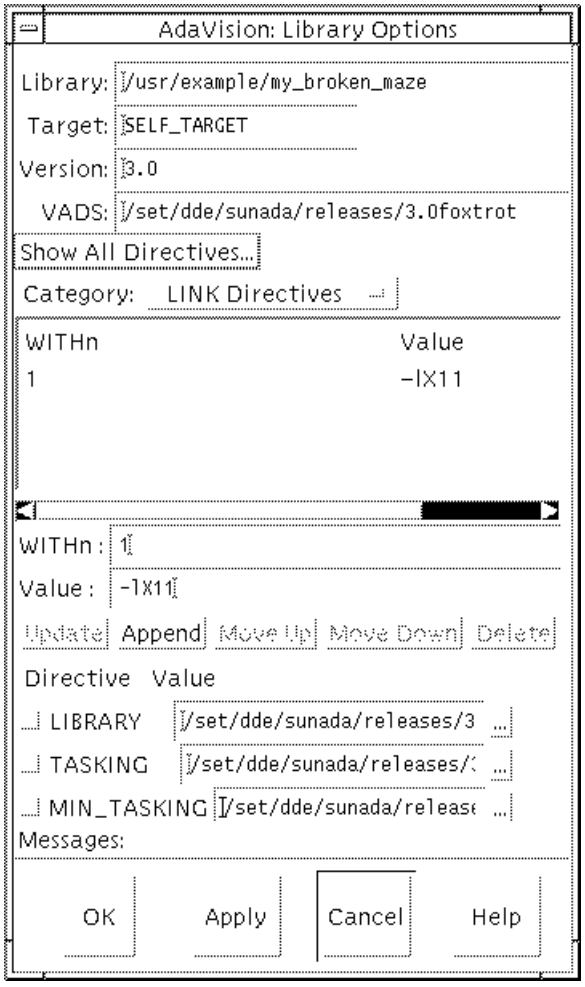


Figure 3-8 Library Options Window Showing LINK Directives

The controls for the options windows vary, depending on whether you are using the self-target or cross-development.

To add, delete, or change a directive:

1. **Select a displayed directive.**
2. **Update the characteristics as needed.**
3. **Click Apply or OK to return to the Library View window.**

3

AdaDebug 4

This chapter explains how to debug programs with AdaDebug, a full-featured, window-based, source-level debugging tool.

4.1 Introducing AdaDebug

AdaDebug is a window-based and object-based interface to the SPARCompiler Ada command-line debugger, `a.db`. AdaDebug makes extensive use of OSF/Motif GUI features and facilities while preserving the full range of `a.db` functionality.

In AdaDebug, you browse and debug programs in a Program View window. For multitasking programs, AdaDebug also provides a Task Inspector and multiple Task View windows. In both Program View and Task View windows, you can issue debugging commands by selecting menu items. You can also

issue `a.db` commands in the Program View window from a command line. AdaDebug supports a separate Program I/O window for entering program input and displaying program output.

Upon opening, AdaDebug displays the Program View window, which shows you the program from the top-down perspective of its main subprogram. The Program View window is an OSF/Motif base window, so you can resize it, move it, or close it to an icon. You can also start more than one AdaDebug session. Each session has its own Program View window.

You can customize AdaDebug properties in the Options menu. This menu is the same for both the Program View and Task View windows.

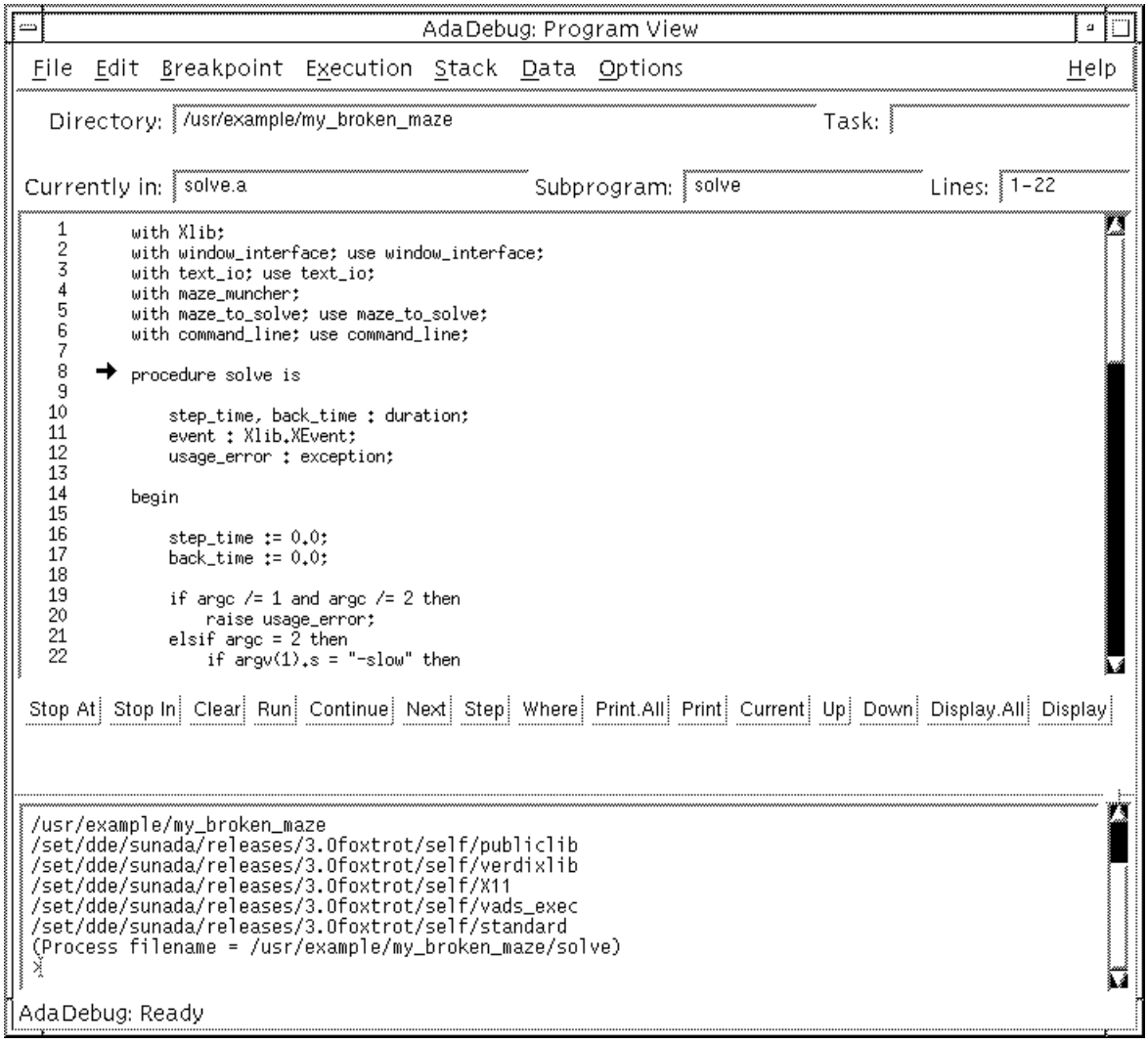


Figure 4-1 Program View Window

Status Area
Contains the following information:

- **Directory**
  The current working directory.
- **Currently in**
  The file, subprogram, and range of lines currently displayed.
- **Stopped in**
  The file, subprogram, and line number at which the program is currently stopped.
- **Task**
  The task currently being executed.

Next Line to be Executed
Line marked with an arrow indicating the next line to be executed. This line is also known as *current focus*.

Button Bar
Buttons allowing you to execute frequently used commands without using the pulldown menus.

Command Line
Area in which to enter `a.db` commands. Refer to the SPARCompiler Ada documentation for the various `a.db` commands.

4.2 Starting AdaDebug

You can start AdaDebug from a command line or from AdaVision.

To run AdaDebug from a command line:

♦ **Type** `adadebug` [*options*] [*program_name*] `&`

You can start AdaDebug with or without specifying which program to load. If you specify a program name, AdaDebug displays the file containing the main subprogram source code in the Program View. If you do not supply an Ada program name as an argument, AdaDebug is invoked with an empty Program View window.

