SPARCworks/Ada User's Guide

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

Part No.: 800-7732-12
Revision C, August 1994

Contents

Figures

Tables

Preface

SPARCworks™/Ada is the OPEN LOOK®, 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*—referred to as the *Guide*—shows Ada developers how to use the suite of SPARCworks/Ada tools.

In this 2.0 release, SPARCworks/Ada consists of six tools:

- AdaVision, a library browser and workplace
- AdaDebug, a GUI for the SPARCompiler Ada debugger
- EditTool, a SPARCompiler Ada-specific, extended version of the OpenWindows™ Text Editor
- LRMTool, a window-based, on-line version of the standard *Reference Manual for the Ada Programming Language*©[1]
- FileMerge, a GUI for merging source files and coordinating source code changes
- SPARCworks Manager, an OPEN LOOK window interface for managing and coordinating tools

Collectively, the release of SPARCompiler Ada and SPARCworks/Ada is known as SPARCworks Professional Ada.

1. Copyright 1983 U.S. Department of Defense

For simplicity, in most places in this *Guide*, SPARCompiler Ada is referred to as SW 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.

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. To read about the underlying Ada toolset, see the SPARCompiler Ada documentation listed on page xxii.

OpenWindows and OPEN LOOK

SPARCworks/Ada tools run on the OpenWindows workspace. OpenWindows is based on the OPEN LOOK GUI specification. This *Guide* does not attempt to document the OPEN LOOK GUI specification or the OpenWindows workspace. To read about OPEN LOOK and the OpenWindows Desktop, see the manuals listed under "Other Documentation" on page xxii.

Since SPARCworks/Ada takes full advantage of OPEN LOOK, the instructions presented in this *Guide* on how to do things in SPARCworks/Ada often include references to the specifics of the OPEN LOOK GUI. All SPARCworks/Ada tools conform to OpenWindows and OPEN LOOK user interface standards. To learn how to use the SPARCworks/Ada interface is to learn how to use most aspects of all conforming OpenWindows applications.

In this *Guide*, the OPEN LOOK default settings for mouse buttons are used:

- Left button—SELECT
- Middle button—ADJUST
- Right button—MENU

You can change the mouse button assignments in an OPEN LOOK UI implementation.

Hardware and Software Requirements

Before you can run SPARCworks/Ada 2.1, the hardware and software configuration for your system must meet the following requirements:

- SPARC® architecture workstation or server
- Solaris® 2.2 or higher operating system
- OpenWindows 3.0
- SPARCompiler Ada 2.1
- SunTech_License floating license server software

For instructions on how to install each of these items, see the documentation that accompanies each of them.

SPARCworks/Ada Installation

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

SPARCworks/Ada Documentation

Chapter 1 of this *Guide* provides a conceptual overview of the SPARCworks/Ada tools. Chapters 2 through 4 describe AdaVision. Chapters 5 and 6 explain AdaDebug functionalities. Chapters 7 and 8 discuss how to use LRMTool and EditTool.

Appendix A consists of some implementation details for the advanced user. The Glossary explains the terminology used in this *Guide*, most of which is specific to SPARCompiler Ada or SPARCworks/Ada.

Two additional tools might be helpful to you: SPARCworks Manager and FileMerge; they are described in their respective manuals:

Managing SPARCworks Tools

Merging Source Files

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	UNIX Bourne and Korn shell prompt
`#`	Superuser prompt, all shells	Superuser prompt, all shells

In written instructions on how to use an item on a submenu, an arrow points from the parent menu item to the child item. For example, the following instruction means to choose the item This File from the submenu for Print:

Choose Print ⟹ This File

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 the *Installing SunSoft Developer Products on Solaris* manual.

*On-Line Release Notes (*README*)*

The software release includes on-line 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/Ada2.1/README
/opt/SUNWspro/SWAda2.1/README
/opt/SUNWspro/GAda2.1/README
```

On-Line Manual Pages

This software release includes on-line manual pages for all SPARCworks Professional Ada products. Refer to the manual, *Installing SunSoft Developer Products on Solaris*, as to how to install these manual pages.

Magnify Help

Magnify Help™, commonly known as on-line help, is available for SPARCworks/Ada. See ‘Magnify Help for SPARCworks/Ada Tools,” on page 2-12, for details.

SPARCompiler Ada Documentation

SPARCompiler Ada and related documents and their part numbers are:

SPARCompiler Ada User's Guide (801-4861-11)

SPARCompiler Ada Programmer's Guide (801-4862-11)

SPARCompiler Ada Reference Guide (801-4863-11)

SPARCompiler Ada Runtime System Guide (801-4864-11)

GXV-Ada Programmer's Guide (801-4884-11)

Multithreading Your Ada Applications (801-4865-11)

Product Notes

An accompanying Product Notes document contains information not included in the SPARCworks Professional Ada documentation set at the time it went to press.

Other Documentation

The following documentation is cited as reference:

SPARCworks/Ada Tutorial (801-6174-11)

The Annotated Ada Reference Manual, ANSI-MIL-STD-1815A-1983 (Annotated) 2nd Edition

OPEN LOOK Graphical User Interface 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. SPARCworks/Ada 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. SPARCworks/Ada tools are designed for use with the SPARCompiler Ada compiler and toolset. SPARCworks/Ada tools give you the functionality of many separate Ada tools in a small number of tightly integrated tools. In this release, SPARCworks/Ada consists of six tools:

- SPARCworks Manager
- AdaVision
- AdaDebug
- LRMTool
- EditTool
- FileMerge

This chapter presents a high-level, conceptual overview of SPARCworks/Ada. For a practical introduction to working with SPARCworks/Ada tools, go directly to one of the other chapters that describes how to use a particular tool.

This chapter is organized into the following sections:

1

1.1 SPARCworks/Ada Tools

This section introduces the individual SPARCworks/Ada tools: SPARCworks Manager, AdaVision, AdaDebug, LRMTool, EditTool, and FileMerge.

1.1.1 SPARCworks Manager

SPARCworks Manager icon

You can start the SPARCworks/Ada 2.0 tools separately in a Command or Shell Tool or from the SPARCworks Manager palette, the OPEN LOOK window interface to SPARCworks. SPARCworks Manager provides a unified means to start SPARCworks tools and control the programming environment.

Tools are displayed in icon form in a palette. Each tool can be started by a double-click on its icon or by dragging the icon and dropping it onto the Desktop workspace. A tool that is started from SPARCworks Manager can be controlled as part of a SPARCworks Manager session.

In this section, you are shown how to start SPARCworks/Ada from SPARCworks Manager.

Note – To learn how to start the SPARCworks/Ada tools in a Command or Shell Tool, refer to the subsequent chapters describing each of the tools in detail.

1.1.1.1 SPARCworks Manager Startup

To start SPARCworks Manager:

- **Type the following command at your system prompt:**

```
% sparcworks &
```

If you are already using SPARCworks Manager as part of another SunSoft language development environment and have customized a configuration file for the toolset, you can add the SPARCworks/Ada tools to that configuration. *Managing SPARCworks Tools* describes the steps you should take for customization.

Another method to set up for convenient startup is to add an alias to your `$HOME/.cshrc` file. For example, if you use the C shell, add the following line to the `.cshrc` file:

```
alias sparcworks "sparcworks &"
```

Once this alias is in your `$HOME/.cshrc` file, to start SPARCworks in the future, you need to type only the command `sparcworks` at the system prompt.

1.1.1.2 User Interface in SPARCworks Manager

All the buttons, menus, and pop-up windows in the SPARCworks Manager palette apply to all the icons being displayed, including those for SPARCworks/Ada.

For example, you can choose View ⟹ Compact Palette to show a minimalist version of the base window or resize the window to display the icons horizontally or vertically. You can also use the Session menu to open, close, hide, or show the tools.

For a complete description of the user interface for SPARCworks Manager, see *Managing SPARCworks Tools*.

1.1.2 AdaVision

AdaVision icon showing the library name, `example`

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 programmer's main workplace for editing, compiling, and debugging Ada units and programs

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

1.1.2.1 Unit-Based Ada Programming Environment

AdaVision is primarily a *unit-based*, as opposed to *file-based*, tool because the Ada language is itself primarily unit-based. Though the Ada units which make up an Ada library reside in files, AdaVision recognizes Ada units within files. AdaVision creates and displays a graphical object for each Ada unit. Generally, you do not have to know which file a unit is in to work with it in AdaVision.

However, it is best when structuring your files to put only one compilation unit in a file for the following reasons:

- 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.
- SPARCworks/Ada operations are performed most efficiently in a one-unit-per-file environment.

1.1.2.2 AdaVision as a Library and Unit Browser

Developers use AdaVision to browse existing Ada libraries or to build new ones, to view dependencies among units, and to locate and select the units with which to work.

AdaVision organizes its views of a library into three viewing *modes*. The top-level mode, called the *library* mode, shows you a list of related Ada libraries. Typically, the list includes the library in the current directory and any other libraries on its ADAPATH. The other two modes are the *library unit* mode (LU mode) and the *compilation unit* mode (CU mode). LU mode displays an object for each LU in the selected library. CU mode displays all of the units in the selected library.

You control which libraries are displayed in the AdaVision display pane. From this list, you select the library you want to browse or work with. You select one library at a time.

1.1.2.3 AdaVision as a Programming Workplace

AdaVision is more than just a browser. It is also 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.

1.1.2.4 AdaVision Views and Modes

The three viewing modes—library mode, LU mode, and CU mode—are the main organizational features of AdaVision. Views of Ada libraries and their units are tightly coordinated with functionality: the functionality presented is tailored to what is appropriate for each of the three modes.

In each mode, AdaVision displays objects representing elements appropriate to the level of viewing. For instance, in LU mode, you can display tree graphs of the units that compose an LU object. In CU mode, you can perform “What If” (I compile this unit) compilation scenarios.

After starting AdaVision, you first select a library from the library mode list of libraries that AdaVision knows about. Then you switch to either LU mode or CU mode to see the units in the library.

To view or edit a particular unit in a library, you first select the library, and then choose the menu item for the operation you want to perform. For example, one way to start a browse or an edit session is to click SELECT on the icon and choose Open Lib Unit from the File menu to open its spec (default), body, or subunit (if any).

In CU mode, you can start a browse or an edit session by double-clicking SELECT on a CU icon. You can also click SELECT on the unit icon and then choose Open from the File menu.

“Understanding AdaVision Icons” on page 4-10 contains a full description of the unit icons.

When you open a unit, AdaVision opens a separate text editor window and loads the file containing the unit into the editor. The default editor is EditTool. You can change the default editor to be the text editor you prefer to use.

In each of the unit modes, AdaVision lets you set variables and properties associated with compiling and linking units. You view these units and manipulate and perform operations on them by using menu and command buttons, pop-up windows, property windows, and other OPEN LOOK user interface features and facilities.

Through its modes, AdaVision presents you with a dynamically changing picture of Ada libraries. AdaVision makes it easier to navigate through large, complicated libraries. Once you reach the desired location in a library, AdaVision provides constant visual feedback on what is happening. This feedback makes for an efficient as well as a more congenial setting in which to develop Ada applications.

- Chapter 2, “Using AdaVision,” describes AdaVision views and modes in detail.
- Chapter 3, “Managing Ada Libraries,” describes how to manage Ada libraries using AdaVision.
- Chapter 4, “Using AdaVision Unit Modes,” describes how to work in the two unit modes.

1.1.3 AdaDebug

AdaDebug icon

AdaDebug is an object-based and window-based interface of `a.db`, the SPARCompiler Ada debugger. 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. You can also enter `a.db` commands in a Custom text field.

1.1.3.1 Task Debugging

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 the following:

- A *Task Browser* that presents a global view of the program tasks
- A *Task Status* window that displays information about the status of a task

Chapter 5, “Using AdaDebug” describes how to use AdaDebug for debugging single tasking programs. Chapter 6, “Debugging Tasks with AdaDebug,” describes how to use AdaDebug when working with a multitasking program.

1.1.4 LRMTool

LRMTool icon

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

Chapter 7, “Using LRMTool,” describes LRMTool.

1.1.5 EditTool

EditTool Iicon

EditTool is a specialized version of the OpenWindows Text Editor. EditTool includes a window-based version of the Ada `a.error` facility. EditTool displays compiler-generated error messages in a display pane beneath the edit pane and scrolls the text to the location of the error. You can cycle easily through the source file to correct errors before recompiling.

Chapter 8, “Using EditTool,” describes how to use the SPARCworks/Ada-specific features of EditTool. This *Guide* does not describe Text Editor itself.

1.1.6 FileMerge

FileMerge icon

FileMerge is an interactive window-based tool for merging source files and coordinating source code changes with other developers.

You start FileMerge from either SPARCworks Manager or from a Command or Shell Tool.

From SPARCworks Manager, FileMerge can be launched in one of two ways:

- By dragging and dropping the FileMerge icon from the SPARCworks Manager palette onto the workspace
- By double-clicking SELECT on the FileMerge icon in the SPARCworks Manager palette

From a Command or Shell Tool, start FileMerge as follows:

♦ **Type `filemerge` at the system prompt and press Return.**

For more information on FileMerge, see *Merging Source Files.*

1.2 An Overview of 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, you start AdaVision, then choose New Library from the File menu in library mode. See "Creating a New Library" on page 3-10 for details.

If you have existing Ada source code, but no associated `ada.lib` file, you create an Ada library for the code in library mode. Then, you switch to LU mode and choose Import from the File menu to import the code into the new library.

When you choose Import, AdaVision uses the Ada `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, you link each main program unit. When the link job completes successfully, the program is ready to be run.

See "Importing Units" on page 4-17, "Linking Program Units" on page 4-31, and "Running a Program" on page 4-32 for a description of the steps.

At each step along the way, AdaVision offers you property windows (accessed from the Props menu) to set options for each operation—creating a library, importing LUs (for example, setting the level of optimization), and linking the main program.

1.3 General Features and Facilities

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

1.3.1 Objects and 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.

Figure 1-1 shows AdaVision in CU mode. Each icon in the display pane represents a SPARCworks/Ada unit object. SPARCworks/Ada tools are object-based tools. They represent Ada entities as objects and display them as icons.

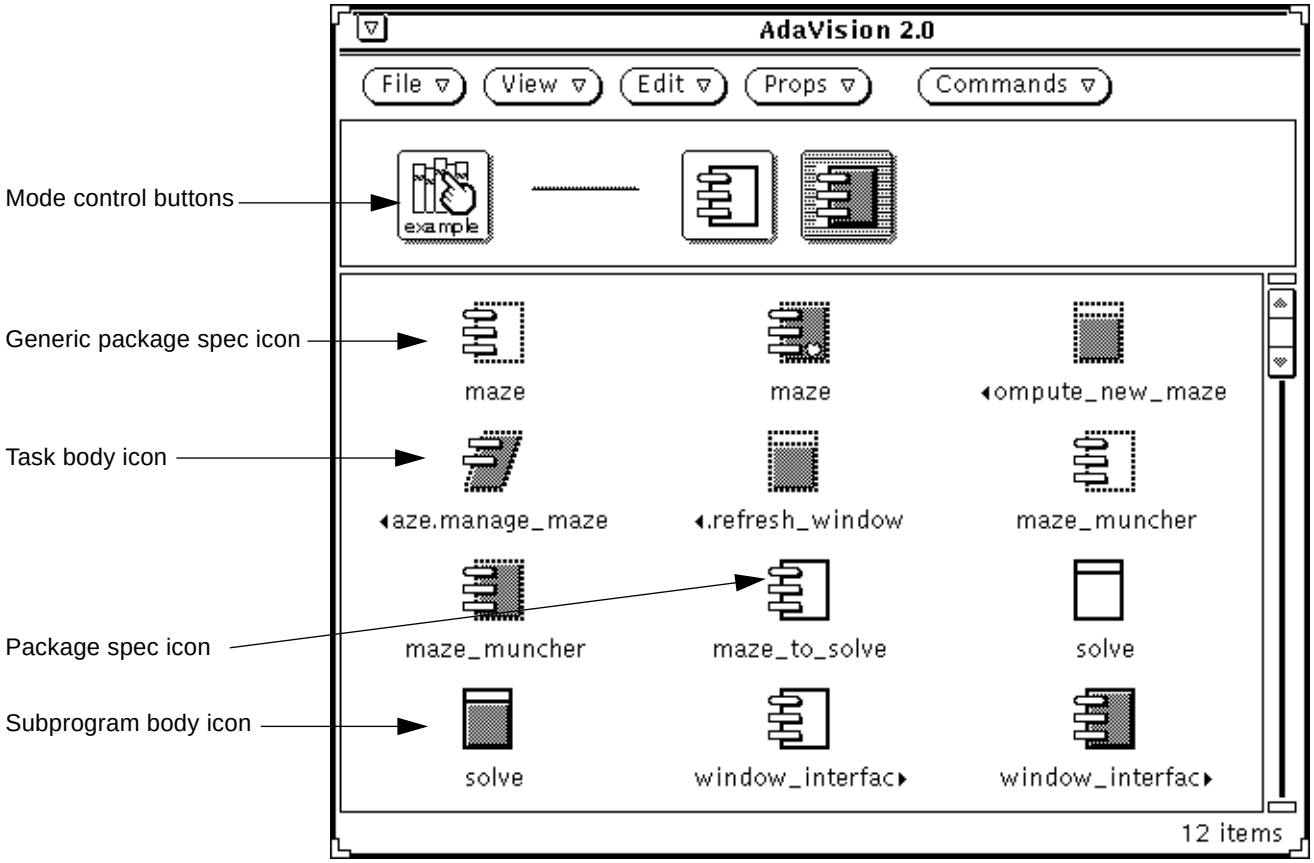


Figure 1-1 Compilation unit mode in AdaVision, showing unit objects as icons

1.3.1.1 Graphically-Coded Icons

Notice also in Figure 1-1 that the icons do not all look alike. 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 SPARCworks/Ada object. Here are a few examples, including one of the task status icons used in the Task Browser in AdaDebug:


Package spec icon


Package body icon


AdaDebug task status icon: task `in rendezvous`


Icon for an executable main program

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

AdaVision icons, because of their graphical coding, also help reduce the need for complicated naming conventions for Ada units.

1.3.1.2 SPARCworks/Ada Objects

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. SPARCworks/Ada objects are objects in three senses of the term.

First, as discussed previously, they are graphical objects.

Second, 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. SPARCworks/Ada tools "dim" or "gray out" inactive menu items or fields.

Third, SPARCworks/Ada objects can be made up of more than one type of Ada entity. A SPARCworks/Ada *LU object* is the main example. In Ada language terms, an LU is the specification unit, a "spec"—as distinct from a "body," which is also referred to as a secondary unit. The LU object is

composed of an LU spec, *plus* its body *and* any subunits. The LU object is constructed in this way to provide the developer with a view of the library unit structure of an Ada library.

1.3.2 Job Status Windows

Compile job running

Compile job completed successfully

Compile job terminated - unsuccessful compile

SPARCworks/Ada takes further advantage of object-based design and multitasking by creating *job status windows.* When you start an import, make, compile, link, tags, or delete (LU or CU mode only) job from a SPARCworks/Ada tool, SPARCworks/Ada starts a job that runs in the background in its own separate window.

When you start a job, its window is closed to an icon. Even here, SPARCworks/Ada makes use of graphics to inform you of the state of the job:

- A gray background indicates that the job is running.
- If the job is completed successfully, the background becomes clear.
- If the job has terminated unsuccessfully, the icon is "broken."

Opening the icon displays the job, the command line used, and any compiler or printer error messages. In the case of a compiler job, for instance, the window contains a button for EditTool which you can use to edit compilation errors. In addition, pressing a Redo button starts the compile job again after the problem has been fixed.

1.3.3 AutoRead

AutoRead is the name of an AdaVision feature that updates the visual representation of an Ada library at regular intervals. As units are added, deleted, recompiled, or brought into new interdependencies, AutoRead visually alerts you of these changes. You set the update interval, which may vary from seconds to minutes. AutoRead distributes information about the current state of a library to the user who is working with a particular library. These updates are especially helpful when more than one person is working with the same library over the same span of time.

"AutoRead Adjustment" on page 2-14 explains how to use the AutoRead feature.

1.3.4 Magnify Help for SPARCworks/Ada Tools

SPARCworks/Ada tools all have on-line help messages, known as Magnify Help. You access help messages by moving the pointer over the item you want to read about and pressing the Help key (the designated Help key on SPARCstation™ keyboards). The help messages identifying the many different kinds of icons are especially useful to new users.

1.3.5 OPEN LOOK GUI and OpenWindows

SPARCworks/Ada takes full advantage of the multiwindow and multitasking capabilities of workstations running OpenWindows. You can make full use of large displays to tailor your workspace. SPARCworks/Ada displays most operations in their own windows. Thus, you can edit in one window while compiling or debugging in another.

OpenWindows is the X11/NeWS window server implementation of the OPEN LOOK graphical user interface. The OPEN LOOK GUI gives the SPARCworks/Ada tools a consistent "look and feel."

This *Guide* does not document the generic OPEN LOOK GUI portions of the user interfaces for the SPARCworks/Ada tools. For information about OPEN LOOK, see *OPEN LOOK Graphical User Interface User's Guide.*

1.3.6 Objects in OPEN LOOK

The SPARCworks/Ada interface follows OPEN LOOK user interface conventions. Here is a quick sketch of what each of these basic operations means in terms of SPARCworks/Ada objects in AdaVision. You can perform three basic operations on a SPARCworks/Ada object.

First, you can:

- Select an object.

After you have selected an object, you can then:

- Open the selected object.
- Drag and drop the selected object.

1.3.6.1 Object Selection

You select an object by clicking SELECT on its icon.

Selecting an object *activates* it. Generally, activating an object by selecting it tells the tool to make that object the *target* of whatever operation you then perform.

Once selected, the range of functionality is determined by the type of object selected and its context. SPARCworks/Ada tools render many inappropriate or illegal menu items inactive by dimming their names.

You can select more than one object at a time by clicking ADJUST on other objects after selecting the first one. You can also click ADJUST to deselect any object after selecting it. You can select all objects in the display pane, including those out of view, through the Edit menu, which has a Select All item.

To deselect all objects, click SELECT in the pane away from any object.

1.3.6.2 Object Browsing

You open a selected object by double-clicking SELECT on its icon with the pointer (typically a mouse). Most often, you open objects to browse them for information about the underlying Ada unit or entity, or you open a unit object to edit the source code.

1.3.6.3 The Drag-and-Drop Feature

You drag and drop the selected object by positioning the pointer over the object and, while still holding the SELECT button down, dragging the icon to some other location on the screen and releasing the SELECT button.

The drag-and-drop feature in a GUI makes it easy to start a new operation. For example, you may find it convenient to keep AdaDebug running. You can then select an executable unit from the AdaVision display pane, “drag” its icon to the AdaDebug Program View window, and “drop” it onto that display pane. AdaDebug then loads the unit source code and is ready to start debugging the unit.

You can also drag and drop library directory icons from the OpenWindows File Manager onto the AdaVision library mode display pane. Doing so imports the Ada library to AdaVision.

1.3.7 Remote Display

Usually, you see your windows displayed on the same machine that runs the tool. As with any other OpenWindows application, however, you can run a SPARCworks/Ada tool on another SPARC system and direct the windows 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 OpenWindows client.

To set up remote display, do the following:

- Enter the `xhost` command at the system prompt on your machine—the display machine.
- Log onto a system on which SPARCworks/Ada has been installed.
- Set the environment variable `DISPLAY` to the display machine.

Use the following sequence of commands to accomplish the tasks listed. The words in *italic* are variables for the two machine names—*display_machine* is the name for the display machine; *swada_host*, the name for the SPARCworks/Ada host.

```
display_machine% xhost swada_host
display_machine% rlogin swada_host
swada_host% setenv DISPLAY display_machine:0
```

You may want to set up `DISPLAY` in your `.cshrc` file on *swada_host*. That way, you can simply log in and run SPARCworks/Ada on remote display.

The `xhost` command is effective until the display machine is rebooted; then you must reenter it.

For more information on using remote display, see the *X11/NeWS Server Guide*.

1.3.8 Remote Execution

By default, AdaVision and AdaDebug start Ada tools on the machine where they are executing. However, you can also configure AdaVision and AdaDebug to run Ada processes on a different SPARC system. Using this feature, you can run the GUI on one machine while submitting compute-intensive jobs or debug sessions to a designated server.

For simplicity, the machine running Ada processes is called *ada_host*; the machine that runs the SPARCworks/Ada tools, *swada_host*.

In remote execution, all Ada tools invoked by AdaVision and AdaDebug execute on *ada_host*, but send their results to the normal places in AdaVision and AdaDebug on *swada_host*. This protocol includes facilities that run on job status windows, such as Compile, as well as those that do not, for example, Cleanlib.

For details on how to activate remote execution, see "Remote Execution Use" on page 2-16.

1.3.9 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 Verdix Corporation, are a line of cross-development tools. Verdix has engineered parts of their VADScross product line to be compatible with SPARCworks/Ada. The use of SPARCworks/Ada and VADScross gives the developer 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. In fact, you can run self-target and cross-development projects at the same time from AdaVision.

Note – For a list of compatible VADScross products, contact Verdix Corporation.

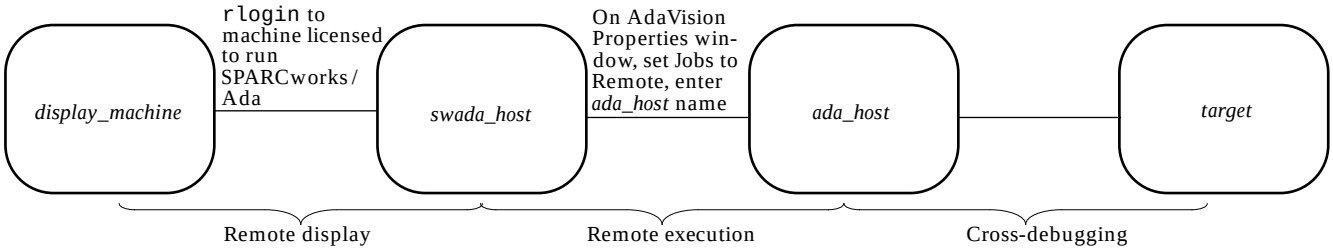
After you have installed a SPARCworks/Ada-compatible VADScross product in your SPARC system, your `/etc/VADS` file includes the cross-development release(s) now available to you. You can then create a new library using the parent release of your choice. For more information on this subject, refer to "Creating a New Library" on page 3-10.

Note – How to check and set up Ada Info and Link library directives for both self-target and cross-development is described in "Setting Info and Link Directives" on page 3-15.

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

The following diagram illustrates the relationships between the various machines:



This example illustrates the versatility of these features. Such a complex configuration is rarely necessary, however, because *ada_host*, *swada_host*, and *display_machine* are typically the same machine.

Using AdaVision 2

AdaVision serves both as an Ada library 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, LRMTool, and EditTool. By manipulating SPARCworks/Ada objects from within AdaVision, you can carry out virtually all of the tasks associated with developing an Ada application.

This chapter shows you how to start AdaVision, how to manipulate its three viewing modes, and how to access the other SPARCworks/Ada tools from within AdaVision. Chapter 3, "Managing Ada Libraries," describes how to manage libraries. Chapter 4, "Using AdaVision Unit Modes," describes working with Ada units.

This chapter is organized into the following sections:

2.1 Starting AdaVision

You can start AdaVision from anywhere in your file system. However, the best place to start is in the Ada library *directory* containing the library you want to work with.

Note – Your system should be completely set up. See "Hardware and Software Requirements" on page 1-xix in the Preface.

2.1.1 Remote Execution Startup

If the main program in your library is an OpenWindows application, and you would like to use remote execution to run it, you must first set your environment for remote display of your main program windows.

Use the following commands at the system prompt of a Command or Shell Tool before starting AdaVision.

swada_host% **xhost** *ada_host*
swada_host% **setenv DISPLAY** *display_machine:***0**

The variables, *ada_host*, *swada_host*, and *display_machine*, are the machine names. Note that *swada_host* and *display_machine* might be the same machine.

After you start AdaVision, remember to set the AdaVision Jobs control setting to Remote and enter the *ada_host* on the AdaVision Properties window. "Remote Execution Use" on page 1-16 shows you how to do so.

2.1.2 AdaVision Startup

AdaVision can be started from SPARCworks Manager. See "SPARCworks Manager" on page 1-2 for details.

To start AdaVision from the command line:

- **Type** `adavision &` **at the system prompt in a Command or Shell Tool window, and press Return.**

Note – The ampersand sign (&) after the command is optional. Including it in the command line causes the program to run in the background and leaves the Command or Shell Tool window free for execution of other commands.

AdaVision then displays its main window, which looks like the one shown in Figure 2-1. The contents of the display pane—Ada library objects—vary, depending on where you start AdaVision in your file system.

You can also add AdaVision to your OpenWindows Workspace Programs menu. See the OpenWindows documentation for instructions.

2.1.3 AdaVision at Startup

The AdaVision interface is organized around three main windows and many pop-up windows. Figure 2-1 shows the top-level, library-mode main window. The title bar states the name and version number of the software. The control panel contains five button menus.

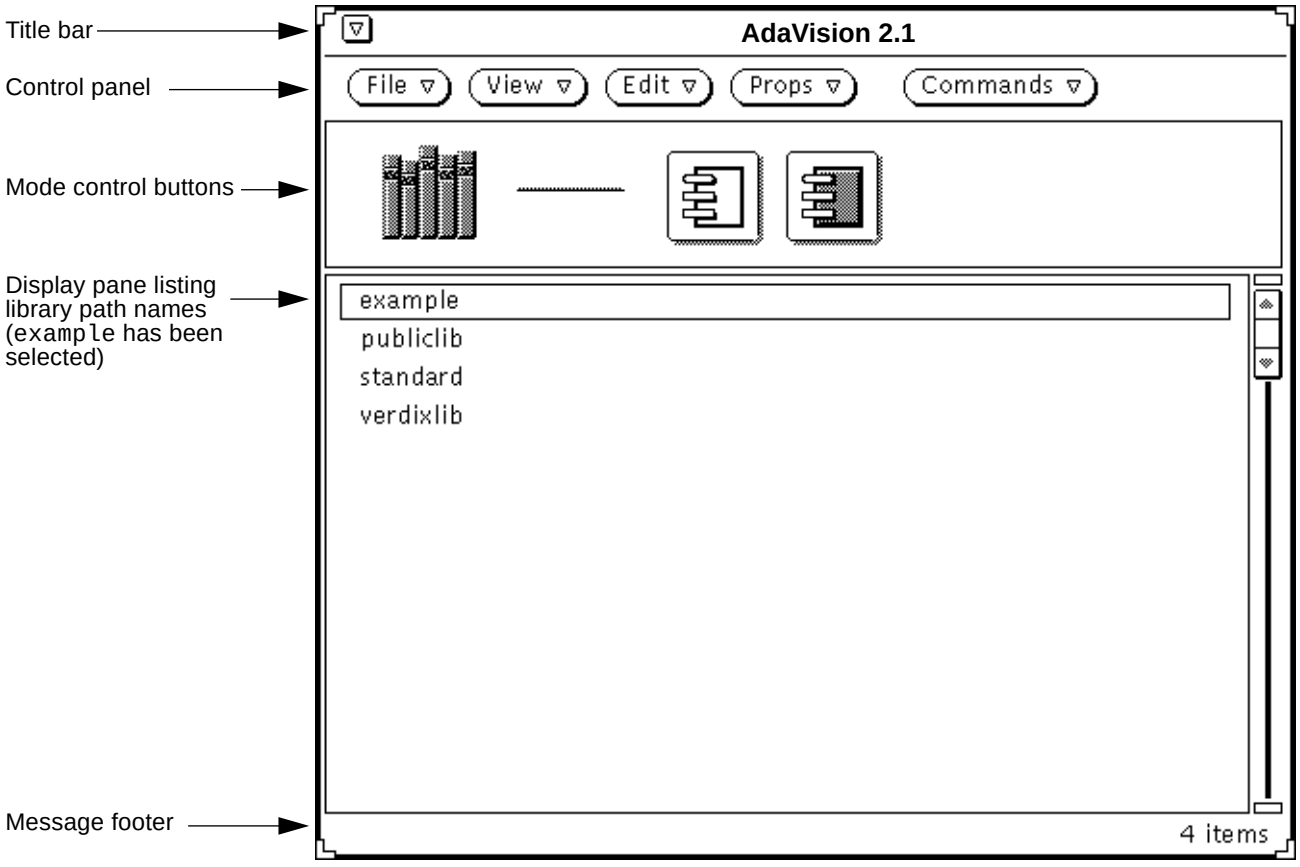


Figure 2-1 AdaVision in library mode, with the `example` library selected

To see what items are on the menu buttons, open the window on your screen and take a few seconds to click MENU on each of them.

The three icons beneath the control buttons are mode control buttons. You click SELECT on them to control which mode AdaVision is in.

AdaVision displays library path names in the display pane beneath the mode selection area.

In Figure 2-1, the `example` library is selected—but not yet opened—as indicated by the rectangle drawn around the library name.

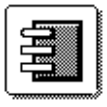
Beneath the display pane is the message footer area. AdaVision uses the left footer to display information about the objects and the operations you perform, including error messages. The right footer always tells you how many objects are being displayed. The display pane has a scrollbar for scrolling through large lists of libraries or icons.

2.2 Understanding AdaVision Modes

AdaVision's views of Ada libraries and their units are organized into three levels: the top-level *library mode*, the middle-level *library unit (LU) mode*, and the lower-level *compilation unit (CU) mode.*

Note – The structure of AdaVision is shown in this chapter. Functionality within each of the three modes is described in the following two chapters.

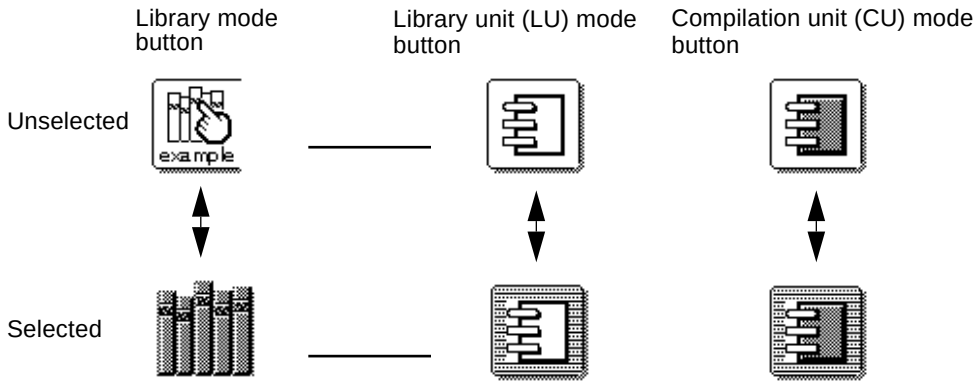
2.2.1 Mode Selection Buttons



Each AdaVision control button represents one of the three viewing modes. You switch among modes by clicking SELECT on one of the iconic buttons displayed permanently in the mode selection control pane. The button representing the current mode is always highlighted.

AdaVision always starts in library mode, the top level. The library mode button that depicts a set of library books is highlighted. That highlighted button prompts you to select one of the libraries shown in the display pane. The two unit mode buttons are inactive until you select a library. Once you select a library and press a unit mode button, the library mode button shows the name of the selected library.

LU mode is represented by an icon that depicts a package specification; CU mode is represented as a package body.



2.2.2 *Library Mode*

In library mode, you examine and manage Ada libraries.

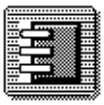
The items on the File, View, and Edit menus are mainly tools for controlling the list of libraries shown in the AdaVision display pane and for sorting the list. You can import libraries to the list and create new Ada libraries. When you create a new library, AdaVision adds it to the list. You can edit the library ADAPATH from the selected library properties window.

Library mode is where you set Info and Link directives for the selected library. Chapter 3, “Managing Ada Libraries,” describes the processes.

Library mode contains the menu item controls for accessing the other SPARCworks/Ada tools. (See "Accessing Other SPARCworks/Ada Tools" on page 1-11).

Library mode houses the controls for setting several AdaVision defaults, for example, which printer and which editor you want AdaVision to use when you open a unit, and so on. (See "Customizing the AdaVision Environment" on page 1-11.)

2.2.3 Compilation Unit (CU) Mode



If you go directly from library to CU mode, that is, if you click SELECT on the CU mode button w*ithout first selecting an LU object*, AdaVision displays an object for every CU in the selected library: specs, bodies, and subunits. For smaller libraries, this display is fine, but for large ones, the sheer number of units and subunits can make navigation difficult.

AdaVision addresses this problem through *filtering*. In CU mode, you can filter the display of the objects by *type* of unit: packages, subprograms, tasks, generics; or by *kind* of unit: specs, bodies, or subunits.

2.2.4 Library Unit (LU) Mode

While filtering is useful for many purposes, it does not tell you much about the structure of the library you are browsing. AdaVision's intermediate-level mode—LU mode—lets you focus on the library units associated with a particular library. If you switch to this view after selecting an Ada library, AdaVision displays an LU object for each Ada library unit (spec) in the selected library.

An LU object is a composite object; its single icon is a representation of that unit's spec, body, and any associated subunits. If you select one of these LU objects and *then* switch to CU mode, AdaVision displays only the CU objects associated with that one LU.

The next three figures show the differences in the views. The `example` library used in the examples is small. For larger Ada libraries, this focusing effect can be dramatic.

Figure 2-2 is a view of the `example` library in CU mode with *no* selected LU object. AdaVision displays all 12 of the library CUs.