AdaDebug accepts all the command line arguments that the SPARCompiler Ada debugger, `a.db`, accepts, including:

- `-L` (to specify a library name)
- `-C` (to debug an executable with a corresponding core file)
- `-a` and `-ag` (to debug a running process)
- `-c` (to debug C code)

Aside from its own options, AdaDebug also supports most of the generic OSF/Motif options. All other options that you specify in the command line are passed to `a.db`, the underlying Ada debugger.

Note – The option commands for OSF/Motif are described in the man pages for `mwm` and `dtwm`.

To start AdaDebug from AdaVision:

1. **Select a program executable from the AdaVision Unit View.**
2. **Choose Tools ➤ AdaDebug or press the AdaDebug button.**

4.3 Loading a Program Into AdaDebug

If you do not specify a program to debug when you start AdaDebug, you can load a program by starting a debugging session from within the Program View.

You can load a program on top of one already being debugged.

To start a debugging session from within Program View with a new or different program:

1. **Choose File ➤ Load Executable to open the Program Loader window.**
   The file chooser displays only files with execute permission.

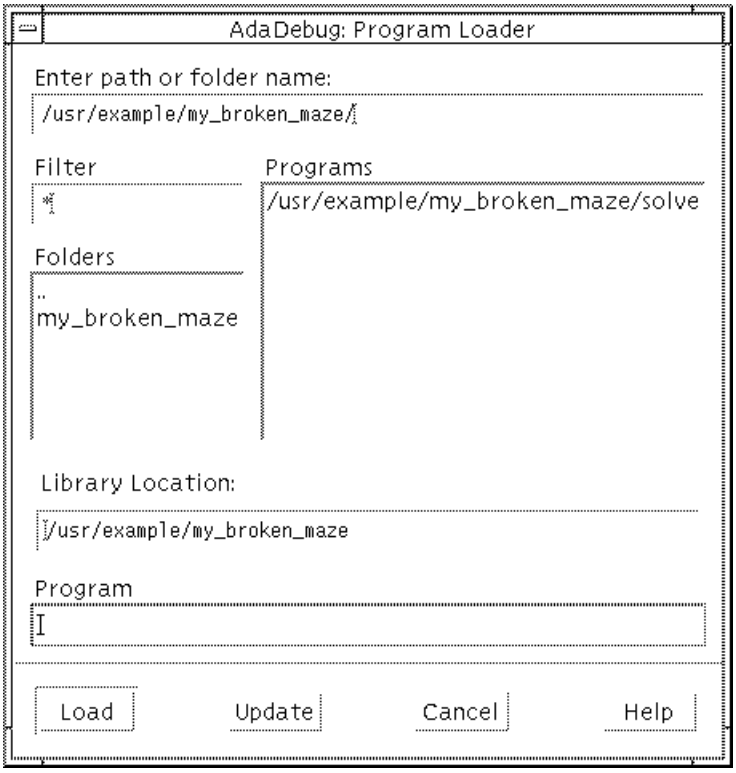


Figure 4-2 Program Loader Window

2. **Type the path name of the program you want to debug in the Program text field or select it from the file chooser list.**
   Double clicking on the file name from the file chooser list acts as an accelerator and loads the program.

3. **Optionally, enter the path to the top level Ada library used to build the executable file.**

4. **Click Load.**
   AdaDebug replaces the current program, if any, with the new one.

If you would like to use remote execution to debug an OSF/Motif application, you must first set your environment for remote display. Be sure to use the `-h` option for host name. Once you have started AdaDebug, you cannot switch back and forth between local and remote execution.

4.4 Opening a Subprogram

To view a subprogram in the Program View source display dialog:

1. **Choose File ➤ Open Subprogram to open a pop-up window containing a list of the subprograms in the executable being debugged.**
   The list may not contain every subprogram in the executable.

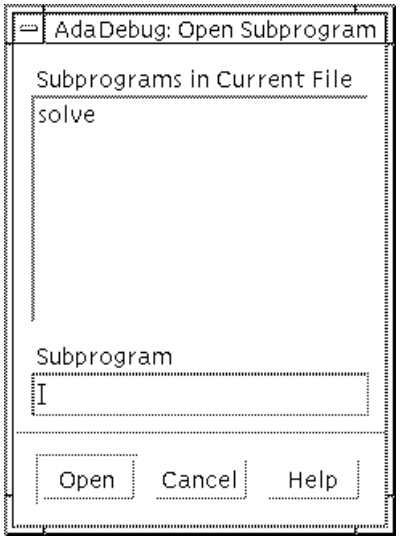


Figure 4-3 Open Subprogram Dialog

2. **Choose the subprogram you want to view from the possible names list or type its name in the Subprogram text field.**
   AdaDebug replaces the current subprogram, if any, with the new one.

3. **Click Open.**
   The source for the subprogram is then loaded in the Program View.

If there are two or more subprograms by the same name, AdaDebug opens the Overload window. Listed in the output dialog are the overloaded subprograms and their parameters.

4.5 Editing a File

To edit a file in the Program View source dialog:

1. **Choose File ➤ Open to open the Open Filename window.**
   The current file is displayed in the Selection text field.

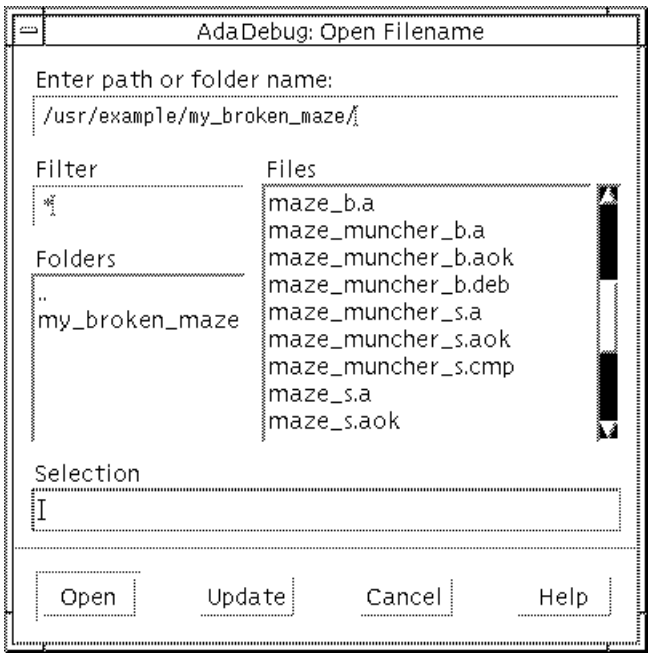


Figure 4-4 Open Filename Window

2. **Click Open to display the selected file in the Program View.**

To edit a file in your editor of choice:

1. **Choose File ➤ Edit to open the Edit window.**
   The current file name is displayed in the Selection text field as a default entry. The currently selected editor is displayed in the Editor text field.

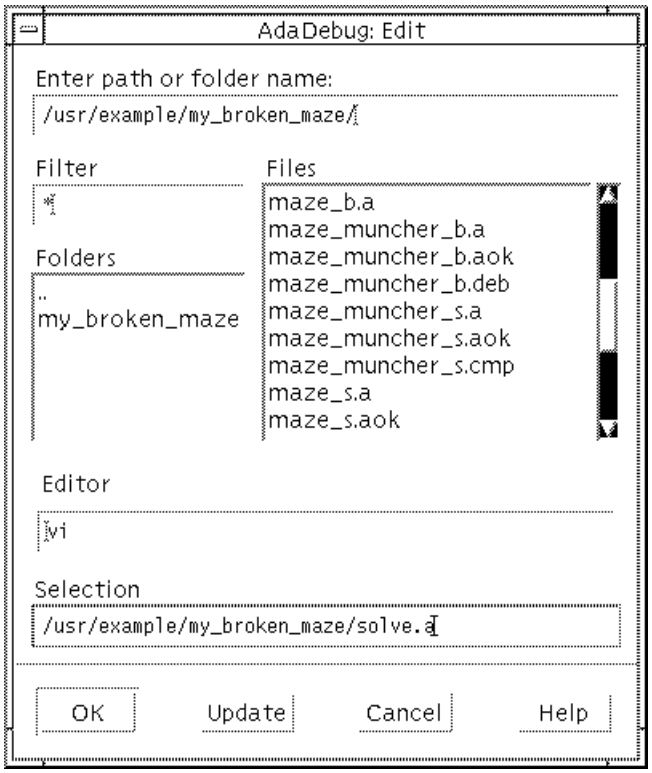


Figure 4-5 Edit Window

2. **Press the OK button to load the source file into the editor of choice you specify.**

Note – Once invoked, the editor operates as an independent tool and runs even after you quit AdaDebug. You must quit the editor separately.