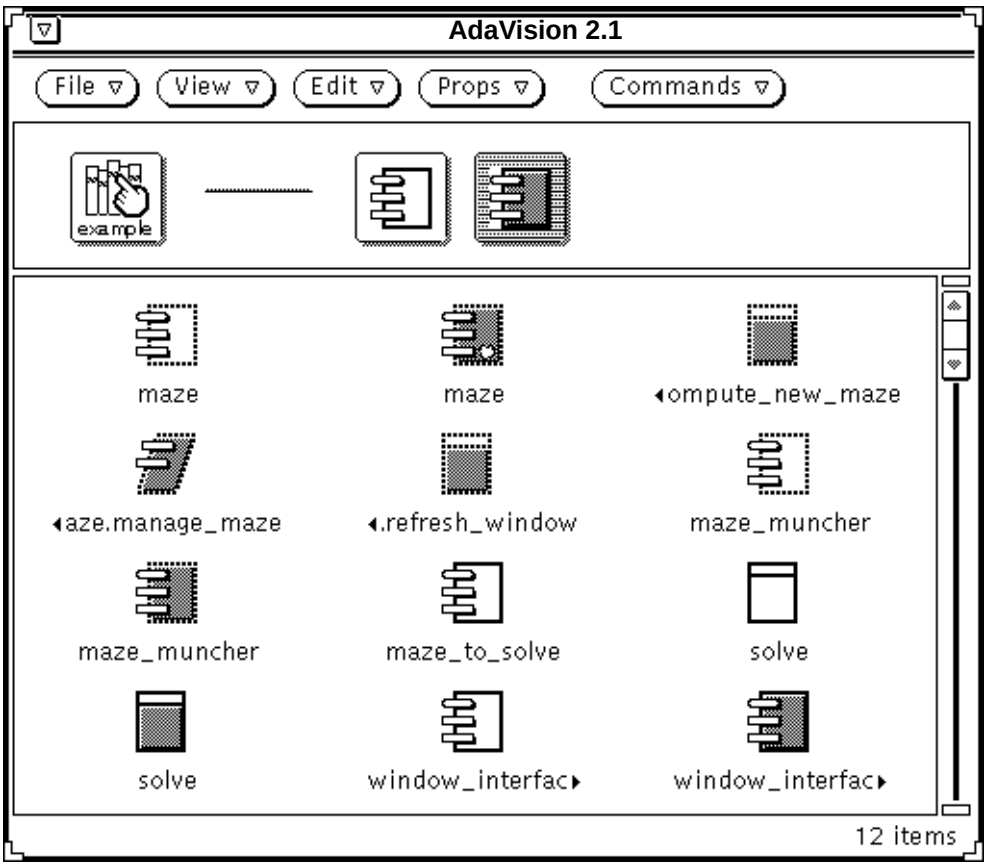


Figure 2-2 CU mode displayed from library mode

The next figure, Figure 2-3, is a view of the library in LU mode, where the LU objects are displayed with `maze_muncher` selected, followed by the view in CU mode again, this time showing only the CUs associated with maze_muncher.

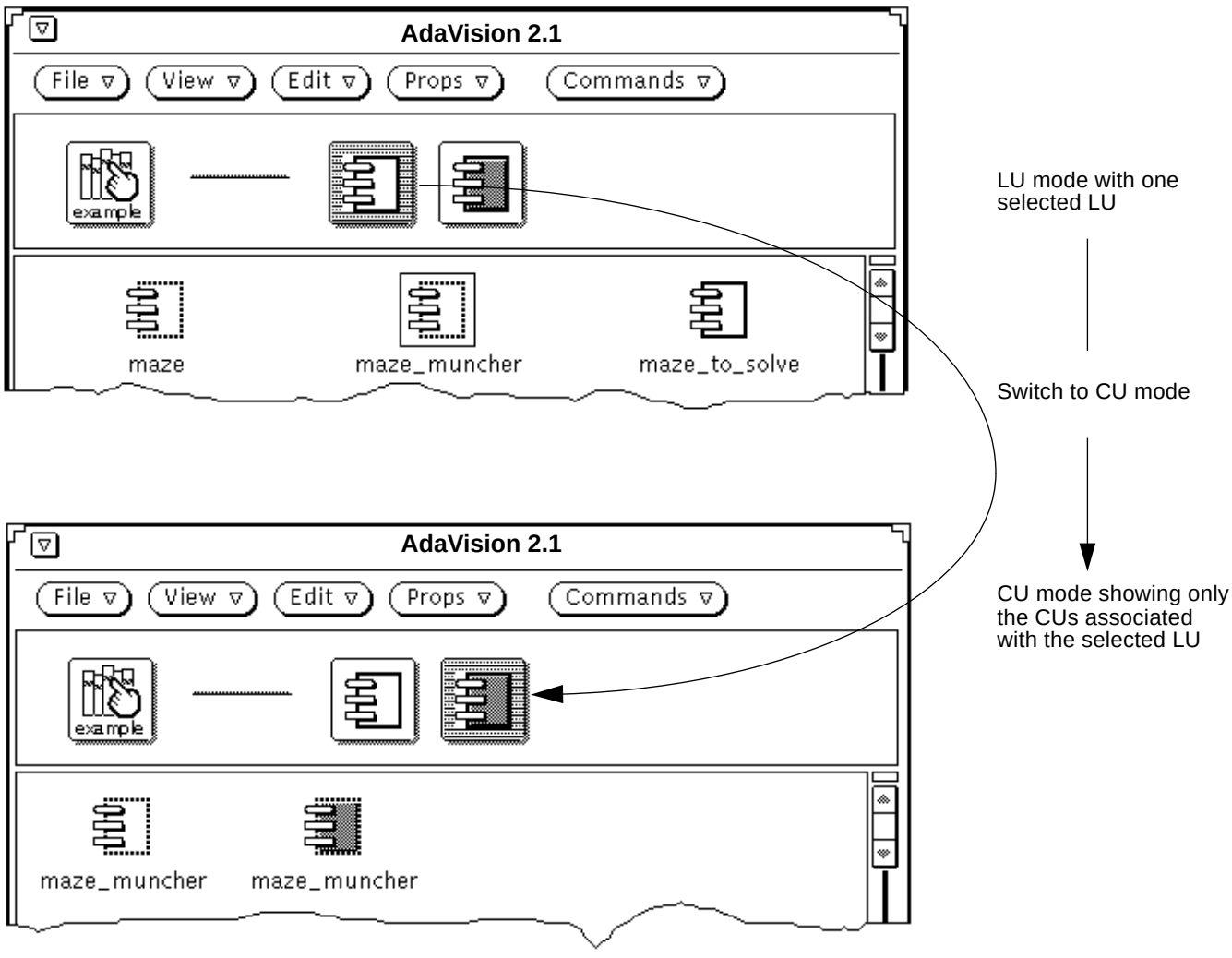


Figure 2-3 CU mode displayed from LU mode

2.2.5 Ada Units in the Unit Modes

Developers do most of their work in the two unit modes. You start an edit session by double-clicking SELECT on a unit icon. You can also select the icon and then choose Open Lib Unit from the File menu in LU mode, or Open from the File menu in CU mode. The File menu also has items for importing units into the selected library or creating new ones.

If you click SELECT on the Commands menu in either unit mode, you see a list of the items associated with the type of objects you are viewing.

Notice that the lists of items in the Commands menus in LU mode and CU mode are slightly different from one another. The differences reflect the functionality associated with working with units at each of the two levels.

For example:

- You link modules and units or run a program by first selecting the executable main program, which is an LU object. Link and Run are on the LU mode Commands menu, but not on the CU mode Commands menu.
- Both lists contain items for starting EditTool and LRMTool.

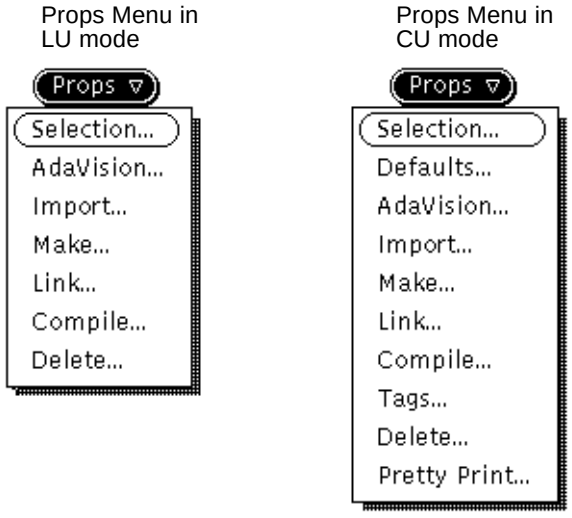
Commands menu in LU mode

Commands menu in CU mode

One major difference between LU mode and CU mode is that in LU mode, when you select an LU object, then highlight Compile on the Commands menu, a submenu is attached to it. You then choose Spec, Body (if any), or Subunit (if any) from the submenu. You have to tell AdaVision which unit *within the LU object* you want to target. Spec is the default choice.

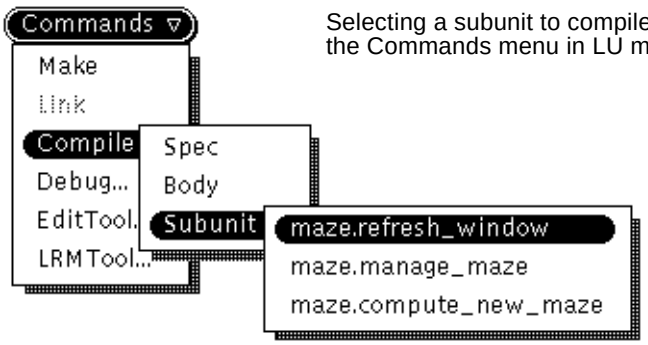
The Props menus for LU and CU modes contain slightly different items. For example, the property window for the selected LU or CU, as shown in the figure above, This applies for AdaVision in general, as well as the windows for the jobs that can be performed in each of these modes. For the jobs, the windows contain controls where you can set options and check other information associated with these jobs.

The following figure shows the two different menus.



Chapter 4, “Using AdaVision Unit Modes,” describes in detail how to work in the two unit modes.

See the following graphic for an example of how this selection is made.



Selecting a subunit to compile from the Commands menu in LU mode

2.3 Accessing Other SPARCworks/Ada Tools

AdaVision serves as an access point for invoking the other SPARCworks/Ada tools. The Commands menu in both LU and CU mode contains items for invoking LRMTool and EditTool. You can start AdaDebug in LU mode, but not in CU mode. Either choose Debug after you have selected an executable LU or choose Debugger with no executable.

When you choose one of these items, AdaVision starts the chosen tool in a separate window. A button for starting EditTool is also present in various job status windows. This *Guide* describes each tool in its own chapter.

2.4 Customizing the AdaVision Environment

You can customize several aspects of the AdaVision environment. A few of the things you can change or adjust are general properties of AdaVision that you set once or infrequently. The controls for these settings are in the AdaVision Properties window. Once you have changed and applied a setting, it is saved for your next startup of AdaVision. "Save Settings Feature for View Options" on page 1-25 describes this feature in more detail.

Features you may want to change more frequently are spread across several menus. They are mainly mode-specific controls for adjusting the display of objects. You can display unit objects in a list with glyphs instead of as icons or, in the case of library objects, list them with full path names instead of simple path names.

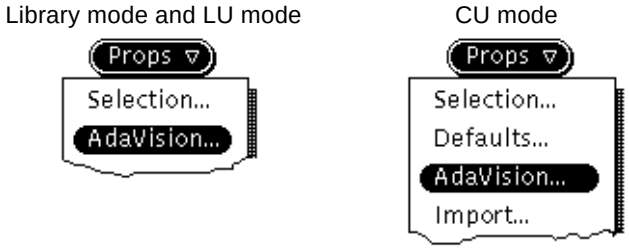
In AdaVision, you can also sort, filter, and key-search through large displays or lists of unit objects. Chapter 4, “Using AdaVision Unit Modes,” describes these features in detail.

2.4.1 Default Editor Selection

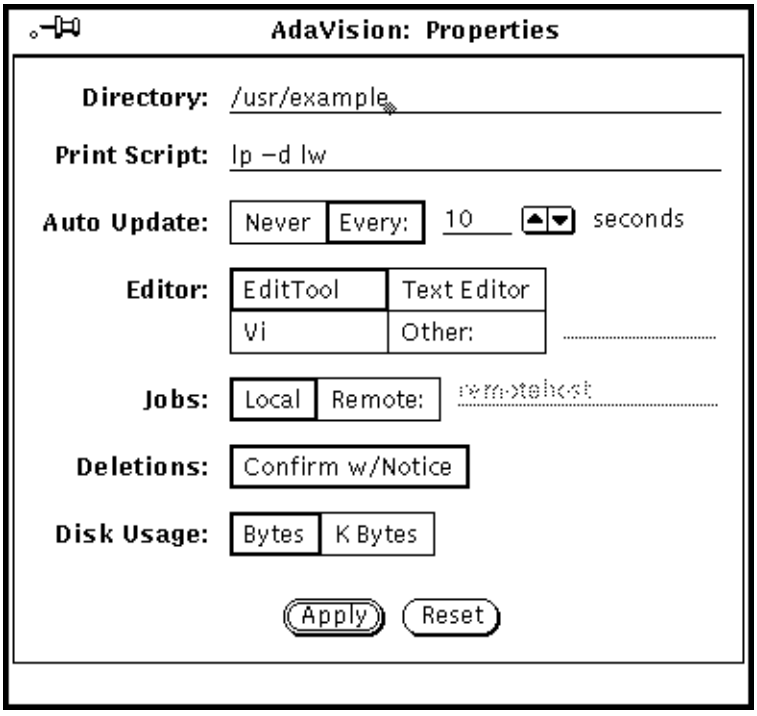
When you open units, AdaVision loads source files into EditTool, the default editor for AdaVision. You can change the default to vi or to some other text editor installed on your system.

To set a default text editor for AdaVision:

1. Choose AdaVision from the Props menu (any mode).



The AdaVision Properties window is displayed.



2. **Click SELECT in the box with the editor of your choice in the Editor control and press the Apply button to apply the setting.**
   Notice that in addition to the three choices offered, you can set the value of the Editor control to Other and then type in the name of some other editor installed on your system.

2.4.2 Printer Name Assignment

The Print Script control is an editable text field which shows the printer you use with AdaVision. Depending on the `PRINTER` environment variable, the setting is, in most cases, the default `lw` immediately following the print command `lp -d`.

To assign a different printer:

1. **Choose AdaVision from the Props menu in any mode to open the AdaVision Properties window.**
2. **Delete `lw` and type in the name of the printer you want to use.**

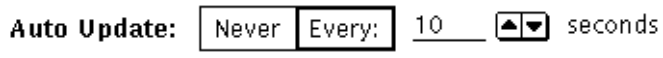
2.4.3 AutoRead Adjustment

AutoRead is the SPARCworks/Ada facility that updates AdaVision's graphic representations of Ada libraries to reflect changes in the library. The updates take place at regular intervals. The default interval is 10 seconds. It is recommended that you work with the default settings for awhile to see how AutoRead works.

You can change the interval between updates and turn it on or off by setting the Auto Update control in the AdaVision Properties window.

To turn AutoRead on or off or to adjust the update interval:

1. **Choose AdaVision from the library mode Props menu to open the AdaVision Properties window.**
2. **Set Auto Update to Every to turn AutoRead on; edit the default of 10 to reflect the number of seconds between updates or use the Up or Down scrolling button to specify the number of seconds.**



3. **Press Apply.**

To disable AutoRead, set Auto Update to Never.

2.4.3.1 Using AutoRead

Because AdaVision relies so heavily on graphic representations of libraries and their units, it is useful to have a way of updating automatically the representations as the library changes.

Every time you select a library in library mode and then open it, AdaVision must check to see what units are in that library so that it knows what unit objects to display. It also must check to see what properties and options are associated with each unit.

AutoRead consists mainly of a dynamic connection between the AdaVision windows and the library. When you enable AutoRead, AdaVision rechecks the library at the specified interval and updates its windows accordingly. It is as if you returned to library mode and asked AdaVision to display the same library again, knowing that it may have changed in the interval.

Consider what happens when you compile a new library containing many units. With AutoRead on and the Timer set to a few seconds, AdaVision can update the CU mode display pane with new objects almost as fast as they are compiled into the library. It works as a visual gauge of the compilation.

AutoRead is also of value when more than one person is working with the same library. If the other user compiles a new unit into the library, you will see its icon being displayed following the next AutoRead. Without AutoRead, you would not see the new unit until you reopen the library.

AutoRead provides feedback in dependency windows. As new units added to a library create new dependencies, the graph is dynamically updated. ‘‘Dependencies Graph of Units in an LU,” on page 4-14 describes this graph.

2.4.3.2 Working with AutoRead Off

If you turn AutoRead off, you view a static snapshot of the library as it was at the time you first selected and displayed it. The contents of the windows do not change unless you reopen the library again from library mode or unless you add or delete specific objects to it. If there is no critical information about a fast changing library that you want AdaVision to track closely, then working without AutoRead is fine. You might want to set the Timer to update the library every 3 or 5 minutes. This way, the library is reasonably up-to-date.

When you positively do not want AutoRead to visually echo every change to a library, you can disable it from the AdaVision Properties window, as explained in the previous section.

2.4.4 Remote Execution Use

Before reading how to use remote execution, see "Remote Execution" on page 1-14 for a description of this feature.

For proper use of remote execution, some initial preparation is required. AdaVision implements this feature 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, that is, your *ada_host*. 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 2.0 Developer Documentation* (825-1453-02).

In addition, Ada libraries and source files must be NFS-mounted on exactly the same path name for both the local machine and the remote machine. Ada libraries should physically reside on the compile server for efficiency.

Note – Make sure that you set up the same environment for both your *swada_host* and *ada_host*.

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*.

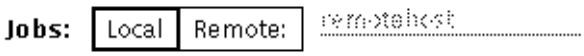
To enable remote jobs from AdaVision:

1. **Choose AdaVision from the Props menu to open the AdaVision Properties window.**

Library mode and LU mode

CU mode

2. The Jobs control is set to Local, as shown below.



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 will still keep the same setting they had when they were started.

2.4.5 Object Display

Library objects are listed by name. You can choose to list libraries with full path names or simple names. Full or simple path names are controlled from the library mode View menu.

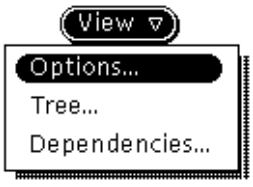
2.4.5.1 Displaying LU Objects

By default, AdaVision displays LU objects and CU objects in *icon display mode*, that is, named icons arranged in rows and columns. This mode is the default.