4.6 Controlling Program Execution

To run a program in AdaDebug:

- **Choose Execution ➤ Run (or click the Run button).**
  This runs your program with no arguments.

To run a program with arguments:

1. **Choose Execution ➤ Run With Arguments.**

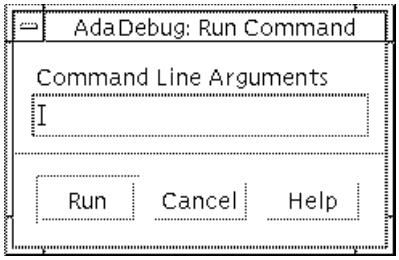


Figure 4-6 Run Commands Dialog

2. **Type the arguments in the Command Line Arguments text field.**
3. **Press Run.**

4.6.1 Continuing Program Execution

After a program has stopped at a breakpoint, choose Execution ➤ Continue to continue program execution from the location of the current breakpoint to the next breakpoint or end of program.

Note – To restart the program from the beginning, use Run instead of Continue.

4.6.2 Stepping Over Subprogram Calls

To single-step from one statement to the next without stepping into subprogram calls:

♦ **Choose Execution ➤ Next (or click Next).**

Each time you choose Next, AdaDebug executes the next statement. If the next statement includes a subprogram call, then AdaDebug executes the call and stops at the next statement after returning from the call.

4.6.3 Stepping Into Subprogram Calls

When you use Step, AdaDebug executes the next statement. However, if the next statement includes a subprogram call, Step displays the source code of the subprogram call and points to the first statement in the call. You can then step through the call, statement by statement.

To step into a subprogram:

♦ **Choose Execution ➤ Step (or click Step).**

Since Ada subprograms are often declared in separate source files, AdaDebug switches the file it is displaying to bring into view the subprogram that the program calls. The control fields—Stopped in and Subprogram—also change, if necessary, to show the new file and subprogram names.

4.6.4 Stopping Execution

The Stop menu item suspends program execution at the statement that is executing. The location where the program stops is therefore indeterminate.

Stop is useful when you want to stop the program to break out of an infinite loop without killing the program and the debugger.

To suspend program execution:

♦ **Choose Execution ➤ Stop (or click Stop).**

Pressing Control-C in the command line is the same as using Stop.

4.6.5 Program I/O Window

AdaDebug automatically pops up a Program I/O window when:

- A program writes to standard output
- User input is required while a program is running

Some programs that run in their own windows can also use the standard I/O facilities to write information to the display. Having a separate window for displaying program I/O makes it easier to keep track of what is happening while you are debugging a program.

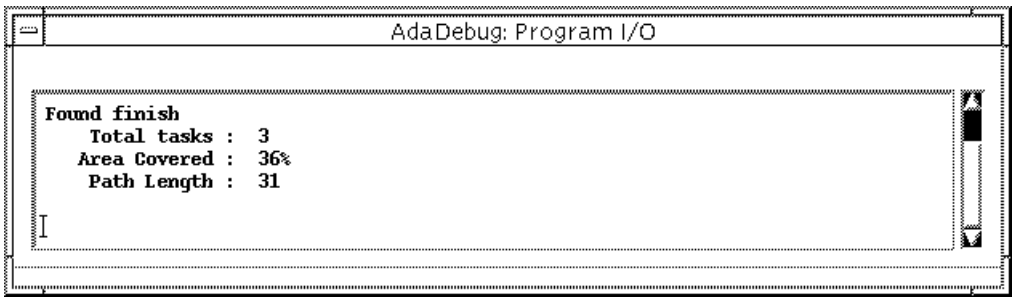


Figure 4-7 Program Input/Output Window

Note – If your program requires user input, AdaDebug may appear to hang. It is actually awaiting an input event.

4.7 Setting Breakpoints

By choosing Breakpoint ➤ Stop At <*selected line*> (or clicking the Stop At button), you can set breakpoints that stop the program when it reaches the line where the breakpoint is set. While in source mode, you can specify a breakpoint at a line or in a subprogram. While in instruction mode, you can also set a breakpoint at an instruction. Breakpoints can also be set at all exceptions in an Ada program.

A breakpoint set in a specific Task View applies to that task only. All other tasks that execute the same line of code continue as though there were no breakpoint. A breakpoint set in the Program View applies equally to all tasks.

Each breakpoint is marked with a glyph STOP representing a stop.

4.7.1 Setting a Breakpoint at a Specific Line

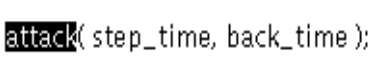
To set a breakpoint at a specific line while in source mode:

1. **Select any part of the line where you want to set the breakpoint.**
2. **Choose Breakpoint ➤ Stop At <*selected line*> (or click Stop At).**

4.7.2 Setting a Breakpoint in a Subprogram

To set a breakpoint in a specific subprogram while in source mode:

1. **Select the subprogram name, as shown.**
   Do not select the parentheses and parameters.



2. **Choose Breakpoint ➤ Stop In <*selected subprogram*> (or click Stop In).**

4.7.3 Setting a Breakpoint at an Instruction

To set a breakpoint for an instruction while in instruction mode:

1. **Select any part of the line where you want to set the breakpoint.**
2. **Choose Breakpoint ➤ Stop At (or click Stop At).**

4.7.4 Setting a Breakpoint at any Exception

AdaDebug can set breakpoints at any exception in a program before it runs. You can, for example, check possible errors while the program is running.

To set breakpoints at all exceptions:

- **Choose Breakpoint ➤ Stop At Any Exception.**

4.7.5 Setting a Breakpoint Upon Subprogram Return

You can set either a temporary or a permanent break upon a subprogram return.

- Temporary sets a temporary breakpoint that is valid only for one call of the subprogram.
- Permanent sets a breakpoint that remains set across calls of the subprogram.

4.7.5.1 Setting a Temporary Breakpoint

To set a temporary breakpoint:

♦ **Choose Breakpoint ➤ Stop At Return (temporary).**
Execution stops when the current subprogram returns, but the breakpoint is then cleared.

4.7.5.2 Setting a Permanent Breakpoint

To set a permanent breakpoint:

♦ **Choose Breakpoint ➤ Stop At Return (permanent).**
Execution stops every time the current subprogram returns.

4.7.6 Clearing a Breakpoint

To clear a single breakpoint:

1. **Select any part of the line containing the breakpoint you want to clear.**
2. **Choose Breakpoint ➤ Clear At <*selected line*> (or click Clear).**

To clear all breakpoints:

♦ **Choose Breakpoint ➤ Clear All Breakpoints.**

4.7.7 Listing Breakpoints

AdaDebug lists all the breakpoints currently set in a Breakpoints pop-up window. The Breakpoints window displays numbered entries. Each entry contains the following information for the breakpoint:

- Breakpoint ID
- Name of the file containing the unit where the breakpoint is set
- Line number
- Unit name

To see a list of all the breakpoints set in a Program View window:

♦ **Choose Breakpoint ➤ List Breakpoints.**

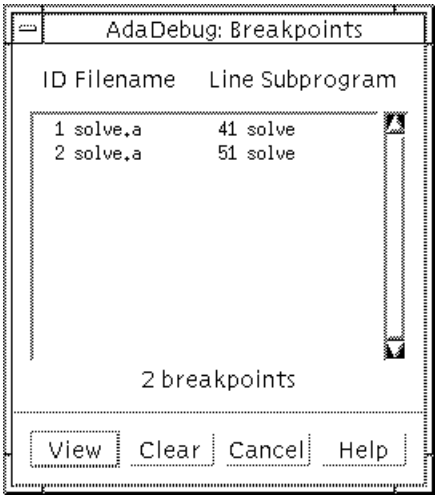


Figure 4-8 Breakpoint List

To view a breakpoint listed in the window:

♦ **Double-click on the breakpoint entry or select the breakpoint entry and click the View button.**

To clear a breakpoint that is listed in the window:

♦ **Select the breakpoint entry and click the Clear button.**

4.8 Call Stacks

The Ada debugger uses a call stack to represent the current state of a program during debugging. The call stack represents all currently active subprograms—that is, subprograms that have been called, but which have not yet returned to their callers.

The subprogram that is executing when the program halts at a breakpoint or after a single step is at the *bottom* of the stack. The subprogram that called the currently executing subprogram is pushed up one level on the stack. The subprogram that called the second-level subprogram is said to be pushed to the third level, and so on. Each position on the call stack is called a *stack frame*.

In a non-tasking program, the main subprogram is always at the top of the stack.

4.8.1 Inspecting Stacks

Each Task View has its own stack frame. To examine a call stack in AdaDebug:

♦ **Choose Stack ➤ Inspector to open the Stack Inspector window.**

The Stack Inspector lists each stack frame in its own row. Each entry shows the call level, line number, and subprogram name of the stack frame. The Stack Inspector window also shows the parameters for each subprogram to help you differentiate between subprograms with overloaded names.

The stack frame displayed in a Stack Inspector window corresponds to the Program View or Task View window from which the Stack Inspector was opened.

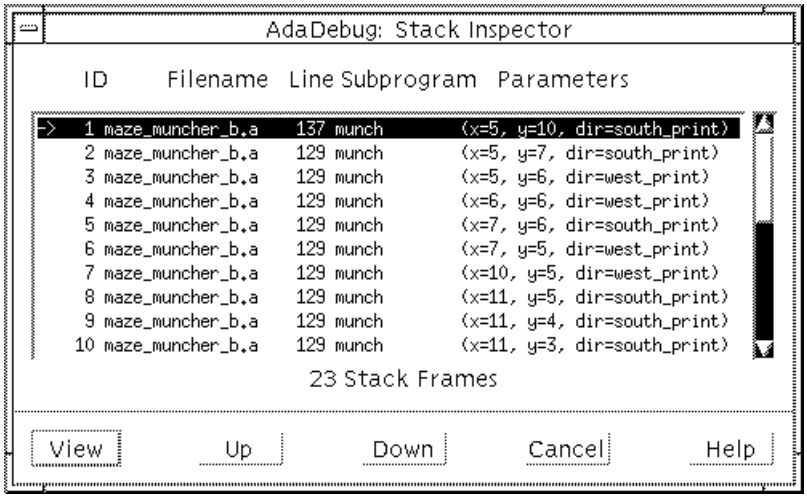


Figure 4-9 Stack Inspector Window

4.8.2 Viewing Code Associated With a Stack Frame

You can navigate the stack frames by choosing the menu items located under the Stack menu or by using the Stack Inspector window.

Table 4-1 contains descriptions of the Stack menu items:

Table 4-1 Stack Menu Items

Menu Item	Description
Up a Frame	Displays the call at the next higher level on the stack—the “caller”
Down a Frame	Displays the call at the next lower level on the stack—the “callee”
Top Frame	Displays the call at the top of the stack—the main subprogram
Bottom Frame	Displays the call at the bottom of the stack—in Program View, the currently executing subprogram

4.8.3 Displaying Code From the Stack Inspector Window

To display code associated with a stack frame entry in the Stack Inspector:

- **Select a stack frame entry and click the View button.**

When you display code associated with a second-level or lower-level stack frame, AdaDebug marks the location of the start of that code with a stack frame glyph.

A solid arrow glyph ➜ points to the code associated with the frame at the top of the stack. A hollow arrow glyph ⇨ points to code associated with other frames in the stack.

4.8.4 Expressions

AdaDebug allows you to evaluate variables and expressions while a program is executing. You can evaluate a variable or expression on demand or continuously update variables or expressions as the program reaches breakpoints or is stepped through.

4.8.4.1 Evaluating an Expression

To evaluate an expression in AdaDebug:

1. **Select the expression in the Program View (or Task View).**
2. **Choose Data ➤ Evaluate <*selected expression*> to display the value of the selected expression in the command line at the bottom of the window.**

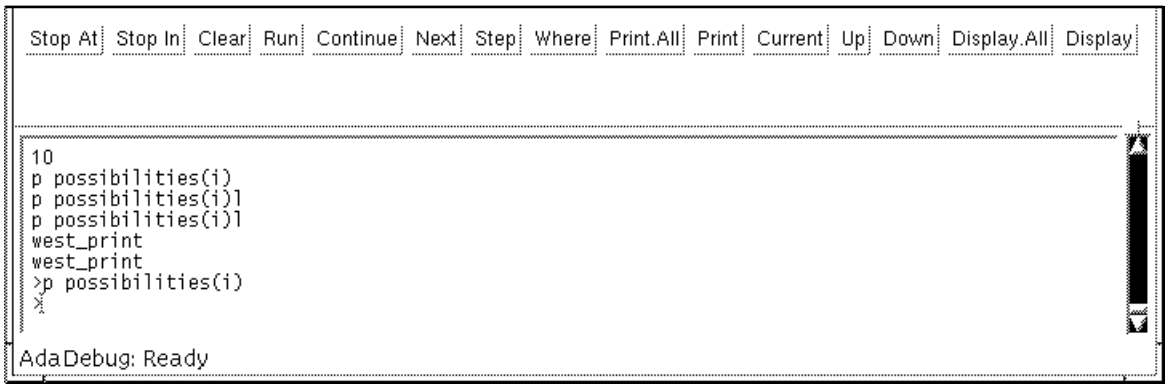


Figure 4-10 Expressions Evaluated in the Command Line

4.8.4.2 *Displaying an Expression*

To display an expression in AdaDebug:

1. **Select the expression in the Program View (or Task View).**
2. **Choose Data ➤ Display <*selected expression*> to display the value of the selected expression in the Data Inspector window.**

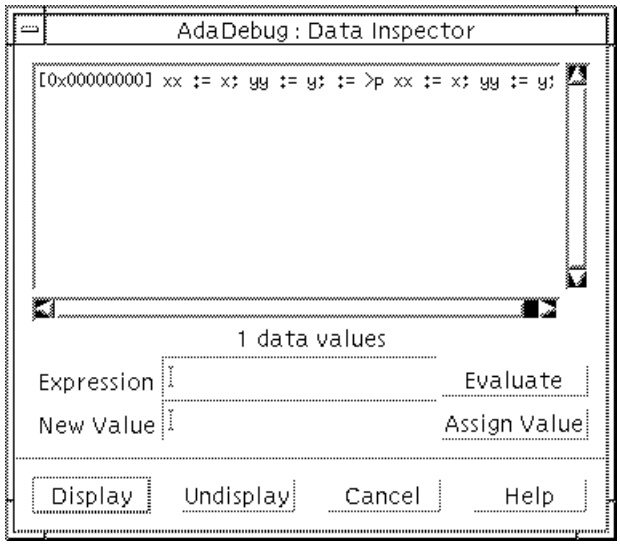


Figure 4-11 Data Inspector Window

The expression is updated as breakpoints are reached and the program is stepped through.