Alternatively, you can display LU objects in *list* display mode, that is, tight lists of unit names with a smaller version of each unit icon, known as a glyph, to the left of the name. In choosing this mode, you can have AdaVision display the type and/or size of the units, sorted by name or type.

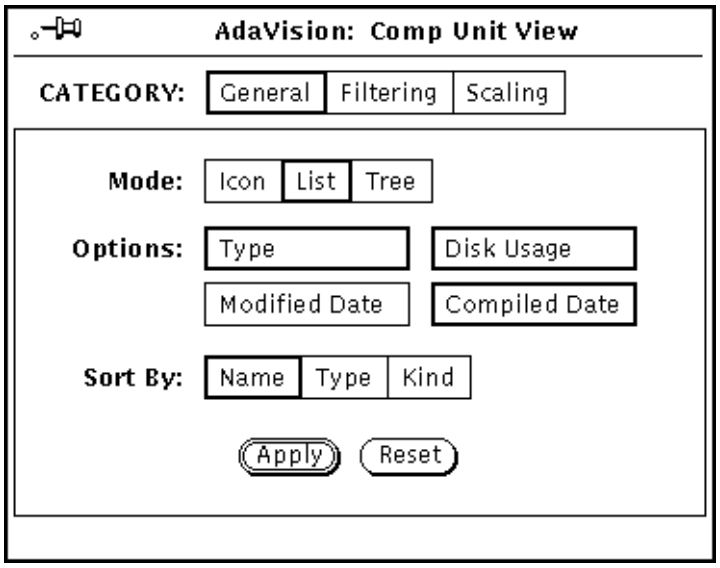
LU types include: generic subprograms, subprograms, generic packages, and packages.

To choose list display in LU mode:



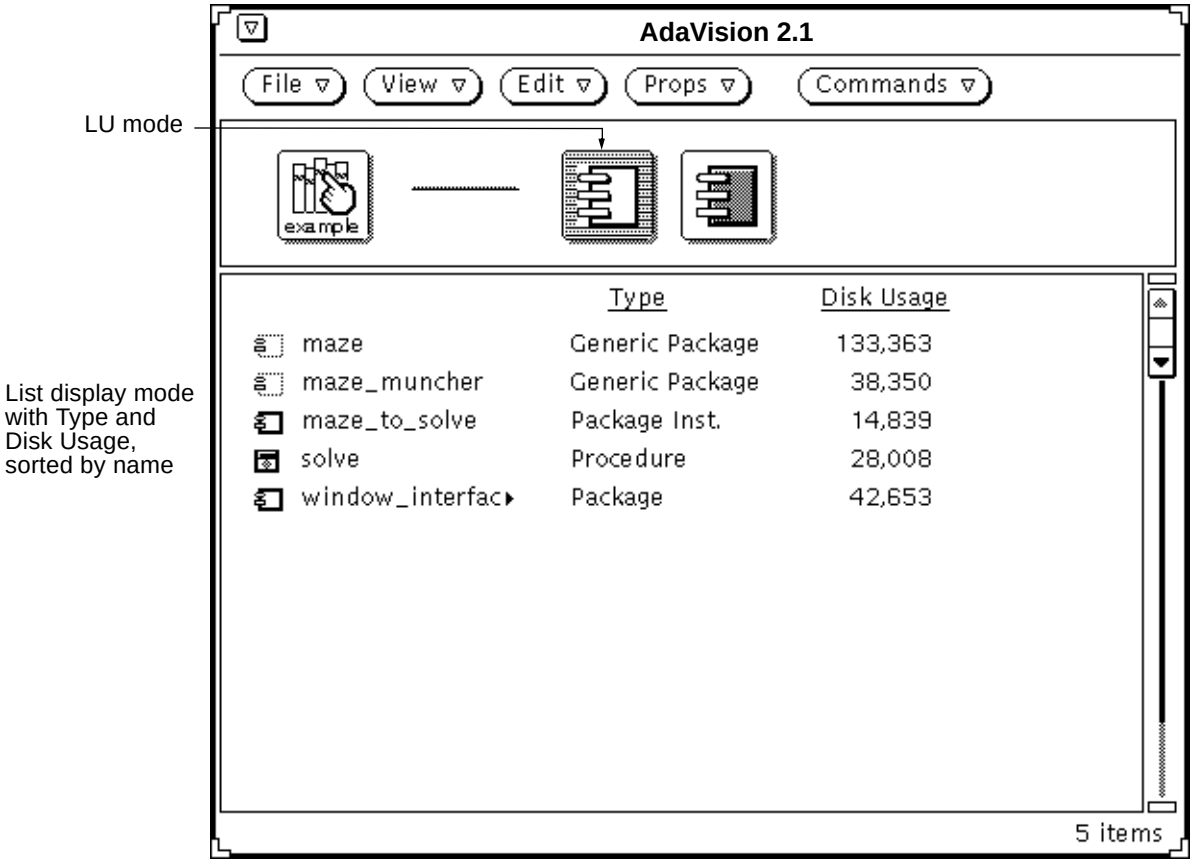
1. Switch to LU mode.

2. Choose Options from the View menu.
The Lib Unit View window is then displayed.



3. **Click SELECT to set the CATEGORY control to General.**
   The Options window opens to the General subwindow by default, but if either the Filtering or the Scaling controls are showing, you must reset CATEGORY to General.
4. **Click SELECT on List under Mode.**
5. **At the Options control, click SELECT on Type and/or Disk Usage to have that information displayed. You can also deselect both.**
6. **At the Sort By control, you can specify whether to sort the list by Name or Type. Click SELECT on your choice.**
7. **Press Apply.**

The following figure shows how the LUs for the library, `example`, are displayed in list mode, with Type and Disk Usage chosen, sorted by name.



2.4.5.2 Displaying CU Objects

You can display CU objects in three different ways: *icon*, *list*, or *tree* display mode.

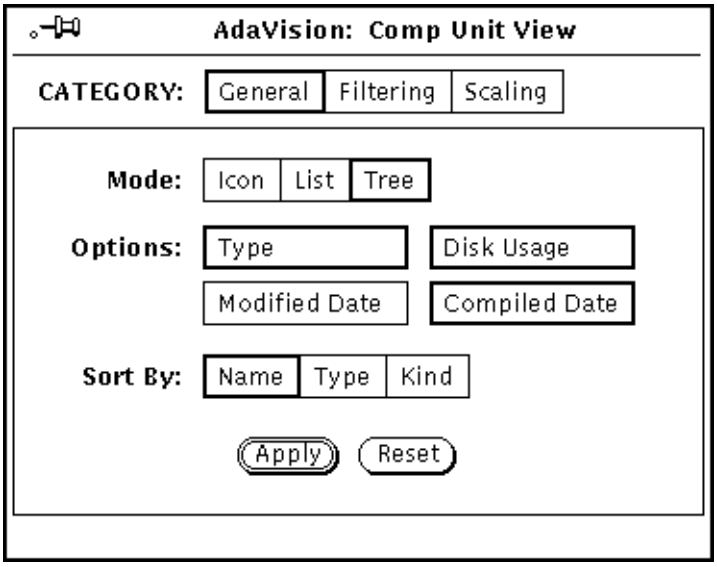
In choosing list mode, you can have AdaVision display all or a combination of the following kinds of information for the CU objects:

- Type
- Disk usage
- Modified date
- Compiled date

You can also sort the objects by name, type, or kind. CU object *types* include: tasks, generic subprograms, subprograms, generic packages, and packages. CU object *kinds* consist of specs, bodies, and subunits.

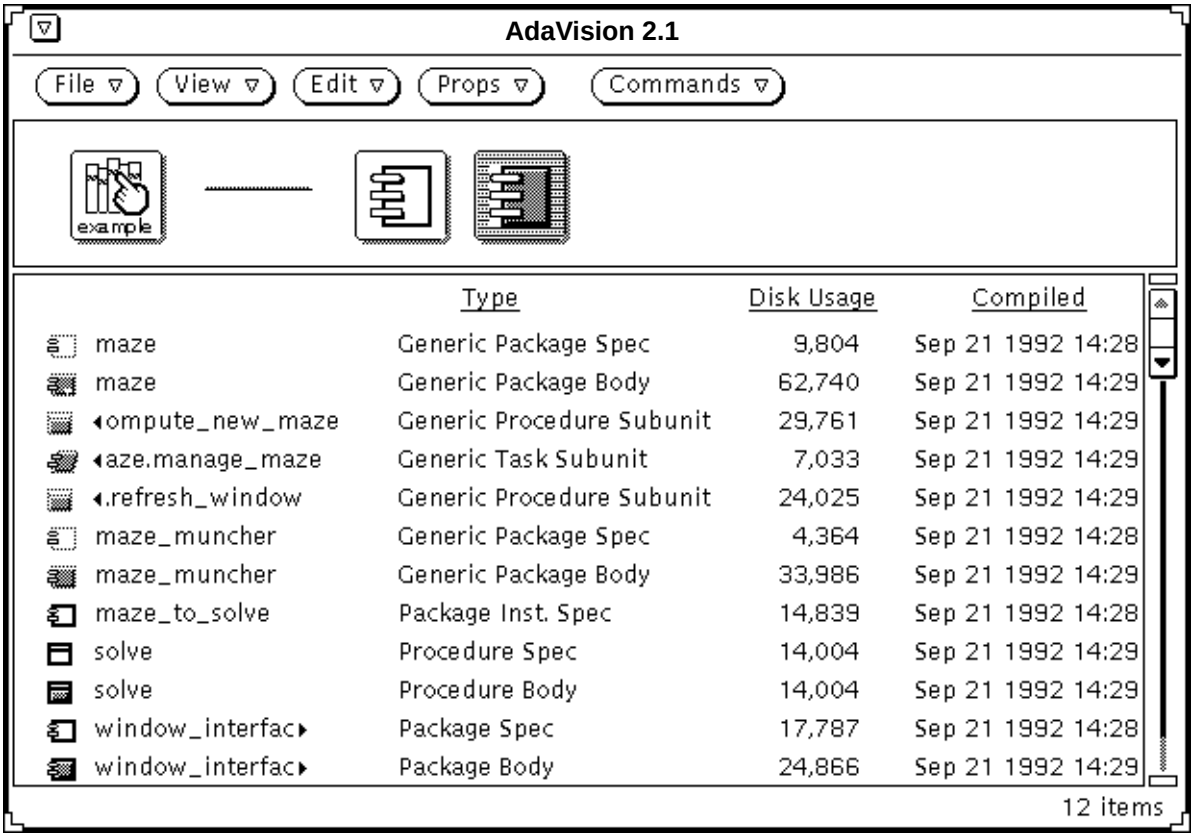
To display CU objects in list mode:

1. **Switch to CU mode.**
2. **Choose Options from the View menu.**
   The Comp Unit View window is then displayed.



3. **As in LU mode, make sure CATEGORY is set to General.**
4. **Click SELECT on List at the Mode control.**
5. **For Options, you can choose to have any or all of the following information displayed: Type, Disk Usage, Modified Date, and Compiled Date. Click SELECT on one or more of these options.**
6. **At the Sort By control, you can specify whether to sort the list by Name, Type, or Kind. Click SELECT on your choice.**
7. **Press Apply.**

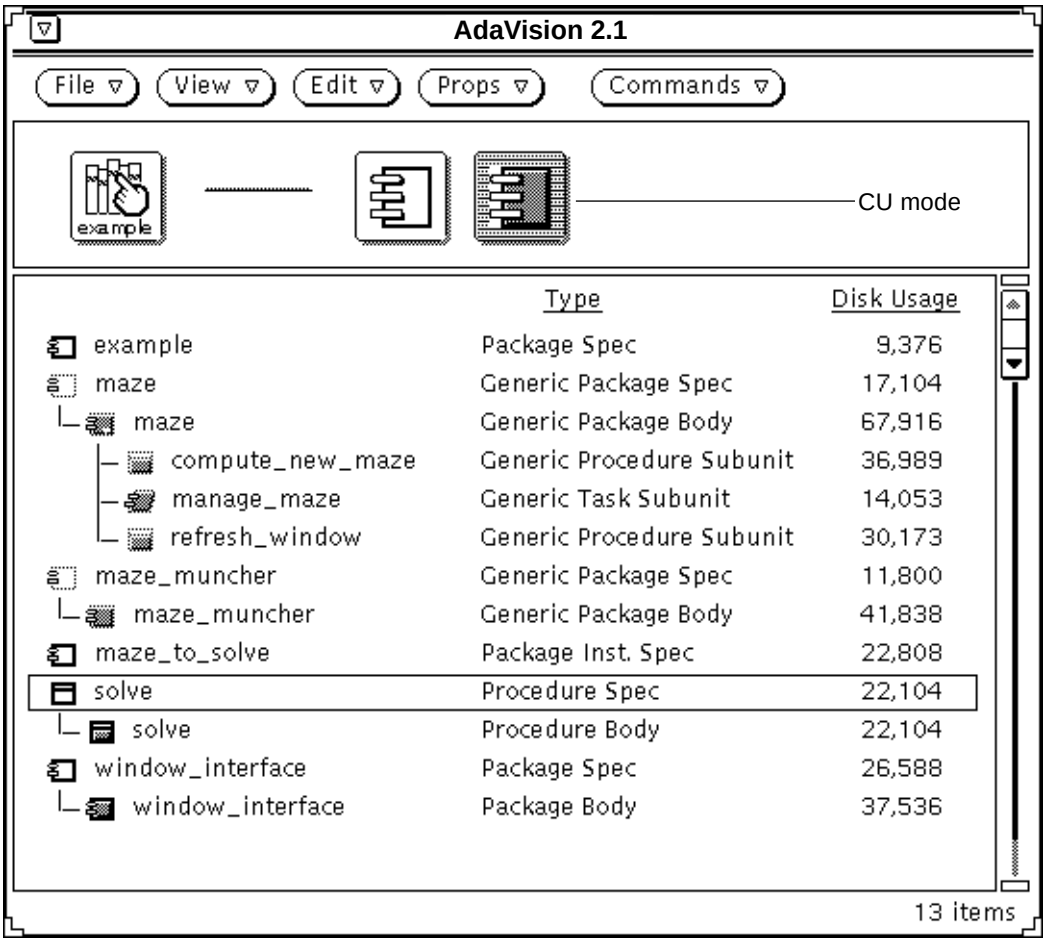
The following figure shows how the CUs for the library, `example`, are displayed in list mode, showing the Type, Disk Usage, and Compiled Date for each of the units. They are sorted by name.



The *tree* display mode arranges a list of icons with lines to show graphically the relationships among the units that constitute an LU object.

To change to tree display mode, set the Options control to Tree at the Comp Unit View window, then press Apply.

The following figure shows the CUs for the `example` program in tree display mode.



Note – For a description of the tree display mode in LU mode, see "Tree Graph of Units in an LU" on page 1-13.

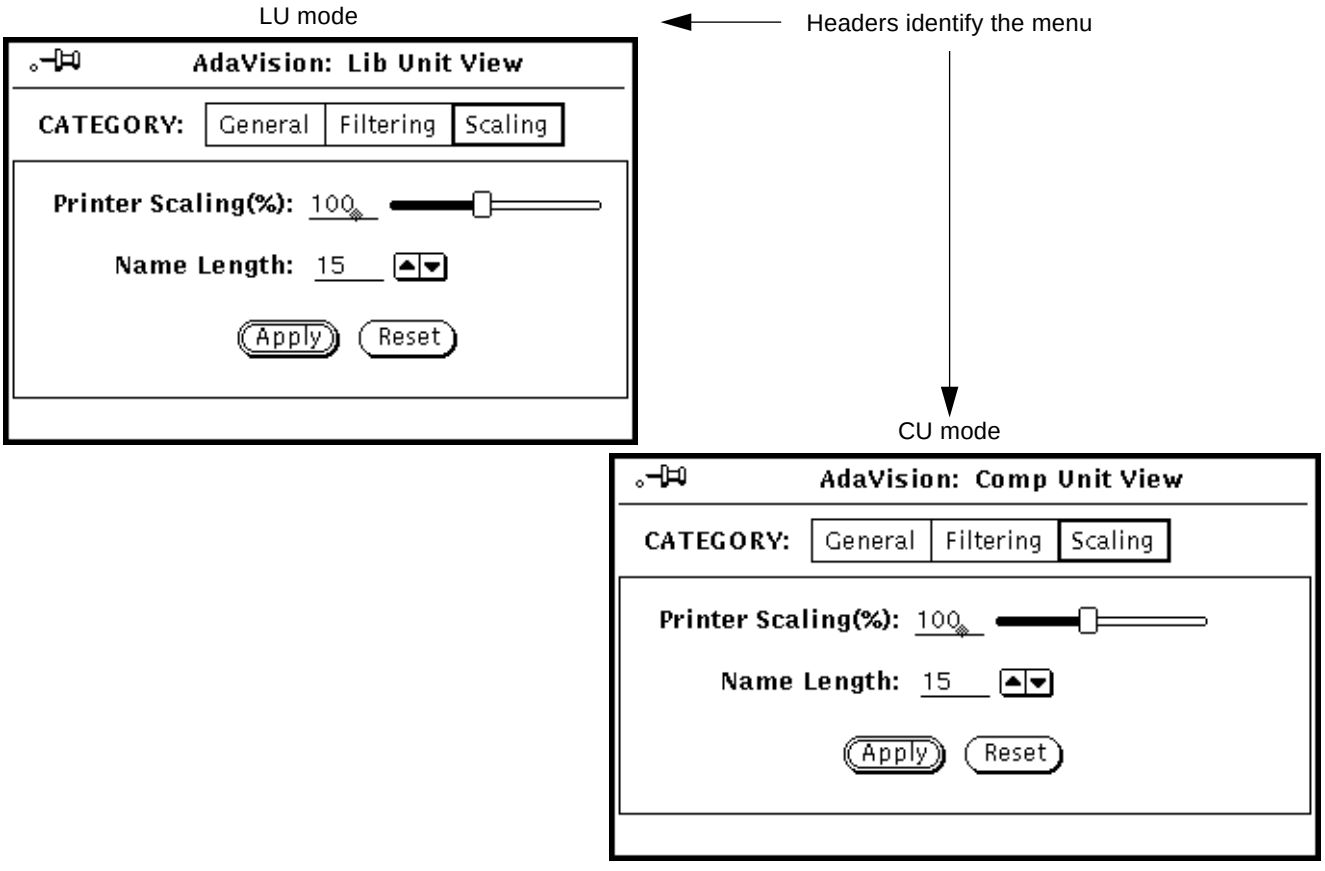
2.4.6 Object Name Length Adjustment

In icon display mode, you can set the number of characters in the object name that will print in the display pane before clipping the name.

Note that this name length adjustment applies to the object names displayed beneath the icons in icon display mode and the name of objects on list display mode only, not the name of objects used in tree display mode.

To adjust how many characters of an object name are displayed:

1. **Choose Options from the View menu in either LU or CU mode.**
   Doing so opens the Unit View pop-up window. When you first open this window, it displays the General controls.



2. **Click SELECT on Scaling under CATEGORY.**

3. **Type the number of characters in the Name Length control.**
   Alternatively, click SELECT on the Up scrolling button to raise the number, or the Down scrolling button to lower it.

4. **Press the Apply button.**

2.4.7 Printer Scaling

"Display Pane Content Printing" on page 1-45 describes how to print AdaVision screens. When you print the screens, you can scale the icons for printing between 40% to 180%.

To adjust printer scaling:

1. **Choose Options from the View menu in either LU or CU mode.**
   Doing so opens the Unit View pop-up window as shown in the previous section.

2. **Move the slider for Printer Scaling to the desired percentage of scaling.**
   The default is 100. The text field reflects the new percentage as you move the slider left or right.

Printer scaling can also be effected for LU tree graphs from the Tree window in LU mode. Refer to "Tree Graph of Units in an LU" on page 1-13 for a description of that window.

2.4.8 Unit Locations in Large Libraries

Ada libraries are often quite large, with scores or even hundreds of units in a single library. In addition to plain scrolling, AdaVision has four major features to help you navigate through many units:

- *Sorting* units by name, type, or kind
- *Filtering* units by type or kind
- *Key-searching* by alphabetical order within a normal, sorted, or filtered display
- *Focusing* on compilation units associated with a library unit selected in LU mode

Chapter 4, “Using AdaVision Unit Modes,” describes these features.

You have already seen how you can use LU mode to reduce the number of CUs that AdaVision displays: you select an LU object and then switch to CU mode. Instead of seeing all the CUs in the library, you see only the ones associated with the selected LU.

2.4.9 Save Settings Feature for View Options

In all AdaVision modes, the settings in the pop-up windows that are invoked from the View menu are saved once they have been changed. Thus, when you restart AdaVision, the display mode along with its options remains the same as that for the previous session.

Managing Ada Libraries 3

This chapter describes how to use the AdaVision top-level library mode to browse and manage Ada libraries. AdaVision provides tools for understanding the structure of a selected library and tools for working more efficiently with library units.

Library management using AdaVision falls into two main parts:

- Controlling which libraries AdaVision lists on the display pane
- Setting or checking selected library Info, Link, or WITH directives, and its ADAPATH

This chapter is organized into the following sections:

3

3.1 Viewing Library Properties

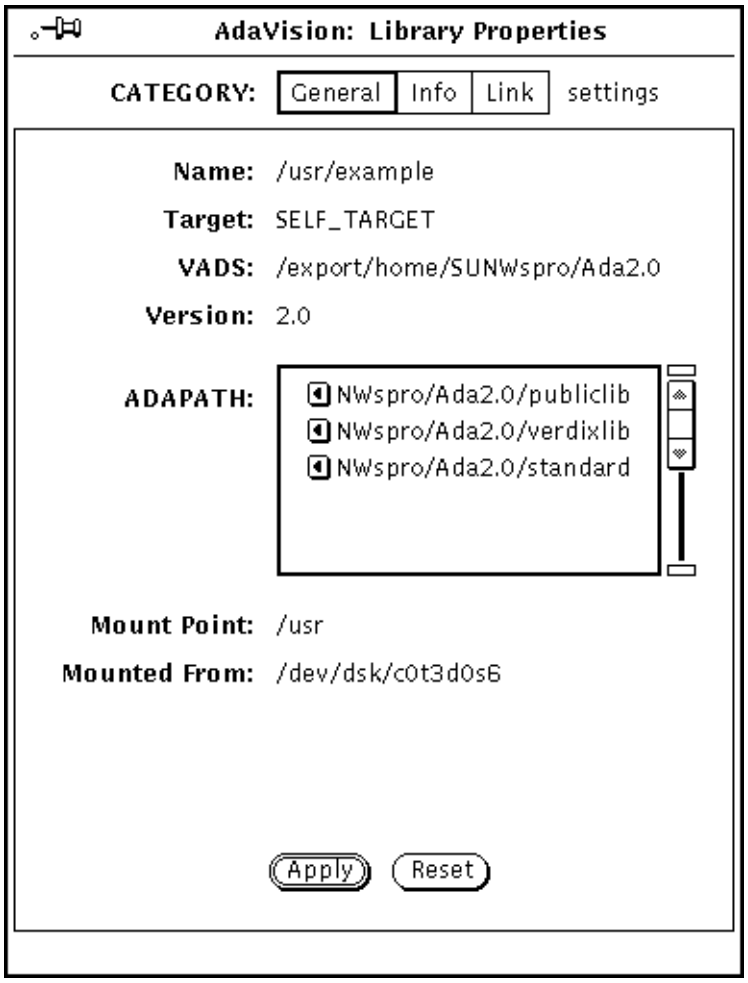
AdaVision maintains a set of Library Properties windows for each library on the library mode list. The General property window for a selected library contains general information about the library: its full path name, the target machine for which Ada generates code, the version of Ada used to compile the library, as well as the Ada mount point on your file system and the location from which it is mounted. This is read-only information.

This property window also displays the library's ADAPATH. (See "ADAPATH Editing" on page 3-12.)

3.1.1 Steps for Obtaining Library Information

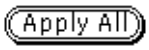
For the basic information about a library:

1. **Select the library from the list by clicking SELECT on its name.**
2. **Choose Selection from the Props menu to open the Library Properties window.**

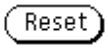


3.1.2 Apply All, Apply, and Reset Buttons

The Library Properties window contains a set of three subwindows under CATEGORY: General, Info, and Link, all of which are described in the sections that follow.



When you make a change to one of these subwindows, for example, General, and then click SELECT on another category—for example, Info—to switch to the Info Properties window, the Apply button changes to Apply All. This feature serves as a reminder that the settings in one or more of the other subwindows have been modified.



At any time prior to pressing Apply or Apply All in any of the three subwindows, you can press Reset to revert all the settings to their previous values when they were last applied. When you do so, the Apply All button then reads Apply again.

3.2 Controlling the Library List

The key to understanding library mode and library management is knowing how to control which libraries AdaVision lists in the display pane.

After a library is on the list, you can select and open it to view or work with its units. You may also want to see if a selected library has any Info or Link directives set (at the level of the selected library or higher up on its ADAPATH).

Controlling which libraries are on the list entails learning where or how to do the following:

- Start AdaVision.
- Add libraries to the list by importing them.
- Create new libraries.
- Relocate AdaVision to a different directory.
- Cut or delete libraries from the list.
- Clean Ada libraries from within AdaVision.

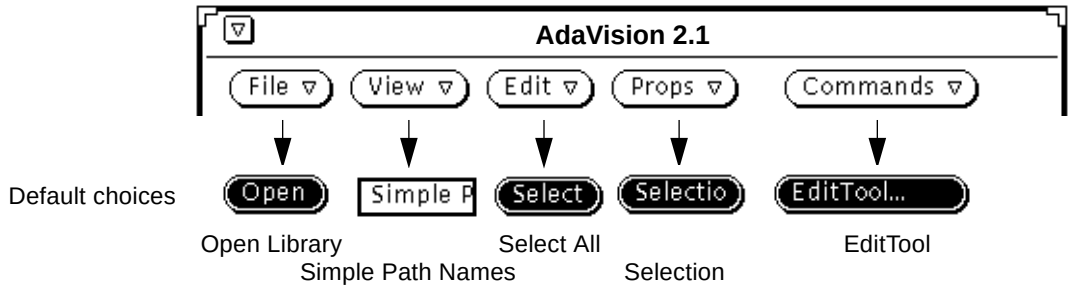
3.2.1 Library Mode at Startup

When you start AdaVision, it is always in its top-level library mode. AdaVision highlights the first of the three large mode selection buttons—a stand of library books.

In library mode, most of the items on the File and Edit menus are tools for controlling the library list. By choosing View ⟹ Full Pathnames, you can list libraries by their full path names instead of the default library names. The Commands menu items provide you with access to the other SPARCworks/Ada tools.

3.2.2 Menu Button Defaults

Note that each of the five menus has a default choice item. When you click SELECT on a menu button, AdaVision executes the default choice. The following figure shows the default choices.

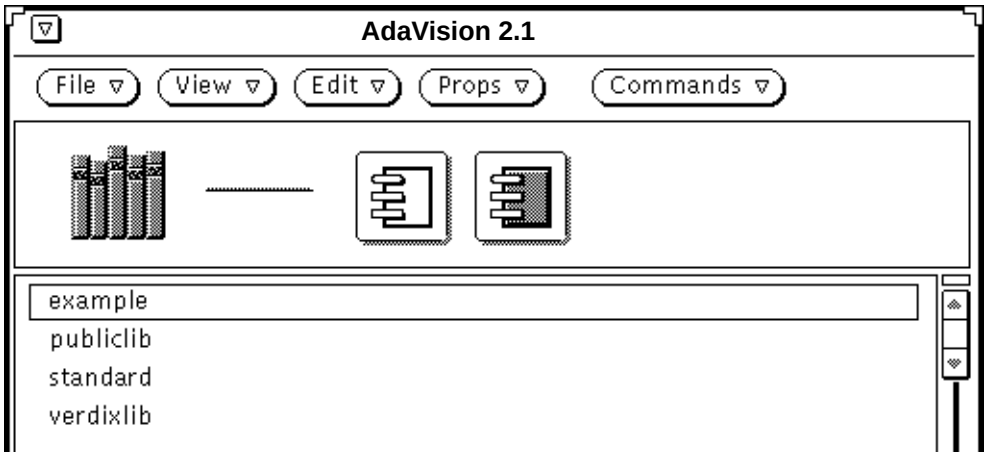


3.2.3 Library List at Startup

The most important variable affecting the library list is the location from which you start AdaVision. At startup, AdaVision looks for an `ada.lib` file in the current directory. If one is present, then AdaVision displays:

- The Ada library in the current directory
- Any Ada libraries on its ADAPATH

Here is the display pane when AdaVision was started from the example library, `example`.



The `example` library (in its simple path name form) is displayed in the pane together with the three libraries on its ADAPATH. The `example` library is selected (as indicated by the rectangle drawn around the library name).

3.2.4 Other Libraries

To work with libraries other than the one in the current directory and its associated ADAPATH libraries, you can choose one of these methods:

- Import libraries to the list by choosing Import to List on the File menu or by using drag-and-drop from File Manager.
- Relocate AdaVision by changing the path name in the Directory control in the AdaVision Properties window.

  The library list will then include the library in the new current directory plus its ADAPATH companion libraries, along with the libraries from the previous AdaVision location. That is, AdaVision does not cut libraries from the list when you relocate AdaVision.
- Start an additional AdaVision session.

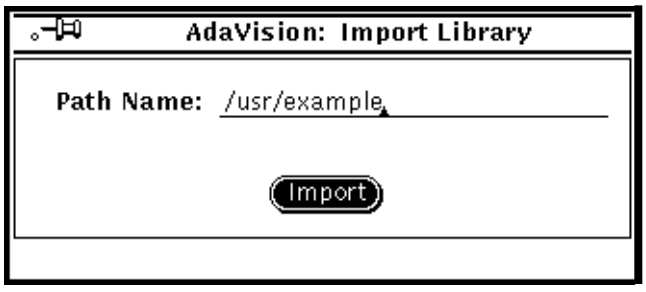
3.2.5 Library Import

The most efficient way to add a library to the list is to import it. When AdaVision imports a library, it adds it to the library list. It also adds any libraries that are on the imported library ADAPATH (if they are not already listed).

Note – Importing a *library* to the list simply adds the library name to the list. Do not confuse importing a library to the list with (1) creating a new library, or (2) importing LUs or CUs to a library from LU or CU mode (choosing the Import item in the File menu).

To import a library:

1. **Choose Import to List from the library mode File menu to open the Import Library pop-up window.**
2. **Type the full or relative path name of the library you want to import, then press the Import button.**



Importing assumes that the library you specify is a valid Ada library.

3.2.5.1 Using Drag-And-Drop with File Manager

An easy way to import an Ada library to the AdaVision library list is to use drag-and-drop with File Manager.

To import a library using File Manager:

1. **Click SELECT on the directory folder icon in File Manager that contains the Ada library you want to import to AdaVision.**

2. **Drag and drop the selected icon onto the AdaVision library mode display pane.**

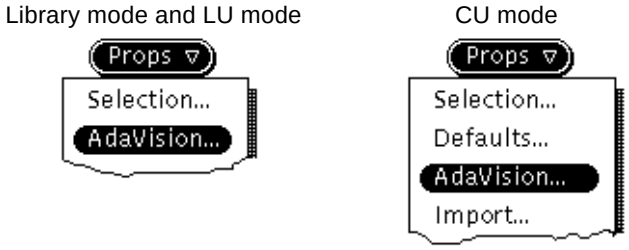
Be sure the icon is for a directory with a valid Ada library. File Manager currently does not identify Ada library folders for you. Also, be sure you are trying to drop the icon onto the library mode display pane, not one of the unit mode panes. You can drag and drop more than one library at a time.

3.2.5.2 Relocating AdaVision

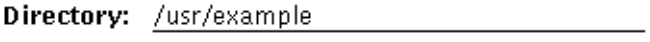
Another way to add libraries to the list is to move AdaVision to a different Ada directory.

To relocate AdaVision to another directory:

1. **Choose AdaVision from the Props menu to open the AdaVision Properties window.**



2. **On the Directory control text field, enter the path name to the directory where you want to relocate AdaVision, then press the Apply button.**



This procedure actually changes the file system directory in which AdaVision resides. On the AdaVision display pane, you see that AdaVision *adds* to the list whichever Ada library it finds in the new location, *plus* any of its ADAPATH companion libraries. Note that AdaVision does *not* remove from the list those libraries added at the former location. Visiting a series of Ada libraries using the Directory control builds up the library list.

3.2.5.3 Starting an Additional AdaVision Session

If you work regularly in two different Ada library directories, you may want to start a session in each directory. You can have both AdaVision sessions open and active on the Desktop at the same time.

3.2.6 Removing Libraries from the Library List

To remove libraries from the library mode list:

1. **Click SELECT on the library name.**
2. **Choose Cut from List from the Edit menu.**

You can select more than one library by clicking ADJUST on each. To select all the libraries on the list at once, choose Select All from the Edit menu. You can then remove them by choosing Cut from List.

Remember that cutting libraries from the AdaVision library list has nothing to do with deleting Ada libraries. You can cut and add to the list freely.

3.2.7 Library Deletion

To delete an Ada library from within AdaVision:

1. **Switch to library mode and click SELECT on the path name of the library you want to delete.**
2. **Choose Delete from the Edit menu.**
   At this point, a dialog box is posted, asking if you really want to delete the object(s).

Caution – Be sure you do want to delete the object(s) before you press the Yes button because there is no undo facility. When you delete a library, AdaVision performs an Ada `a.rmlib` command.

3.2.8 Library Cleaning

In AdaVision, you can “clean” an Ada library. Cleaning a library means removing 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 directories: `.imports`, `.nets`, `.lines`, `.objects`. For details on Cleanlib, see the section on the `a.cleanlib` command in the SPARCompiler Ada documentation.

To clean the selected Ada library from within AdaVision:

1. **Switch to library mode and click SELECT on the library you want to clean.**
2. **Choose Cleanlib from the Edit menu.**

3.2.9 AdaVision Restart

When you restart AdaVision in the same directory in which you last used it, AdaVision does *not* list libraries that you may have added to the basic list during the previous session. It starts again with the basic list—the library in the current directory plus its ADAPATH libraries.

There is no limit on how many library objects you can list in a single AdaVision session.

3.3 Creating a New Library

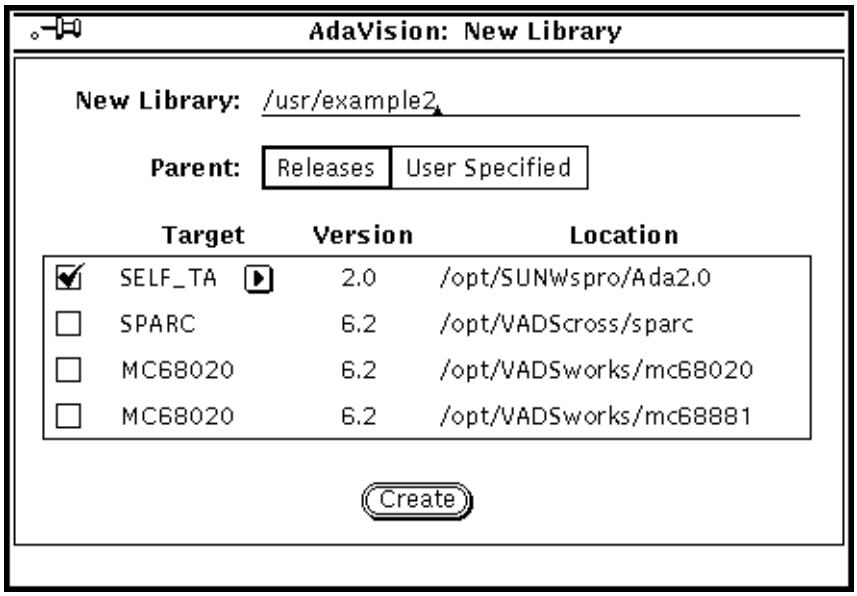
The library import operations previously described apply to existing Ada libraries. This section describes how to create a new library.

If AdaVision does not find an `ada.lib` file in the current directory, the AdaVision display pane is empty when it opens. Typically, you open AdaVision in a directory that contains an Ada library. The exception is when you want to create a *new* Ada library.

3.3.1 Library Creation

To create a new Ada library:

1. **Choose New Library from the File menu to open the New Library pop-up window.**



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

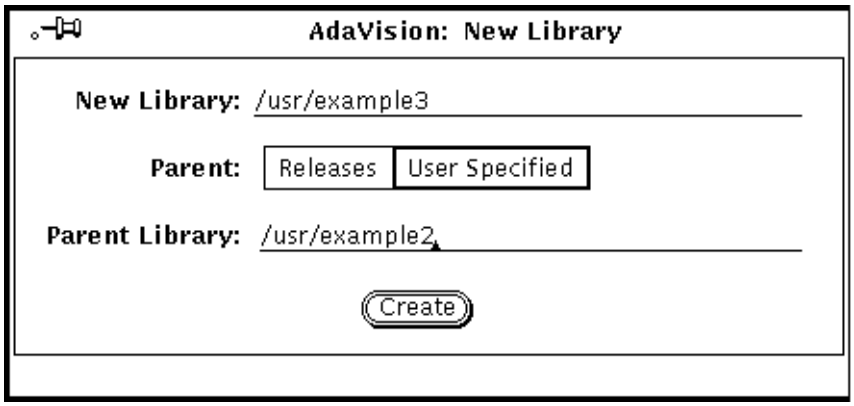
3. **Press Create.**
   AdaVision then proceeds to create a new library. When the process is complete, the new library path names are added to the ADAPATH and displayed in the AdaVision display pane.

The New Library window uses information in the `/etc/VADS` file to list all of the Ada target releases available to you (which is why it is a good idea to keep it up-to-date). In the Parent control, Releases is selected to mark the release pointed to by the first line in `/etc/VADS`, thereby designating it the default parent library. You can change the parent library by clicking SELECT in the box next to whichever release you want AdaVision to use.

To specify a parent library other than one contained in an installed release of SPARCompiler Ada (or VADScross) listed in the display pane, follow the steps below:

1. **Set the Parent control to User Specified.**
   This step gives you a Parent Library text field.

2. **Enter the name of the parent library.**



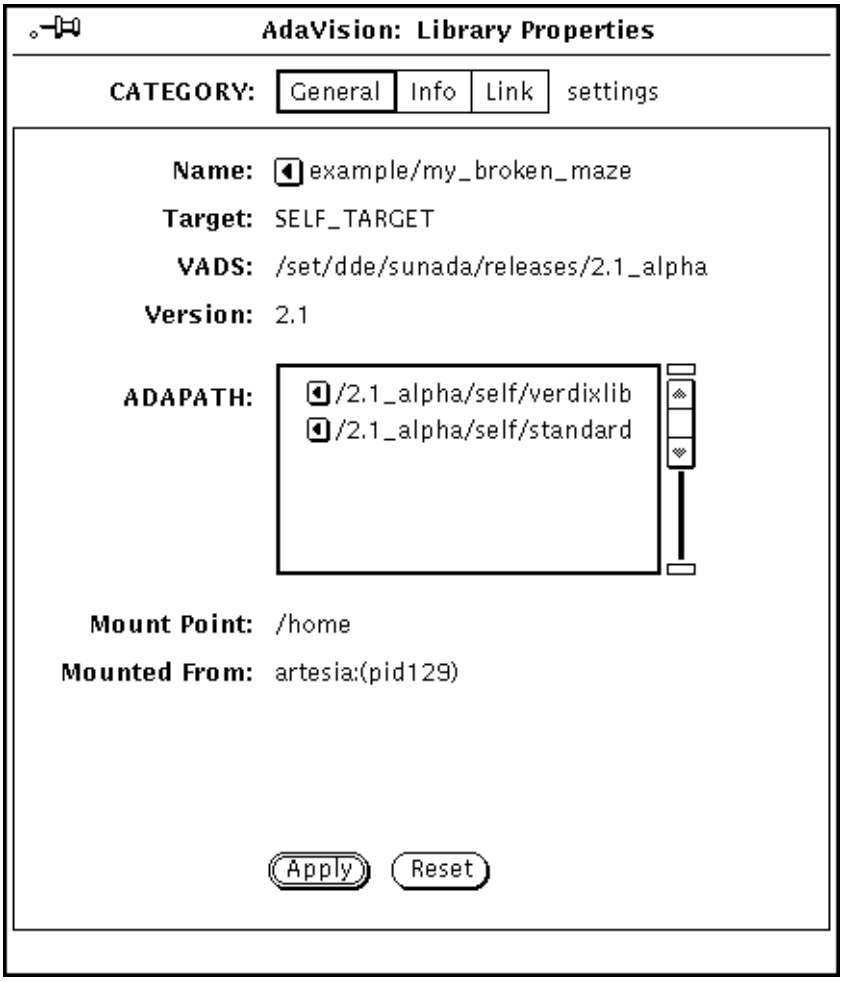
After you complete steps 1 and 2 immediately above, click the Create button to continue.

3.3.2 ADAPATH Editing

When you edit the ADAPATH for a selected library, you can add, delete, or reorder the libraries. ADAPATHs in AdaVision are displayed in a scrolling list in the selected Library Properties window.

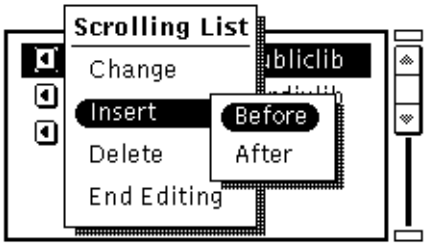
To edit a library ADAPATH:

1. **In library mode, click SELECT on the library path name.**
2. **Choose Selection from the Props menu to open the Library Properties window.**
   If the Props menu is already open to another set of controls, click SELECT on General under CATEGORY to display the ADAPATH control and scrolling list.



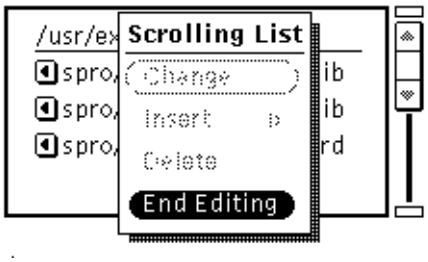
3. **In the ADAPATH list, click SELECT on the library name that you want to come *after* the library you are going to add.**
   This action selects the library.

4. **Press MENU to invoke the floating Scrolling List menu, then choose Insert ⟹ Before.**
   This step causes a text field to be displayed in the ADAPATH.



5. **Type the path name of the new library in the text field.**

6. **Click MENU to invoke the Scrolling List menu again and choose End Editing.**



7. **Press the Apply button.**
   When you press Apply, AdaVision analyzes the changes to the ADAPATH and alters the library list accordingly.

As you can see from the Scrolling List menu, you can change or delete a library from the ADAPATH list in addition to adding a new library to the list. (As a side effect of editing the ADAPATH, AdaVision adds libraries associated with the new ADAPATH to the library mode list.)

If you delete a library from an ADAPATH, the libraries associated with the deleted library remain on the AdaVision library mode list. You must delete libraries from the list by choosing Cut from List on the Edit menu.

3.4 Setting Info and Link Directives

Info and Link directives control work the same way. If a box is checked, then the directive is set at the level of the selected library. If the box is not checked, then the value for that directive is either the system default (which may be “no value” in the case of WITHn directives) or the value shown (which was set in a library higher up on the ADAPATH of the selected library).

Normally, you do not have to deal with Info and Link directives. Someone sets the directives early on in a project and they tend to stay set. Occasionally, you might want to change either or both types of directives. In AdaVision, you change Info and Link directives from the subwindows in the selected Library Properties window.

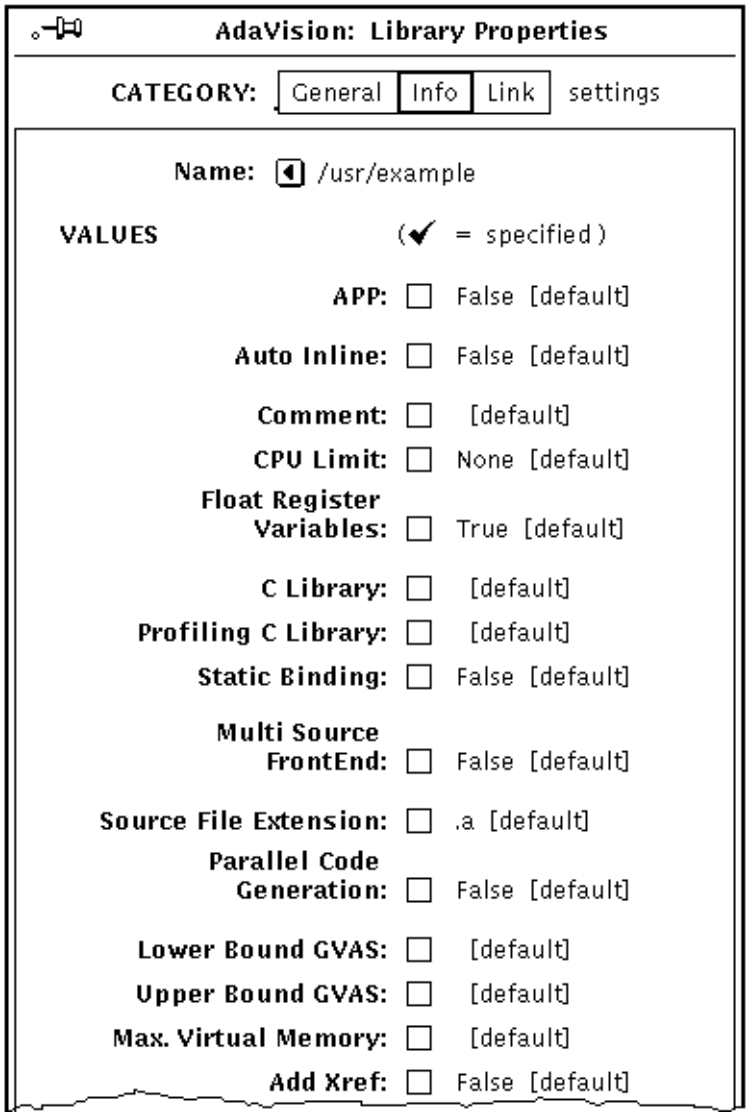
The property windows for the Info and Link directives list the directives set in the selected library and the directives that the compiler and the linker see when compiling and linking in the current library. The compiler and linker see directives set in the current library plus any directives set higher up on the ADAPATH of the current library. The SPARCompiler Ada documentation provides detailed descriptions of these directives.

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

3.4.1 Info Directives

To check Ada Info directives:

1. **Click SELECT on the library path name from the library mode display pane.**
2. **Choose Selection from the Props menu to open the Library Properties window.**
3. **Click SELECT on Info under CATEGORY to display the Info directives subwindow.**



(Continued on next page)

The first line, Name, shows the full path name of the library that owns the property window (that is, the selected library). A separate Library Properties window exists for each library object.