4.8.4.3 *Turning Off Display of an Expression*

To turn off the display of an expression currently in the Data Inspector window:

1. **Select the expression in the Data Inspector window.**
2. **Choose Data ➤ Undisplay <*selected expression*>.**

4.8.4.4 *Viewing Register Values*

To view register values:

1. Choose Data ➤ Register Inspector.

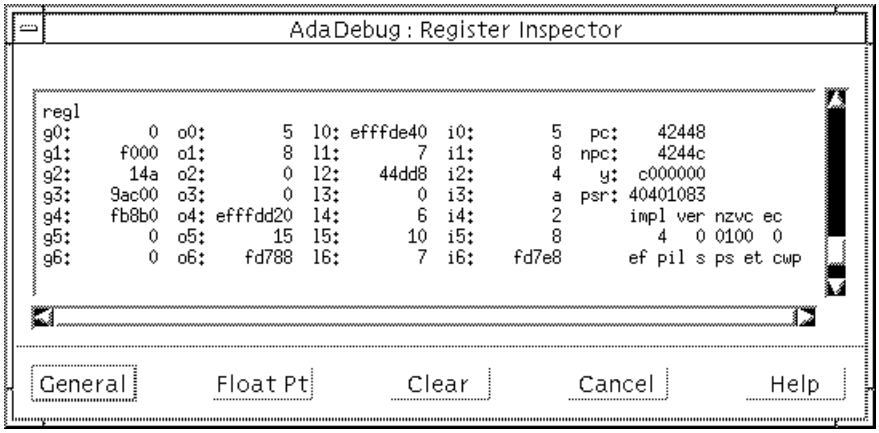


Figure 4-12 Register Inspector Window

2. Click on the General button to display general registers or the Float Pt button to display floating point registers.

4.9 *Task Inspector*

If more than one task is active in your program when it stops at a breakpoint, you can examine and debug the tasks as separate threads of control using the Task Inspector.

The Task Inspector displays a list of all of the active tasks, using icons to represent the tasks. The display of task objects in the Task Inspector gives you a global view of a program as a collection of Ada tasks.

4.9.1 Opening the Task Inspector

The Task Inspector displays tasks, *not* task units. To open the Task Inspector:

♦ **Choose Execution ➤ Task Inspector**

In Figure 4-13, the Task Inspector window displays task objects active while debugging `solve`, the executable in the `maze` example.

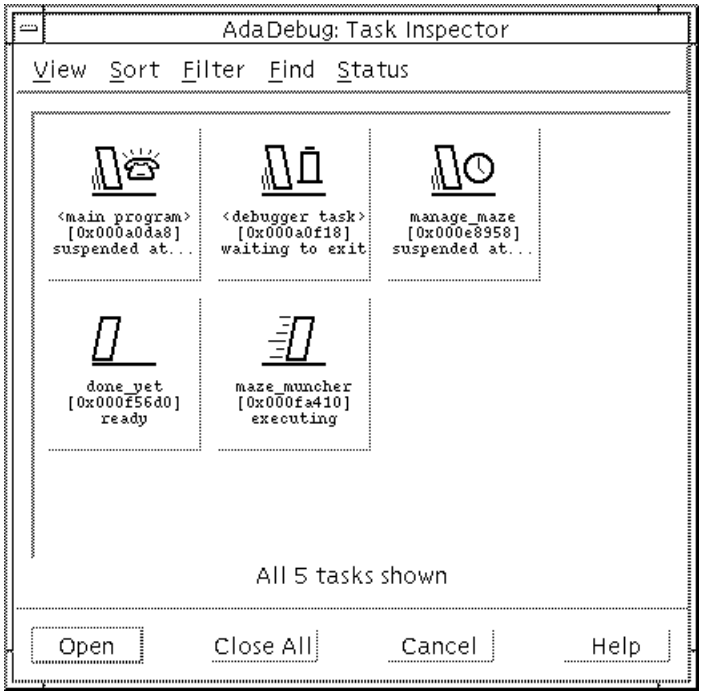


Figure 4-13 Task Inspector Window

4.9.2 Task Icons

The Task Inspector represents each task as a task object, displaying an icon which reflects the task object itself along with the task name and status.

You can sort and filter task objects by state or name and search for tasks that match a specified pattern. No special icons exist for substates.

The following list identifies and briefly describes each icon and the task state it represents. Some icons represent two different task states.

`attempting rendezvous`

The task is attempting a rendezvous with a called task.

`awaiting activations`

The task is activating child tasks.

`awaiting terminations`

The parent task is suspended, waiting for its child tasks to terminate.

`called in rendezvous`

For fast rendezvous, this task state occurs when a rendezvous is executed by the calling task.

`completed`

The task has executed all of its code body. Upon completion of all child tasks, the task terminates.

`destroyed`

The task has been terminated and is in the process of being destroyed.

`executing`

The task is currently executing.

`finished passive call`

The task, suspended at a passive call, has been resumed. The guard for the called entry has changed from closed to open. Alternatively, if the task was waiting on an ABORT_SAFE condition variable, the condition variable has been signaled.

`finished rendezvous`

The task has just finished its rendezvous with a called task.

`in rendezvous`

The task is in rendezvous with the called task.

`not yet active`

The parent has created the task, but the program has not yet activated it.

`ready`

The task is on the run queue, ready to execute.

`ready to start`

The task is on the run queue, ready to start its first execution.

`suspended at accept`

The task has executed an `accept` statement on an entry; it is waiting for a task to call that entry.

`suspended at call`

The task has executed an entry call and will remain in this state until transition to the `in rendezvous` state.

`suspended at delay`

The task has executed a `delay` statement.

`suspended at fast accept`

The task has executed a “fast” `accept` (one without a `do...end` sequence of statements) on an entry; it is waiting to be called.

`suspended at passive call`

The task is suspended calling a passive task’s entry whose guard is closed or the task is waiting on an `ABORT_SAFE` condition variable.

`suspended at select`

The task has executed a `select` but no tasks are calling open entries. The task is waiting until some event allows it to proceed.

`suspended at select (terminate not possible)`

The task has executed a select but no tasks are calling open entries; the select statement has an open terminate alternative and the task is waiting until some event enables it to proceed. The task’s terminate conditions are not satisfied; it is waiting for child tasks to terminate.

`suspended at select (terminate possible)`

The task has executed a select but no tasks are calling open entries; the select statement has an open terminate alternative and the task is waiting until some event enables it to proceed. The task's terminate conditions are satisfied.

`terminated`

The task's execution has terminated.

`waiting for interrupt`

The task created for the interrupt vector is waiting to be signalled by its interrupt handler. After being signalled, the task calls the attached ISR. For a Solaris or POSIX thread, the task is blocked at a sigwait() for the attached UNIX signal.

`waiting for signal`

The task created for the interrupt entry is waiting to be signalled by its interrupt handler. After being signalled, this task does an entry call to the interrupt entry in the attached task.

`waiting to exit`

The `main program` task is suspended while waiting for its child tasks to terminate.

4.9.3 Changing the View of Task Icons

To display tasks as rows of icons:

♦ **In the Task Inspector window, choose View ➤ Icon.**

To display tasks in a textual list:

♦ **In the Task Inspector window, choose View ➤ List.**

4.9.4 Sorting Tasks

Table 4-2 shows how the Sort menu item in the Task Inspector window sorts task objects by number, state, or name.

Table 4-2 Task Sorting in the Task Inspector Window

To sort tasks:	**Choose:**
Numerically by task number	Sort ➤ By Number
Alphabetically by state name	Sort ➤ By State
Alphabetically by name	Sort ➤ By Name

4.9.5 Filtering Tasks

To filter tasks from the display list that are in selected state(s):

1. **Choose Filter ➤ By State in the Task Inspector window to open the Filter Tasks pop-up window.**

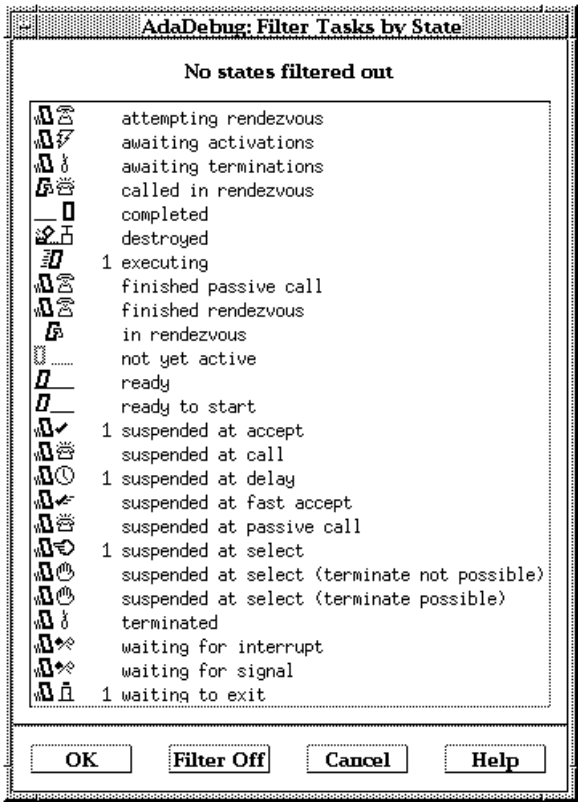


Figure 4-14 Filter Tasks Window

2. **In the list, select the state(s) for which you want corresponding tasks removed from the display.**
   Pressing Control while you are selecting allows you to select multiple states; pressing Shift allows you to select a range of states. The number to the right of an icon indicated how many tasks are at that state.

3. **Press OK.**

4.9.6 Searching for a Task

When you have a large number of tasks in the Task Inspector window, you can use the Search facility to search for a specific task by name.

To search for a task:

1. **Choose Find ➤ Enter Pattern in the Task Inspector window to open the Find Pattern popup window.**

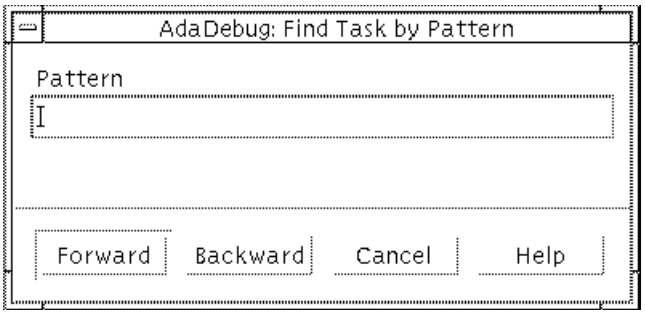


Figure 4-15 Find Task by Pattern Window

2. **Enter the name or pattern.**
3. **Press either the Forward or Backward button.**
   Pressing the Return key activates a search forward, the default.

Each task is selected when it is found.

4.9.7 Displaying Task Status

To display information about a selected task:

- **Choose Status ➤ Tasks from the Task Inspector window to open a Task Status window.**

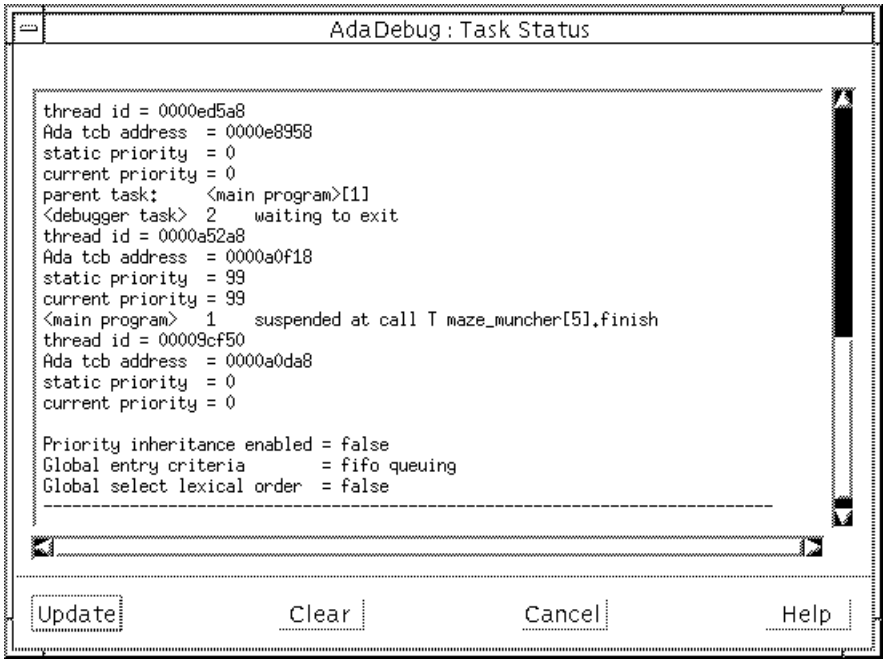


Figure 4-16 Task Status Window

This window displays the following information:

- Name, address, and status of the tasks
- Name and status of each entry, and whether tasks are waiting on a particular entry
- Static and current priority value of the task
- Information about the parent task: the parent task name and address, the file name containing the parent task, and the line number at which the parent task declaration is located in the source file

Whether or not tasks are executing when you stop a program at any particular point depends on the control flow of the program.

4.10 Task View Window

The Task View window displays the code that a selected task executes. You can open a Task View window for each task.

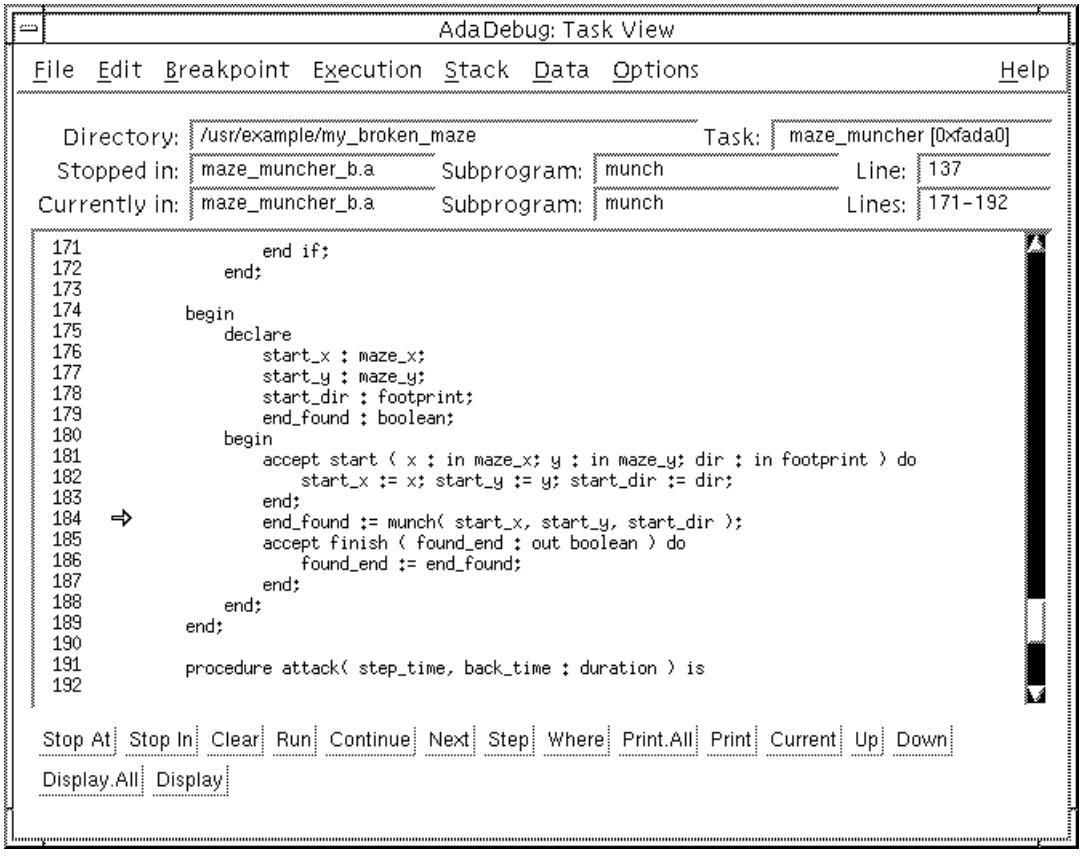


Figure 4-17 Task View Window

4.10.1 Debugging Using Task Views

From Task View windows, you can view and debug a program from the multiple perspectives of the specific tasks the program has active at any one time. Working in these windows, you can manipulate each thread of control independently of the others.

Open the task in a Task View window by doing either of the following:

- Double-click on its icon in the Task Inspector.
- Click on the task icon and press Open.