The second line is a header that reminds you that a check mark inside of the box means that the value for that Info directive is set *in the selected library.*

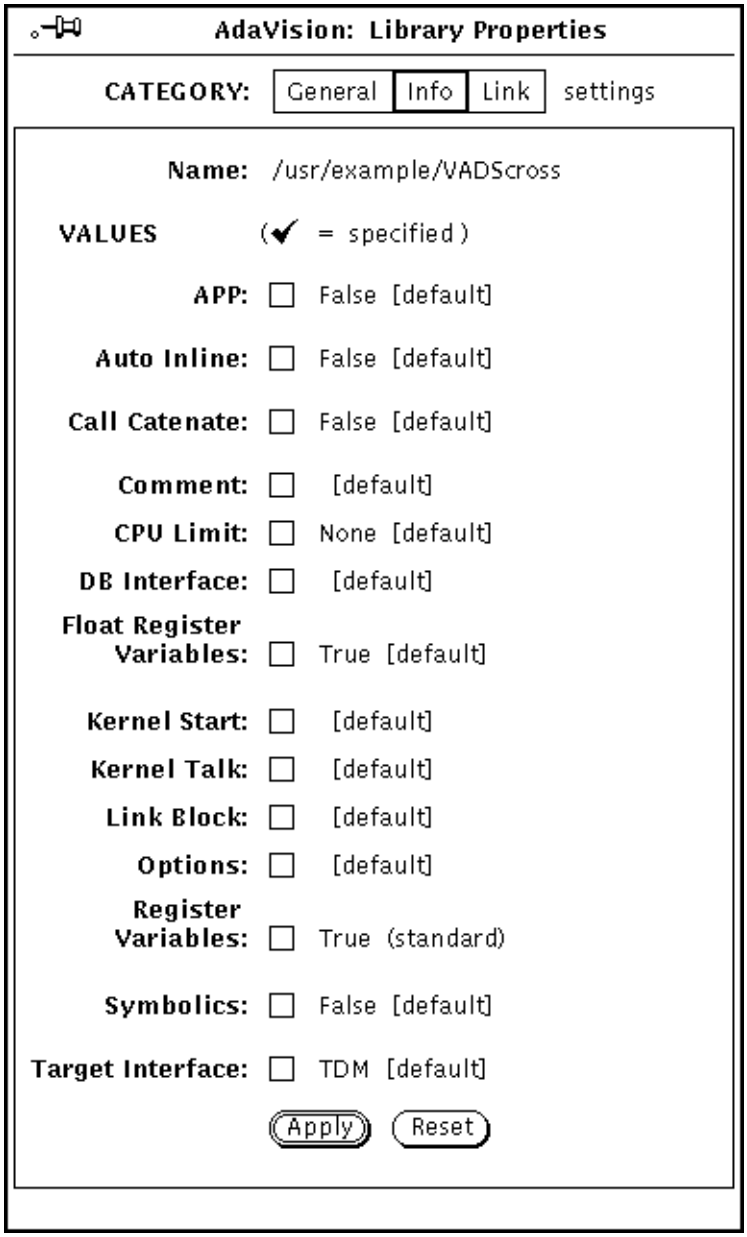
The Info subwindow shown is for the `example` library. Notice that none of the boxes is checked. This means that none has been set in the `example` library.

Now, look at the value to the right of each of the Info directives. Each value is followed by `[default]`, which indicates that the value is the system default value.

A `(standard)` value indicates that, in the case of this library, the value for the VADS control, `/home/sunada`, was set in the standard library, which is higher up in the ADAPATH.

The selected Ada library inherits Info directives that are set higher up in its ADAPATH. If AutoRead is on and someone changes a directive in a library higher up in the ADAPATH, AdaVision updates the current directory's Info Library Properties window.

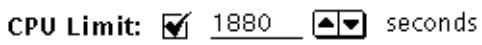
The Info Properties window for cross-development offers more controls. The following example is from a VADScross product.


AdaVision: Library Properties
CATEGORY:
General
Info
Link
settings
Name:
/usr/example/VADScross
VALUES
(✔ = specified)
APP:
False [default]
Auto Inline:
False [default]
Call Catenate:
False [default]
Comment:
[default]
CPU Limit:
None [default]
DB Interface:
[default]
Float Register Variables:
True [default]
Kernel Start:
[default]
Kernel Talk:
[default]
Link Block:
[default]
Options:
[default]
Register Variables:
True (standard)
Symbolics:
False [default]
Target Interface:
TDM [default]
Apply
Reset

To set an Info directive for the selected library:

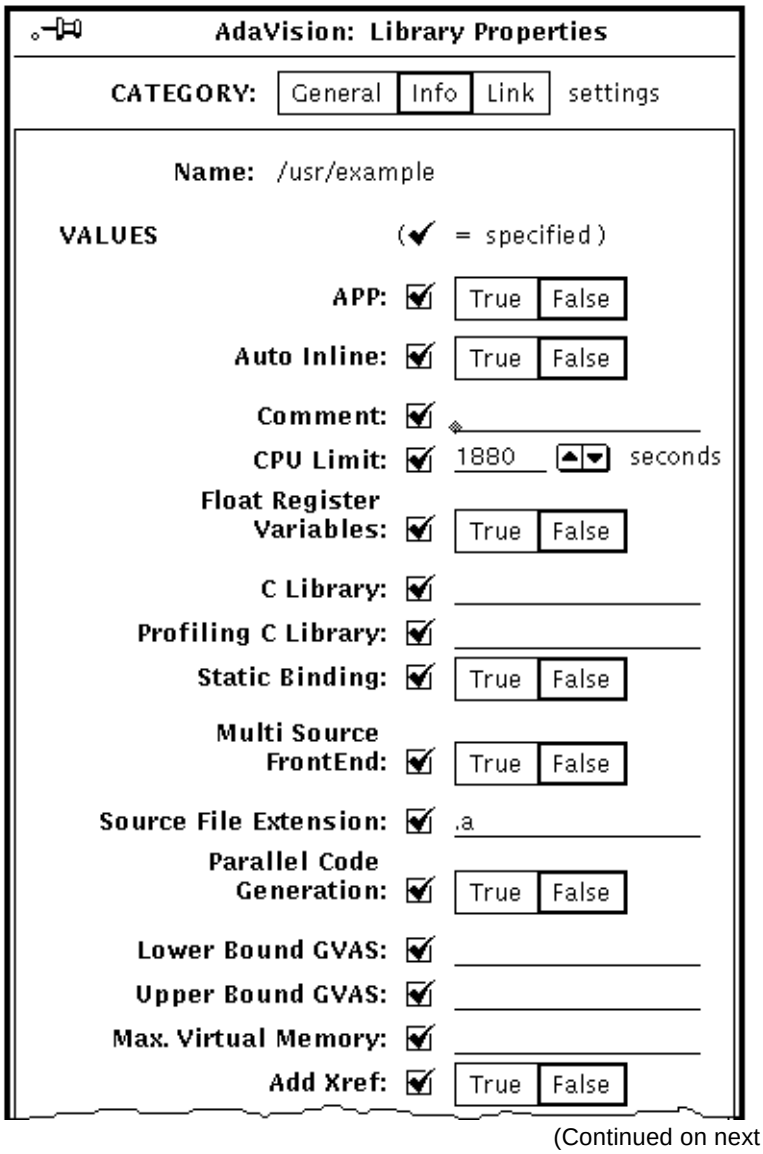
1. **Check the box next to the Info directive you want to specify by clicking SELECT in the box.**
   The current setting, if any, is highlighted. When you check the box for Comment, a text field is displayed for you to type in comments for inclusion in the Ada library.

   When you check the box for CPU Limit, the standard value of 1880 seconds is displayed. Use the Up or Down scrolling button to adjust the value.

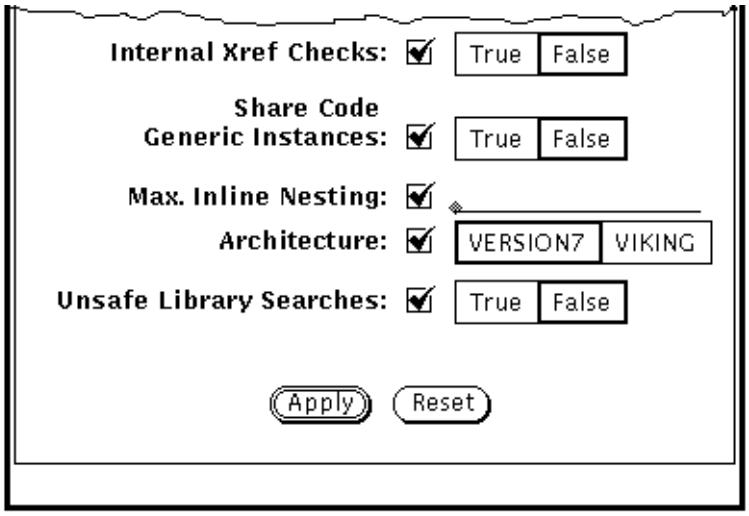


2. **Set the values for the check-marked Info directives and press Apply.**

3. **The following figure shows the** `example` **Info directive subwindow with all the directives set.**
   The currently specified value is always highlighted for toggle-type selections (not for the 1880 seconds or the `.a`, and so forth).



(Continued on next page)

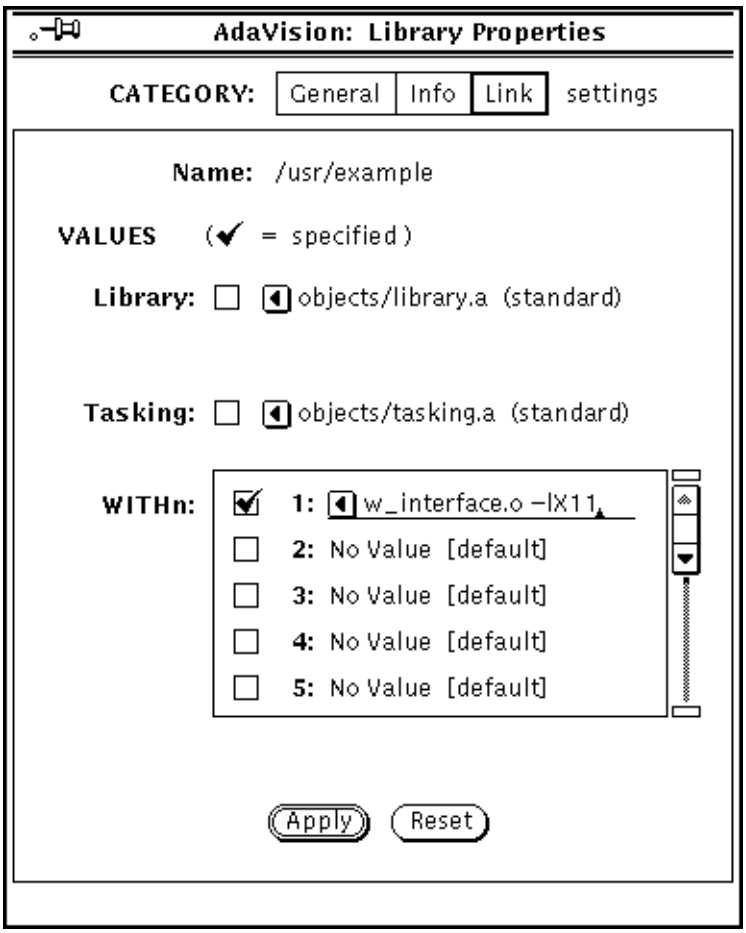


3.4.2 Link Directives

You use the Ada Link directives to add additional dependent files to links performed in the selected library and its child libraries. Set Link directives on the third of the Library Properties subwindows.

To check the selected library's Link directives:

1. **Click SELECT on the library path name on the library mode display pane.**
2. **Choose Selection from the Props menu.**
3. **Click SELECT on Link under CATEGORY to display the Link Properties subwindow.**



As stated before, the Link directive controls work the same way as the Info directives. (See the previous section for details.)

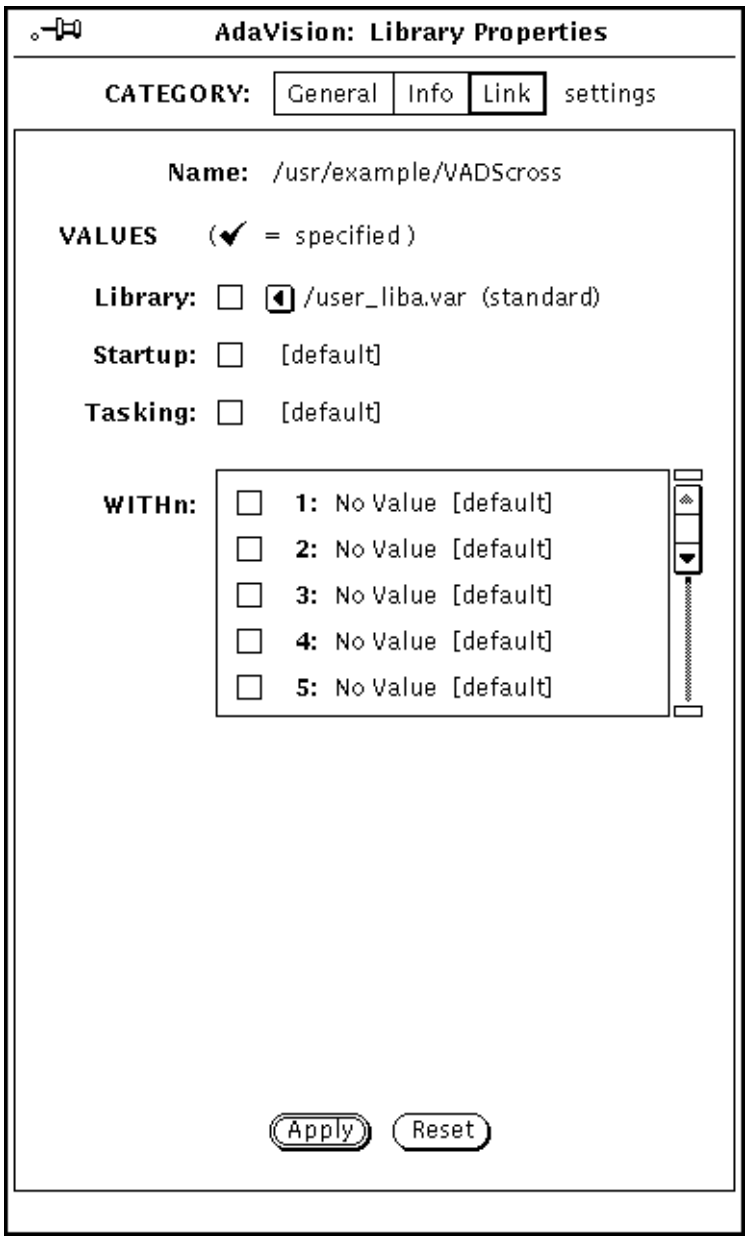
3.4.3 WITH Link Directives

The AdaVision interface for setting Link directives preserves the Ada requirement that directives be numbered consecutively, starting with the number 1.

In the AdaVision interface, this requirement means that Ada stops reading down the list when it encounters a blank (numbered) line. If you remove a directive from the middle of the list and you want Ada to read in the remaining WITH directives, you must move each item up, closing the gap. Of course, if you enter your directives in a hierarchical order, you can tell Ada to stop reading the remainder of the directives on the list by simply unlocking the WITHn directive at the correct level in the hierarchy.

Note, however, that even if you unchecked a WITHn directive in the selected library, it may still be active because it is set in a parent directory higher up in the ADAPATH.

A cross-development environment offers its own Link directives property subwindow. The following example from a VADScross product offers an additional directive, Startup, which is the name of the object file that contains the program startup routine.


AdaVision: Library Properties
CATEGORY:
General
Info
Link
settings
Name: /usr/example/VADScross
VALUES (✔ = specified)
Library: /user_liba.var (standard)
Startup: [default]
Tasking: [default]
WITHn:
1: No Value [default]
2: No Value [default]
3: No Value [default]
4: No Value [default]
5: No Value [default]
Apply
Reset

Using AdaVision Unit Modes 4

The two AdaVision unit modes, library unit (LU) mode and compilation unit (CU) mode, provide access to the selected units. Within these two modes, you edit code in a unit or write code for new units; make, compile, and link units; sort and filter units; and print files or screens.

The basic distinction between LU mode and CU mode is one of *focus*. In LU mode, you can focus on a family of CUs by displaying them as a single icon. CU mode gives you access to all of the units in the selected library, displayed as separate icons. (See “Understanding AdaVision Modes” on page 2-4 for more information about modes.)

This chapter describes working with Ada units in the AdaVision unit modes. It is organized into the following sections:

4

4.1 Opening a Library To Display Its Units

When you first open a library, AdaVision displays graphical objects for the units in the library. You can open a library at either of two levels: at the level of its LUs (LU mode) or at the level of its CUs (CU mode). At either level, you then select one or more unit objects to work with. You may want to perform some operation on the unit as it stands, for example, compile, link, or debug, or you may want to open a unit to view or edit its underlying source code.

To view or work with units in an Ada library:

1. **Click SELECT on the path name of the library displayed in library mode.**
2. **Press either the LU mode or CU mode control button.**
   Alternatively, you can select the library and then choose Open from the File menu, or double-click SELECT on the library path name to open the library into LU mode.

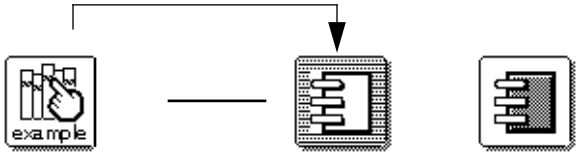
You can open only one library at a time. If you select more than one library, AdaVision displays the selected library highest on the list. If no library is selected, then the unit mode buttons are inactive.

4.1.1 Mode Changes

You can switch freely from one viewing mode to another. The units that AdaVision displays when you move from LU to CU mode depend on whether you select an LU unit before switching to CU mode.

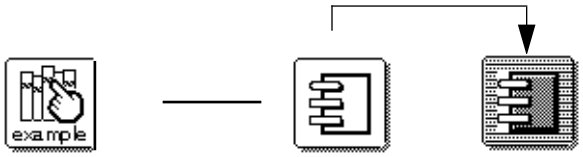
If you switch from library mode to LU mode, AdaVision displays an object for each LU in the library.

Library mode to library unit mode



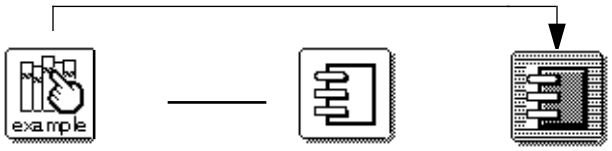
If you select an LU object and then switch to CU mode, AdaVision displays an icon for each CU belonging to the library unit *object*; that is, the LU spec, plus its body (if any), and subunits (if any).

Library unit (LU) mode to compilation unit (CU) mode



If you switch directly from library mode to CU mode, then AdaVision displays an icon for each CU in the library.

Library mode to compilation unit (CU) mode

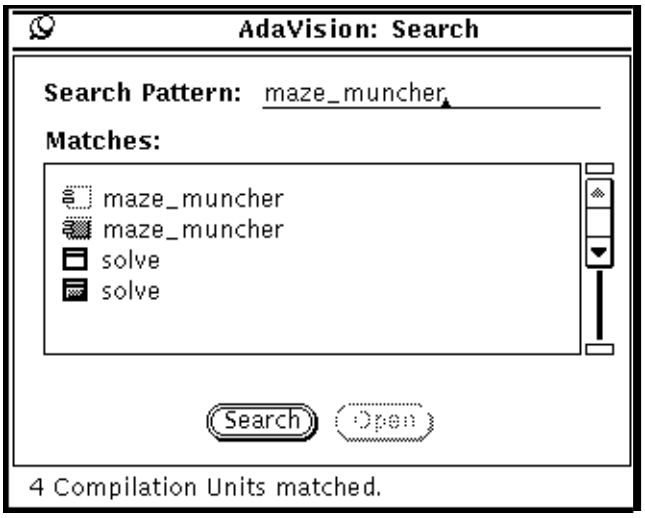


4.1.2 Search Facility

CU mode has a Search facility to locate source files of CUs that match a string pattern you enter.

To use Search while in CU mode:

1. **Choose Search from the Commands menu.**
   AdaVision then opens a pop-up window with a text field entitled “Search Pattern.”

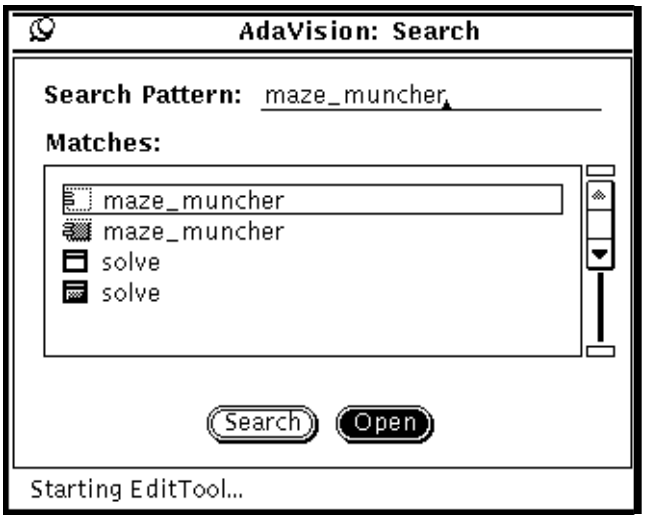


2. **In the Search Pattern text field, type the string pattern you want to search for.**

3. **Press the Search button.**

When the search is complete, AdaVision lists the CUs whose source file(s) contain a match of the string you entered, as well as the unit names. The number of CUs that match the string pattern is printed on the left footer.

If you click SELECT on an icon, the Open button becomes active.



Press the Open button to load the source file of that unit into the default editor.

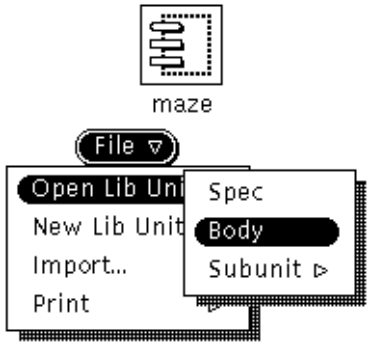
4.2 Editing Units

You edit or browse an existing unit by selecting and then opening its object. When you open a unit object, AdaVision loads the source file containing the unit into a window running an editor. You specify the editor with the Editor control in the AdaVision Properties window. The default editor is EditTool—the SPARCworks/Ada-extended version of the OpenWindows Text Editor tool.

You can open more than one unit object at a time. AdaVision opens a separate window editor for each unit.

Each window editor opened from within AdaVision operates independently of AdaVision. You can treat the window the same as you would any other window opened from the Workspace Programs menu. For example, if you start EditTool to work on an Ada unit, you can replace the current unit object source file (the current document) with any other editable file in that same window. You can move, resize, and close the windows to icons.

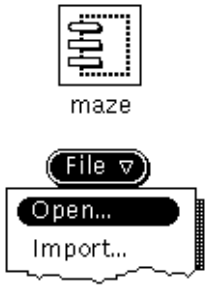
4.2.1 Unit Object Browsing or Editing



To open a unit object in LU mode to browse or edit it:

1. **Click SELECT on LU and on the LU object.**
2. **Select Open Lib Unit from the File Menu or from the floating pop-up menu.**
3. **Choose Spec, Body, or Subunit from the submenu.**
   Spec is the default choice. If you choose Subunit, pull to the right again, and choose one of the subunits on the list.

Note – If you double-click SELECT on the LU object or click SELECT on the File menu button instead of performing step 2, AdaVision chooses the default item: Open Lib Unit ⟹ Spec.



To open a unit object in CU mode to browse or edit it:

1. **Click SELECT on the CU object.**
2. **Choose Open from the File menu or the floating pop-up menu to open the source file containing the unit.**

Note – Open is the default choice on the File menu, so if you just click SELECT the File menu button, it chooses the Open item for you.

To open objects quickly, *double-click* SELECT on the object icon.

AdaVision opens a window for each selected object, using the editor specified in the AdaVision Properties window.

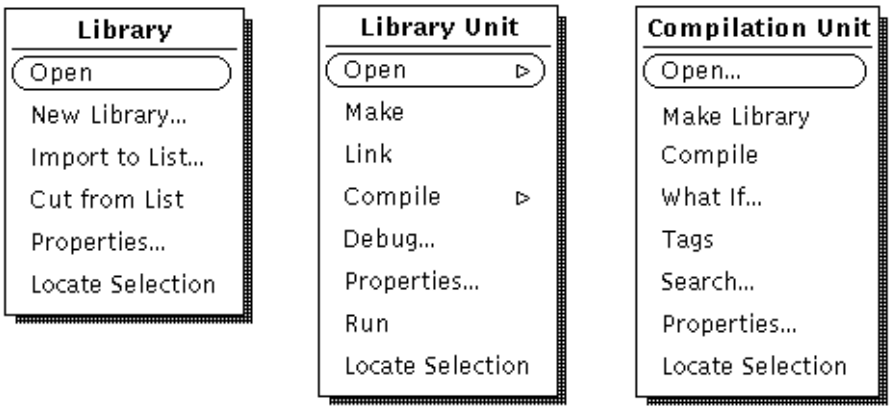
4.2.2 Unit Icon Dragging and Dropping onto the Editor Display Pane