A Task View window looks and behaves just like a Program View window except that, in a Task View window, debugging commands apply only to the single thread of control represented by the selected task.

Program View is actually a special case of Task View. Program View always displays the currently executing task. Thus, in some circumstances, you can view the currently executing task object in both the Program View and Task View windows simultaneously. For instance, if you open a task and set a breakpoint in the Task View window when (and if) the program executes that task it will stop at that task-specific breakpoint. At this point, AdaDebug is displaying the same task in both the Task View and Program View windows.

4.10.2 Creating a Task

Tasks become available to the Task Inspector from the time the program first allocates storage for them. Once a program creates a task object, you can open the Task Inspector to display the task objects.

You can place a breakpoint at the `begin` statement in a task body. When you run the program, it reaches this breakpoint when a task first attempts to execute the task body statements. Because you placed this breakpoint in the task body, the breakpoint applies to each and every task that executes that task body. At this point, you may wish to set a breakpoint at one or more specific tasks using the Task View window.

4.10.3 Terminating a Task

If a task has no dependent task(s), it terminates when it has completed its execution. For details on the conditions under which a task is said to have completed its execution, refer to the *Ada Language Reference Manual*.

4.10.4 Main Program Task

In Ada programs, the main program is itself executing within a task, so the Task Inspector always displays an object for it with angle brackets around its name to indicate its special status. When all other tasks have terminated and are no longer displayed, the main task object continues to be displayed in the Task Inspector. The *Ada Language Reference Manual* provides more details on the life history of tasks.

4.11 Building Executable Files

You can build your executable file without exiting the debugger.

To build an executable file:

♦ **Choose File ➤ Build.**

4.12 Entering Information in Command Lines

You can enter an `a.db` line command in the command line, the area below the Program View window's source display dialog.

To enter an `a.db` line command from AdaDebug:

1. **Click at the command line to obtain the input focus.**
2. **Type the line command at the prompt.**
3. **Press Return.**

In addition to accepting all the `a.db` commands, the command line accepts some `dbx` commands. However, the command parameters and pattern arguments must conform to `a.db` commands. Only the most basic `dbx` commands are aliased.

The following table lists the commands in AdaDebug aliased from `a.db` commands.

Table 4-3 `dbx` Commands and Corresponding `a.db` Commands

`dbx` **Command**	`a.db` **Command**	**Description**
`bsearch` *pattern*	`?`*pattern*	Searches backwards for a pattern
`cont`	`g`	Continues executing
`delete` *parameters*	`d` *parameters*	Deletes breakpoints
`dis` *parameters*	`li` *parameters*	Lists disassembled instructions
`down` *parameters*	`cu` *parameters*	Moves down a frame on the call stack
`file` *parameters*	`e` *parameters*	Loads a file into the debugger
`frame` *parameters*	`cu` *parameters*	Moves down a frame on the call stack
`listi` *parameters*	`li` *parameters*	Lists disassembled instructions
`print` *parameters*	`p` *parameters*	Displays the value of a variable or expression
`run` *parameters*	`r` *parameters*	Runs the loaded program
`search` *pattern*	`/`*pattern*	Searches forward for a pattern
`status`	`lb`	Lists the breakpoints
`step up`	`bd`	Moves up a frame on the call stack
`stop at` *parameters*	`b` *parameters*	Sets a breakpoint
`stop in` *parameters*	`b` *parameters*	Sets a breakpoint
`stopi at` *parameters*	`bi` *parameters*	Sets a breakpoint on an assembly instruction
`use`	`set`	Shows debugger parameters
`use` *parameters*	`set source` *parameters*	Sets source search path
`up` *parameters*	`cd` *parameters*	Moves up a frame on the call stack
`where`	`cs`	Shows call stack

4.13 Setting AdaDebug Options

The Options menu contains controls for a number of features you can customize. AdaDebug saves all option settings except the Instruction Mode once they have been modified. Table 4-4 summarizes the options you can set for AdaDebug.

Table 4-4 Options for AdaDebug

Options for AdaDebug	
Line Numbers Displayed	Specifies whether to display line number
Searches Case-Sensitive	Specifies whether the `a.db` search command should ignore or respect uppercase/lowercase distinction
Instruction Mode	Specifies whether to debug at source level or machine instruction level
Debugging Session Logging	Specifies whether this session will be logged in the filename entered in the text field
Source Search Path	Specifies a search path name for use with AdaDebug
Font Selection	Specifies the font used by AdaDebug
Decimal Mode	Displays decimal numerical output as specified when evaluating expressions
Hexadecimal Mode	Displays hexadecimal numerical output as specified when evaluating expressions
Octal Mode	Displays octal numerical output as specified when evaluating expressions
Program I/O Window	Specifies whether the Program I/O Window appears by default

LRMTool 5

The *Reference Manual for the Ada Programming Language* for ANSI/MIL-STD-1815A-1983—known informally as the *Language Reference Manual*, or LRM—contains the complete, official definition and description of the Ada language.

LRMTool presents the manual online as one continuous document in an OSF/Motif base window. LRMTool consists of a set of user-interface features and facilities to help you navigate the document.

5.1 Starting LRMTool

You can start LRMTool from a command line, from the AdaVision Tools menu, or while error browsing in AdaVision. LRMTool accepts all the command line arguments of a Motif/X application.

LRMTool depends on the file `$SWADAHOME/lib/lrmbook.html`, which contains the LRM document in HTML format.

To start LRMTool from within AdaVision:

- **Choose Tools ➤ LRM.**
  LRMTool displays the first page of the manual. LRMTool closes when you exit AdaVision.

To start LRMTool while error browsing in AdaVision:

- **In the Job Information window, click on the linked text beginning with *RM* that corresponds to the error you are looking at.**
  LRMTool opens to the section number indicated, with the paragraph number in parentheses. LRMTool closes when you exit AdaVision.

Figure 5-1 Opening LRMTool from the Job Information Window

To start LRMTool from a command line:

- **Type** `lrmtool &`
  LRMTool displays the first page of the manual. LRMTool runs on your workspace for as long as you want, independent of the other SPARCworks/Ada tools.

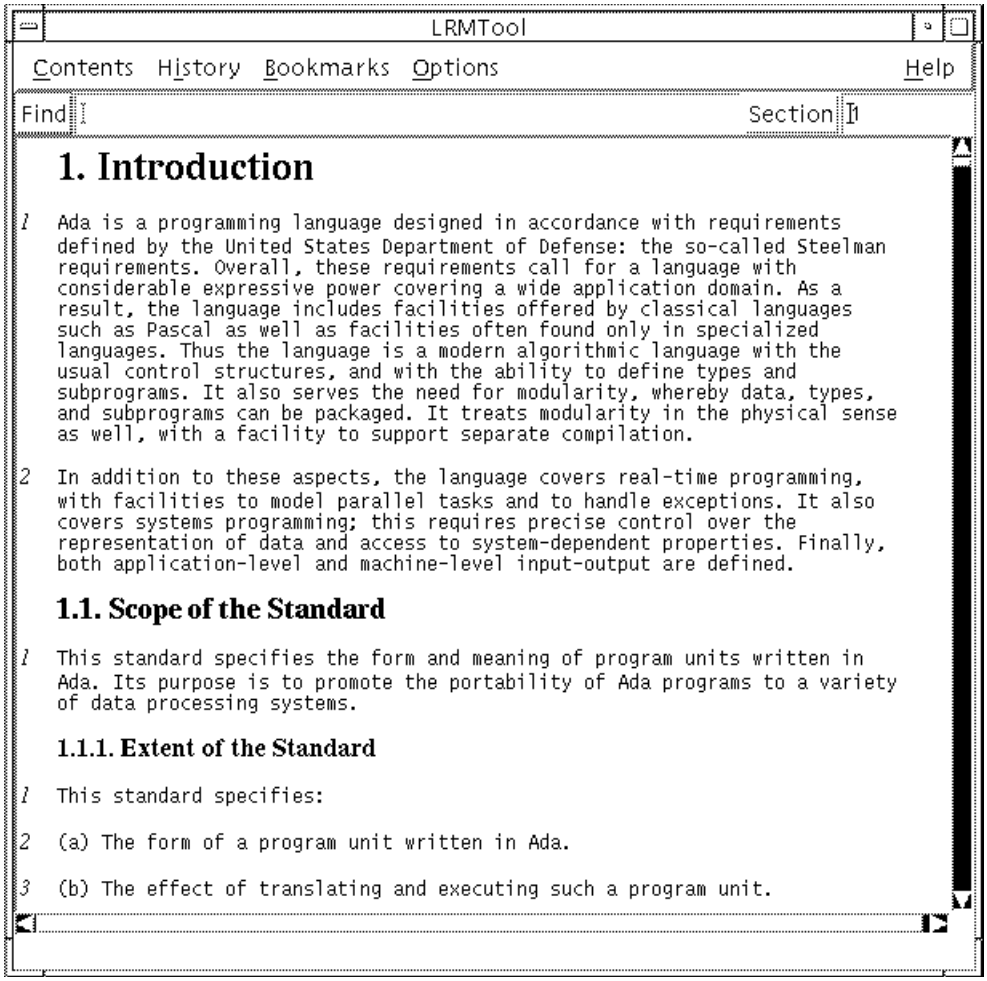


Figure 5-2 LRMTool Base Window

5.2 Contents Menu

The LRM is divided into numbered chapters, sections, subsections, appendices, and an index. The chapters and appendixes appear in the Contents menu, with menus of sections and subsections appearing as you pull progressively to the right.

When you choose an item from the Contents menu, LRMTool displays that section. It also records the move on the History menu.

The Contents menu lists chapters, appendixes, and the index on the first-level menu. Sections within each chapter are listed as items on submenus. In turn, subsections within sections are listed on submenus.

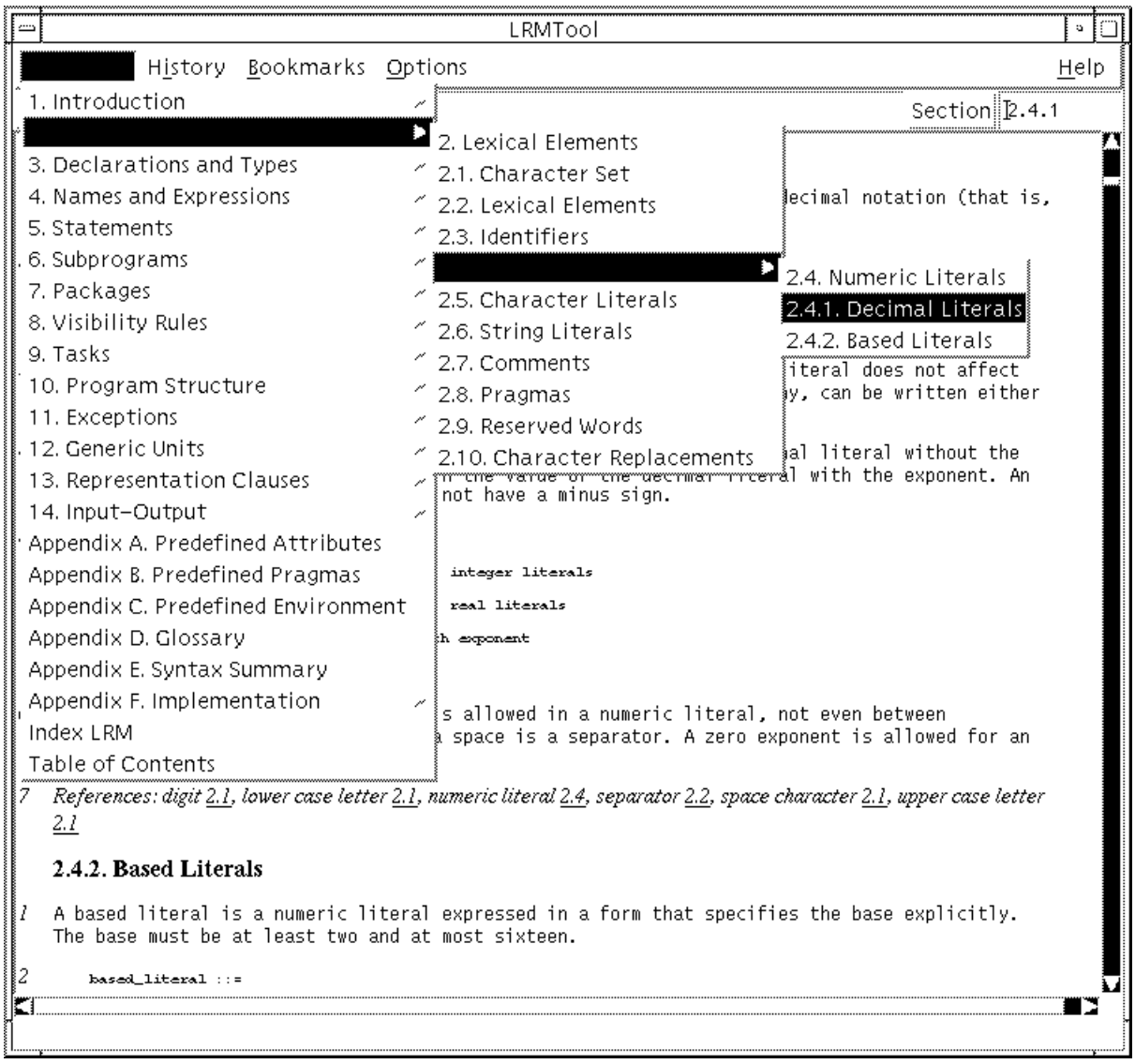


Figure 5-3 Contents Menu

5.3 History Facility

Each time you display a different section of the manual, History records your movements, with the current location listed at the top. You can return to any of the locations recorded in a given session by clicking on them in the History menu.

5.4 Finding Words or Phrases

The Find facility locates the first or next occurrence of any phrase you enter in its editable text entry field. The search for strings is *not* case-sensitive.

To use Find:

1. **Type the expression in the text field located to the right of the Find button.**
2. **Press the Find button or the Return key.**
   To find the next occurrence of the expression, press the Find button or the Return key again.

5.5 Finding Sections by Numbers and Appendix Letters

You can use the Section button to display a section or appendix by entering the section number or appendix letter in the Section text field.

Section interprets an entry that corresponds to a section number or appendix letter (uppercase A through F) as a request to display the corresponding section or appendix. For example, if you enter *A* in the Section text field, LRMTool displays Appendix A, not the next occurrence of that uppercase character.

The editable Section text field displays the current section, even when not used for searches.

5.6 Adding Bookmarks

Adding a bookmark for a section saves that section as an item in the Bookmarks menu for the current session as well as future sessions. When you click on the Bookmarks menu item for that section, LRMTool scrolls to it.

To add a bookmark for a section of the manual:

1. **Verify that you are at the desired section, as listed in the Section text field.**
2. **Choose Bookmarks ➤ Add to Bookmarks.**

5.7 Index Entries

All technical terms used in the manual are listed in the index.

The three types of index entries are:

- *Section* entries, which list one or more section numbers; the definition of the term is located in the first section number listed. Additional section numbers refer to sections with related information.
- *See* entries, which appear in brackets [ ] and contain words or phrases that the index references using either different terms or the same terms in a different order.
- *See also* entries, which appear in brackets [ ], alert you to other index entries that refer to sections with related information.

5.8 Linking Text

LRMTool creates links from index and reference entries to the sections to which the entries refer. Clicking on a link replaces the text in the current view with the linked text.

5.8.1 Showing and Hiding Links

To toggle between LRMTool showing links in blue and showing them as normal text:

- **Choose Options ➤ Show Links.**

5.8.2 Choosing Style of Link Underline

To choose the style of link underline:

♦ **Choose Options ➤ Anchor Underlines ➤** *underline_style***.**

5.9 Changing the Font

You can change the font of the text in the LRMTool window.

To change the font size:

♦ **Choose Options ➤ Font ➤** *desired_font***.**

5

Index

C

D

E

G

I

J

L

M

O

P

R

S

T

U

V

W