While an editor is running in a window, you can edit an Ada unit by dragging and dropping the unit icon onto the editor display pane. The editor then loads the source file containing the unit. If a file is open in the editor, the unit you drop onto the pane replaces it. If changes are unsaved, the editor asks if you want to save the changes before closing and replacing it.

4.2.3 Floating Pop-up Menus

In each of the modes, the AdaVision display pane has a floating pop-up menu. The pop-up menu contains a set of the most commonly used items from the main menus. You open the floating pop-up menus by pressing MENU.

Here is what the three floating pop-up menus in the three modes look like. All items are shown as being active; in reality, some of them are dimmed if they do not apply to the selection.



4.3 Viewing Unit Properties

For basic information about an LU or a CU, click SELECT on a unit, and then open its property window.

4.3.1 CU Properties

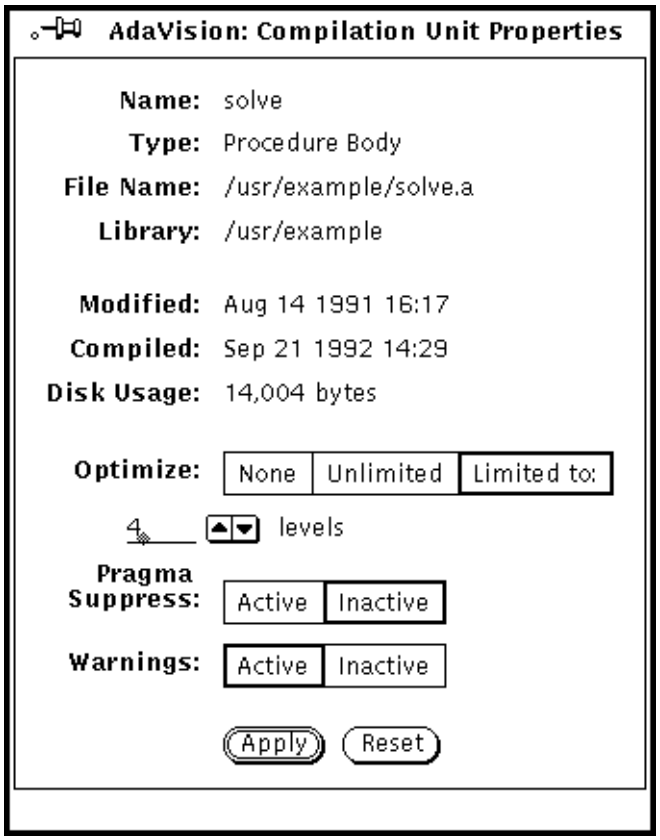
To display information about an Ada CU, in CU mode:

1. **Click SELECT on the unit icon.**

2. **Choose Selection from the Props menu to open the Compilation Unit Properties window.**

Note – Selection is the default choice on the Props menu. If you just click SELECT on the Props menu button, Selection is chosen for you.

Here is a diagram of a Compilation Unit Properties window.



The Compilation Properties window shows you:

- Name, type of unit, file, and library name
- When the unit was last changed and last compiled
- Size of the unit

The Compilation Unit Properties window also contains controls for three compiler options:

- Level of optimization
- Pragma Suppress, active or inactive
- Warnings, active or inactive

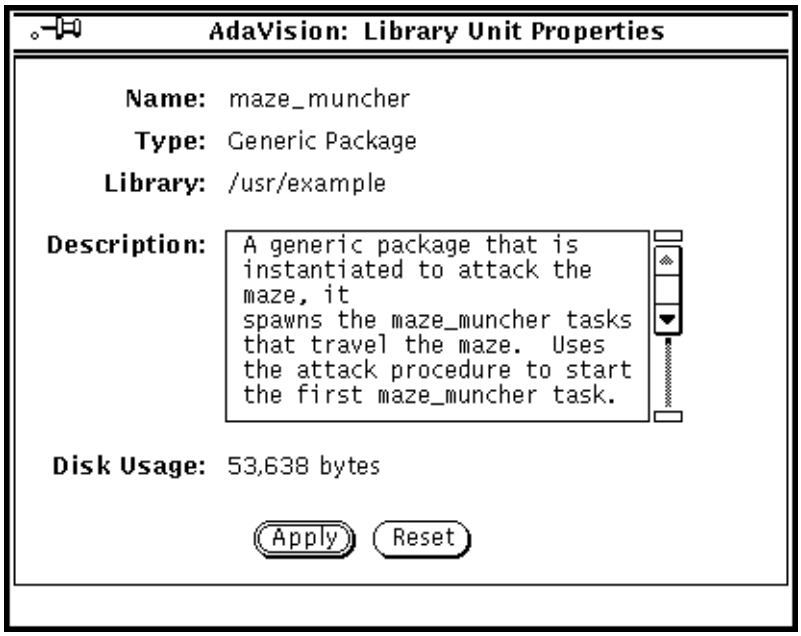
You may want to reset these controls before recompiling the unit. (See "Compiler Output Option Settings" on page 4-25 for more information about these settings.)

4.3.2 LU Properties

For information about LU objects, in LU mode:

1. **Select one or more LU object(s).**
2. **Choose Selection from the Props menu to open the Library Unit Properties window.**
   For each LU selected, AdaVision opens a Library Unit Properties window.

Note – Selection is the default choice on the Props menu. If you just click SELECT on the Props menu button, Selection is chosen for you.



The Description control has a multiline text field for entering notes about the LU.

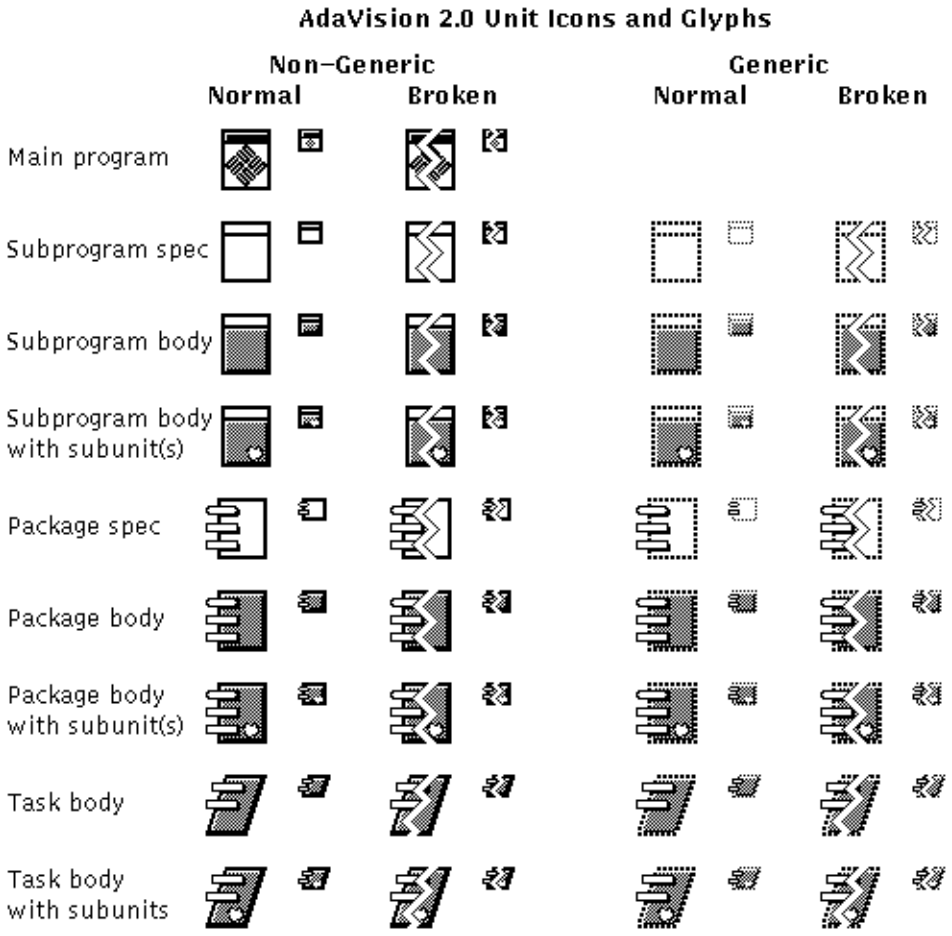
Note that the number of bytes reported by Disk Usage is the total for all of the CUs in the LU *object*, not just the number of bytes in the LU spec. To see the disk usage figure for a specific CU, go to CU mode, select the icon, and then open the Compilation Unit Properties window, as shown under "CU Properties" on page 4-7.

The Library Unit Properties window applies to all LU objects that are *not* main programs. The properties window for a main program is described in "Main Program Properties" on page 4-32.

4.4 Understanding AdaVision Icons

Unit images in AdaVision graphically represent the kind and compilation state of an Ada unit. Depending on the display mode, they are shown as large images (icons) or as small images (glyphs).

Here is a diagram showing all AdaVision icons and glyphs.



4.4.1 Three Types of Icons

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

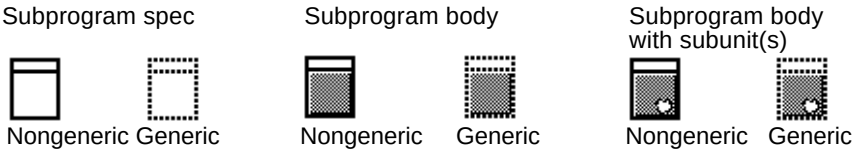
Generic unit icons look like their nongeneric counterparts, except that they are bordered by dotted rather than solid lines.

Specs and bodies for subprograms and packages are displayed as separate icons. Since a task is not an LU, there is no task spec, only task bodies. Specs are white; bodies are shaded. A body that contains subunits has an amorphous blob in the lower right-hand corner of the icon. Subunits are displayed as bodies, with or without the amorphous blob, as appropriate.

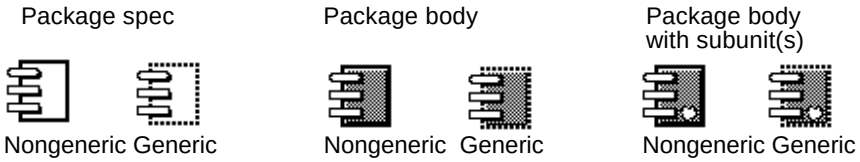
You distinguish subunits from bodies by subunit names, which always contain a period (.). For example, in the `example` program, the `maze` unit consists of one body, `maze`, which in turn consists of three subunits. The subunit names are: `maze.compute_new_maze`, `maze.refresh_window`, and `maze.compute_new_maze`.


Main program

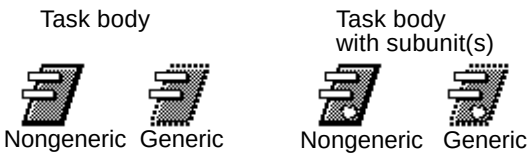
Subprograms, which are functions and procedures that you call, are represented as follows. A subprogram that is a main program has a Sun logo in the center of the icon.



Packages are represented as shown in the following diagram, with three white bars to denote the visible package interface.



Tasks, which are objects of parallel execution in an Ada program, are represented as parallelograms—tilted at an angle.



4.4.2 "Broken" Icons

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.

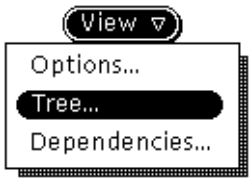
The broken state alerts you that the unit is obsolete and has to be recompiled. "Broken" icons are shown with zigzag lines down the middle.

The following diagram examples show "broken" icons.

4.5 Displaying Special Graphs of Units

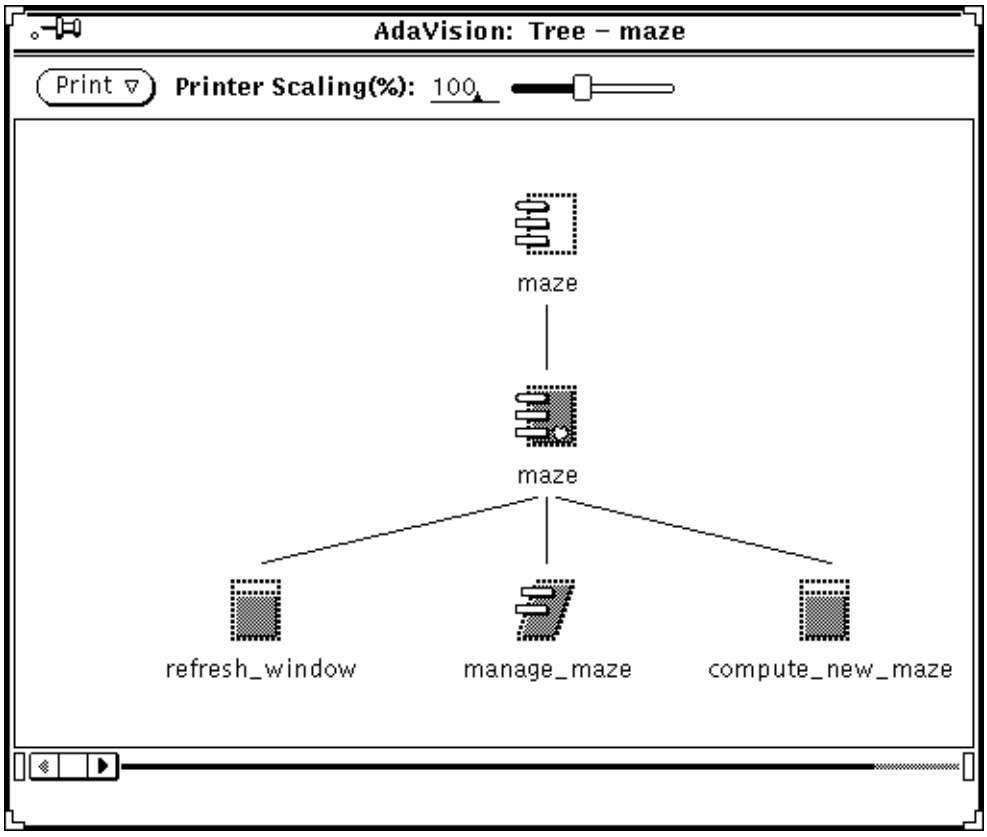
The two unit modes also provide graphics tools that help you examine the structure of a selected library and the implications of making changes to it.

4.5.1 Tree Graph of Units in an LU

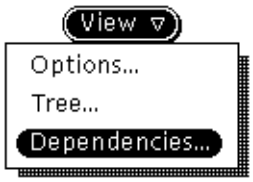


In LU mode, choosing the Tree item on the View menu opens a pop-up window with a two-dimensional graph showing the relationships among an LU and its associated secondary units and subunits (if any). The tree shows all of the units that make up an LU object.

Use the horizontal scrollbar to browse a tree structure that occupies more than one full screen.



4.5.2 Dependencies Graph of Units in an LU

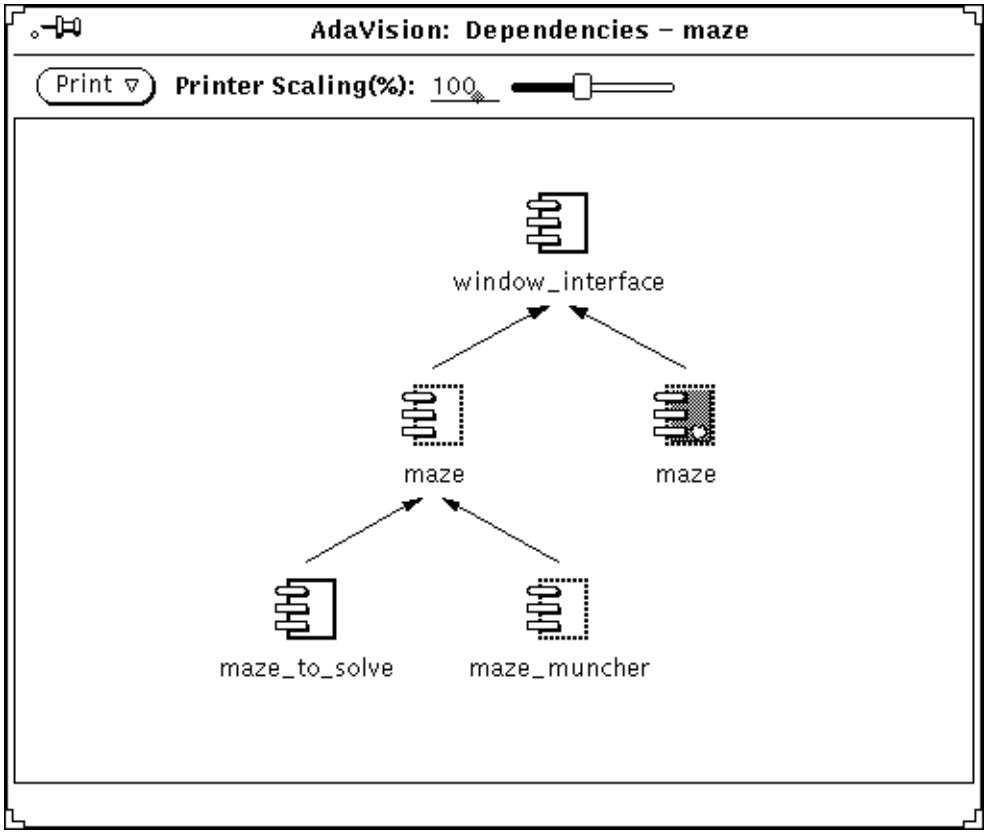


Choosing the Dependencies item on the View menu starts a two-dimensional view of unit dependencies, that is, units which contain WITH clauses or inline dependencies.

The Dependencies graph is displayed in a pop-up window, which you can keep on your workspace throughout an AdaVision session.

When you select a different LU object and choose Dependencies, AdaVision opens a new window for displaying the graph.

In the following example, you see the graph generated for the `maze` LU object: the arrows point from the dependent units to the specification units on which they depend. Dependency graphs show the dependencies for each of the CUs in the LU; in this example, for the `maze` LU object, the graph shows dependency relationships of `maze`.



4.5.3 Print Menu and Printer Scaling in Graph Windows

A Print button is present in the Tree and the Dependencies windows with a submenu that consists of two items, Portrait and Landscape. You can print a hard copy of the tree or dependency graph in portrait or landscape mode.

Before dispatching print jobs, you can scale the icons for printing between 40% to 180% by moving the slider under the Printer Scaling control. The text field, defaulted at 100, changes to reflect the percentage of scaling as you move the slider to the left or right.

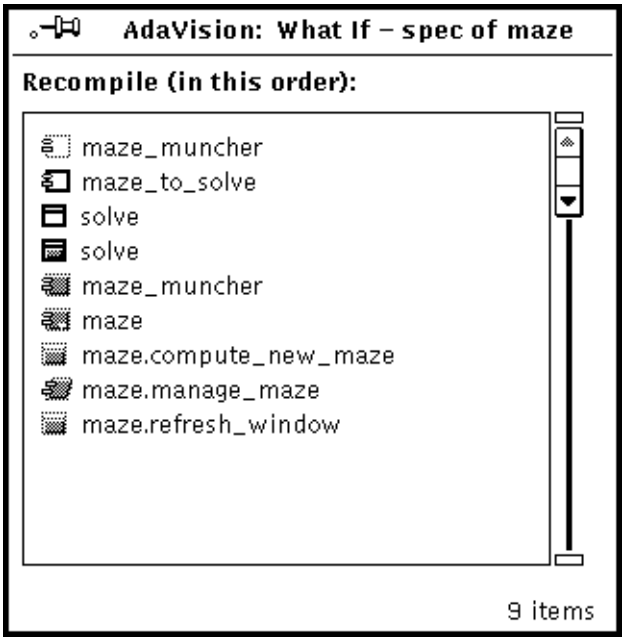
4.5.4 Compilation Effects (What If)

Before compiling a unit, you may want to choose What If in the CU mode View menu to see a list of units you need to recompile if you change the selected one. The list also shows you a valid order in which to recompile them.

To use What If:

1. **Switch to CU mode.**
2. **Click SELECT on the unit you want to compile.**
3. **Choose What If from the View menu.**
   When you choose What If, AdaVision opens a separate window displaying an ordered list of units that have to be recompiled if you compile the selected unit.

Here is a What If window for the `example` program:



If the AutoRead facility is on (the default setting), AutoRead automatically updates the contents of Dependencies windows and What If windows.

AutoRead causes AdaVision to reread all of its Ada libraries at regular intervals. The default interval is 10 seconds, but you can change it in the AdaVision Properties window. In that window, you can also turn AutoRead on and off. See “AutoRead” on page 1-11.

4.6 Importing Units

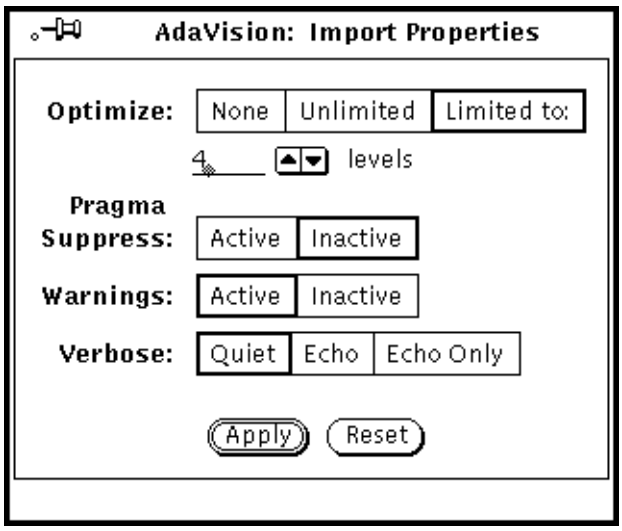
When you import units to a library in AdaVision, an `a.make -f` command is performed on the files you specify in the Import pop-up window.

4.6.1 Import Properties

Before importing units, you may want to check the settings for the options to the compiler and modify them, if necessary.

To set compiler options for importing units, in either LU or CU mode:

1. **Choose Import from the Props menu to open the Import Properties window.**



2. **Set the value you want for each of the four option controls and press Apply.**

Note that Import Properties options are inherited by the units you are importing, so that when you compile the unit again, it uses these options again. You may want to check the settings on each unit Compilation Unit Properties window after the import.

When you import units, AdaVision puts up a new icon for each. If AutoRead is set to a short interval, you see the unit icons being displayed on the display pane as they are compiled into the library.

4.6.2 Unit Import Facility

The Import item is available on the File menu in both LU mode and CU mode. You can also drag and drop files containing Ada units from File Manager to the LU or CU mode display pane.

LU mode

CU mode

To import units into a library, in either LU or CU mode:

1. **Choose Import from the File menu to open the Import window.**

2. **In the File Names text field, type the name of the source file(s) containing the units you want to import.**
   You can use wildcard expressions.

3. **Press the Import button.**
   AdaVision then proceeds to import the unit(s) into the library.

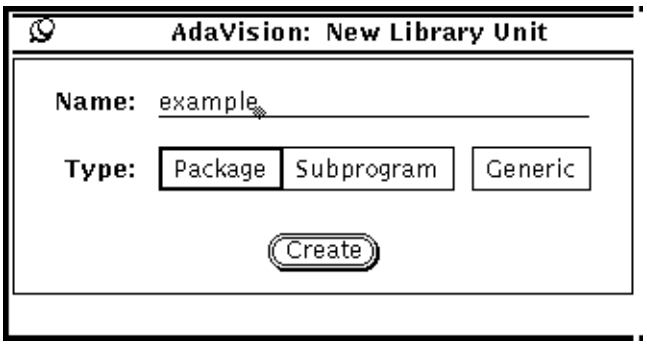
4.7 Creating New LUs

You can create new LUs in LU mode. Supply a name for a new unit and specify what type of unit you want to create in it: a package, a subprogram, a generic package, or a generic subprogram.

To create a new LU:

1. **Switch to LU mode.**

2. **Choose New Lib Unit from the File menu to open the New Library Unit window.**



3. **In the Name text field, type the name of the new LU.**

4. **Click SELECT on Package or Subprogram in the Type control.**
   To create a generic package unit or a generic subprogram unit, click SELECT on Generic in addition to one of the other values.

5. **Press the Create button.**
   AdaVision then proceeds to create a new LU by starting a compile job.

   With AutoRead on, the LU icon is displayed in the AdaVision display pane as soon as the compile job is completed successfully.

Note – Make sure Auto Update is on when you create a new LU, otherwise the new unit icon is *not* displayed even after a successful compile. “AutoRead Adjustment” on page 2-14 shows you how to turn on Auto Update.

4.8 Compiling

In AdaVision, you can:

- Compile individual or selected groups of units, choosing Compile in the Commands menu in either LU or CU mode
- Compile all of the units dependent on a selected LU, choosing Make under the Commands menu in LU mode
- Compile all of the units in a selected library, choosing Make Library in the Commands menu in either LU or CU mode

4.8.1 Compile Facility

Choosing Compile from the Commands menu, you compile only the unit you select. If you select more than one unit, AdaVision compiles them in the order in which they are displayed in the display pane, which may not be the correct order. When you are working on new units that are likely to change or fail to compile, or when you are sure that a particular unit has no dependents that would become obsolete if you changed it, then you will probably want to use Compile to compile that unit by itself.

4.8.2 Make or Make Library Compile (Overview)

When you have a number of units to compile, it is generally better to use the `make` facility. This facility first determines the correct order in which to compile the units on which the selected LU depends—or, in the case of Make Library, all the units in the library—and then compiles those which need recompiling.

While Make Library is a powerful utility that saves you from having to figure out exactly which units to recompile and in what order, it can take a long time to run if the library is large.

Using `make` to rebuild only the units on which a selected LU depends is a fast way to compile a set of units, which typically form a distinct module in a larger program.

This section describes how to compile individual units and how to set compiler options. See “Using Make To Compile Library Units” on page 4-28 for coverage of `make` and Make Library.

4.8.3 Compilation of Selected Units

You can compile units in either LU or CU mode. If you work from LU mode, you must use the submenu to specify which kind of unit you want to compile: spec, body (if any), or subunit(s) (if any). You may want to check or set compiler options before compiling a unit. See “Compiler Output Option Settings” on page 4-25.

To compile a spec, body, or subunit in CU mode:

1. **Click SELECT on the object.**
2. **Choose Compile from the Commands menu.**

AdaVision runs each compilation process in the background. It creates a separate Compile Status window.

4.8.4 Job Status Windows

Each job in SPARCworks/Ada runs in its own job status window. You can direct the job to run locally or remotely on an Ada host by changing the setting on the Jobs control in the AdaVision Properties window and specifying the Ada host name. (See “Remote Execution Use” on page 2-16.)

Six different types of jobs run in job status windows:

- Import
- Make (and Make Library)
- Compile
- Link
- Tags
- Delete (LU or CU mode only)

Compile job running

Note – To execute a delete job for an Ada object in a job status window while in CU or LU mode, you must first set the Verbose control in the Delete Properties window to Echo, and press Apply.

The job status windows for each of these operations behave in basically the same way. When a job status window first appears on the workspace, it is closed to its icon. The icon tells the type of job and the name of the target unit, object, or library.

Compile job completed successfully

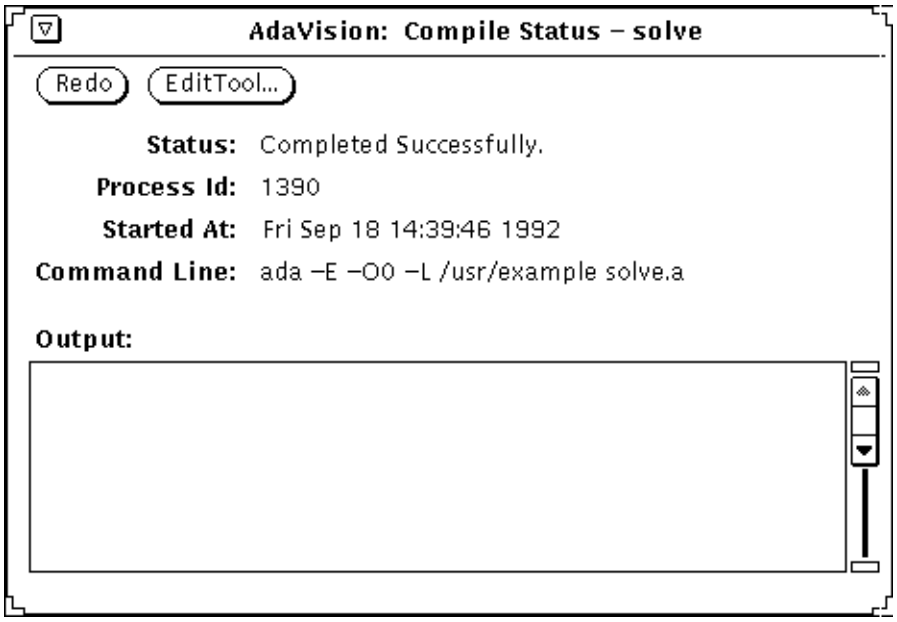
The icon shows a gray background when the job is running. If a job completes successfully, the background becomes clear. If the job terminates unsuccessfully, the icon is broken.

To open a job status window:

- **Double-click SELECT on its icon.**
  Alternatively, place the cursor on the icon, then press MENU and choose Open from the Window menu.

Compile job terminated - unsuccessful compile

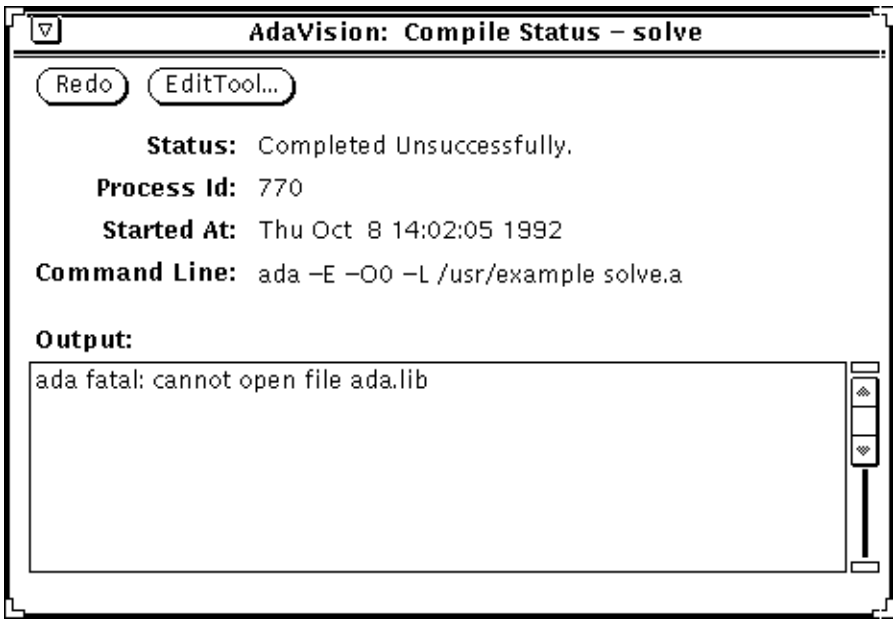
Here is a Compile Status window which shows that the compile job has been completed successfully.



All job status windows report status, the job process ID number, when the job started, and the Ada command line used to perform the job.

The lower portion of each job status window contains a job output pane. This read-only, scrollable pane displays whatever information you asked the command to send or print to standard output, for example, a list of file-to-file dependencies for a `make` process. It also prints any error messages the command reports.

The following diagram shows a Compile Status window for a failed compilation.



4.8.5 Error Correction and Job Resubmission

The Compile Status window contains an EditTool button and a Redo button. These buttons allow you to fix errors in the code that prevented successful compilation and then resubmit the job.

The Jobs control setting in the AdaVision Properties window that you had when you first started a job, whether Local or Remote, does not change when you press Redo; AdaVision will continue to run on the same host. Even if you change that setting before resubmitting the job, the same machine is used on a redo.

The EditTool button launches one of the SPARCworks/Ada programming support tools, EditTool. EditTool loads the source file containing the unit that failed to compile into a specialized version of the OpenWindows Text Editor. EditTool displays source code in the upper part of the display pane and the compiler error message in the lower part; it positions the cursor in the source file at or about the line reported by the error message. Chapter 8, “Using EditTool,” describes the tool in detail.

4.9 Setting Compiler Options

Before you compile, you may want to check or set compiler output options or the options set for individual units. You may also want to check dependencies among units to see which of them you have to recompile as a result of changing some unit, as well as check the order in which to recompile the obsolete dependent units.

The next three subsections show you how to perform these precompilation checks.

4.9.1 Compiler Output Option Settings

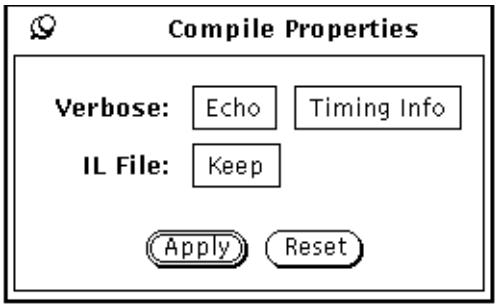
At times, you may want to track compiler activity more closely. You can get more information by setting the Verbose and/or Timing compiler output options. The controls for these options are on the Compile Properties window.

Note – Output options set from the Compile Properties window apply across all units every time you choose Compile; that is, they are compilation-specific rather than unit-specific options. (Don't confuse this Compile Properties window, accessed from Props ⇒ Compile, with the Compilation Unit Properties window, accessed by choosing Selection from the Props menu in CU mode.)

You set unit-specific compilation options from the Compilation Unit Properties window (discussed in "Unit-Specific Option Settings" on page 4-27) or from the Import Properties window (discussed in "Import Properties" on page 4-17).

To set the Verbose or Timing compiler output options:

1. **Choose Compile from the Props menu in LU or CU mode to open the Compile Properties window.**



2. **Set the values and press Apply.**
   If you set the IL File control to Keep, AdaVision keeps the Intermediate Language file when you compile the selected units.

4.9.2 Default Option Settings

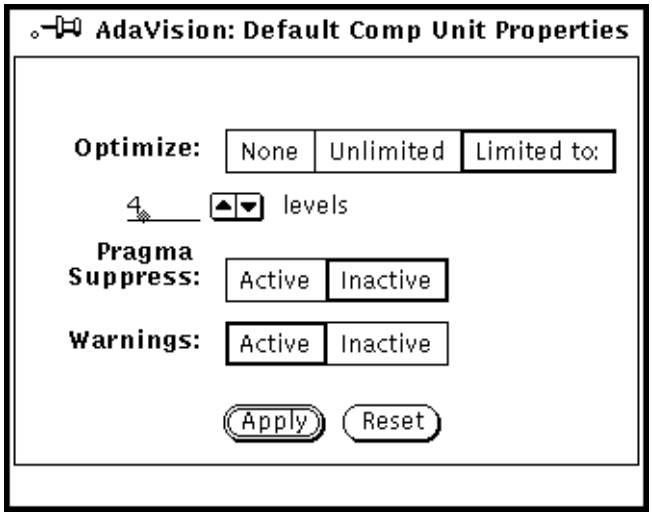
You set default compiler options for all the CUs in a library from the Default Comp Unit Properties window. These options are used for all newly created CUs that have never been compiled, for importing units (unless overridden on the Import Properties window), and for all CUs whose property windows have never been changed (see the next section).

The default options are:

- Optimize
- Pragma Suppress
- Warnings

To check or set the default compiler options:

1. **Switch to CU mode.**
2. **Choose Defaults from the Props menu to open the Default Comp Unit Properties window.**



3. **Click SELECT on the setting you want in each of the three controls.**
   If you set the Optimize control to Limited to, the default number of levels is four. Specify the number you want the compiler to optimize by clicking SELECT on the Up or Down scrolling button located to the left of `levels`.

4. **Apply the settings.**

4.9.3 Unit-Specific Option Settings

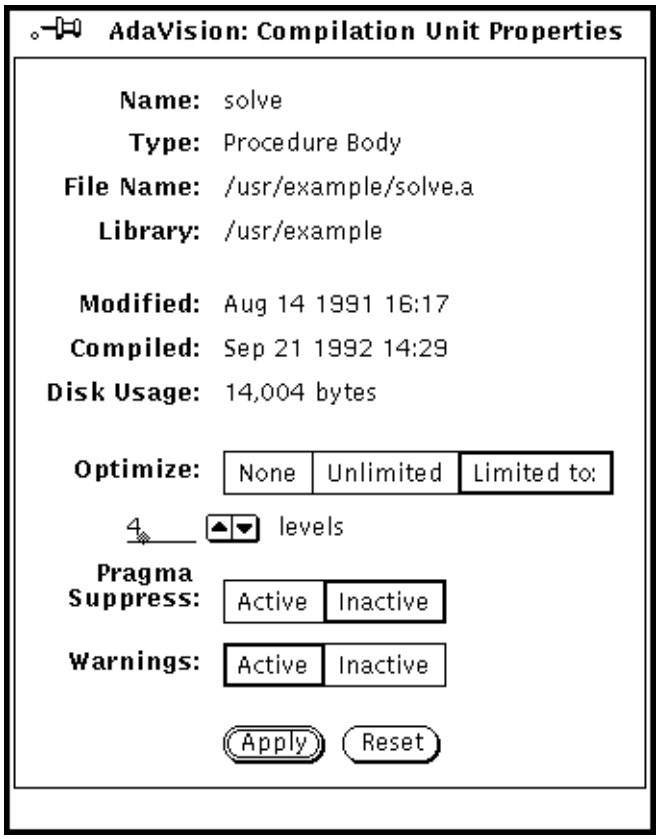
You set compiler options for individual units from the Compilation Unit Properties window. (This is the same window as the one shown in "CU Properties" on page 4-7.) These settings only apply to the specific CU(s) that you select and would override the defaults in the Default Comp Unit Properties window as described in the previous section.

To check or set compiler options for selected CUs:

1. **Switch to CU mode.**

2. **Select the unit(s).**

3. **Choose Selection from the Props menu to open the Compilation Unit Properties window.**
   AdaVision opens a separate property window for each CU selected.



4. **Click SELECT on the setting you want in each of the controls.**

5. **Apply the settings.**

4.10 Using Make To Compile Library Units

Make examines the dependencies for a selected LU, determines the correct order in which to compile units, and then compiles them if necessary. Make Library does the same for *all* instead of just one of the units in the library.

You run Make from LU mode and Make Library from either LU or CU mode. Note that if you do not select any LU in LU mode, the Commands menu item changes from Make to Make Library.

4.10.1 Steps for Running Make or Make Library

To use Make to compile a selected LU and all of the units on which it depends:

1. **Switch to LU.**
2. **Select the LU object(s) you want to `make`.**
3. **Choose Make from the Commands menu.**

In CU mode, Make becomes Make Library. Make Library compiles all of the units in the selected library.

To use Make Library:

1. **Switch to LU or CU mode.**
2. **Choose Make Library from the Commands menu. (This item is available in LU mode only if no LU is selected.)**

Each Make or Make Library job runs as a separate process in the background. AdaVision displays a Make Status window for each Make job. See “Compilation of Selected Units” on page 4-21 for more information about job status windows.

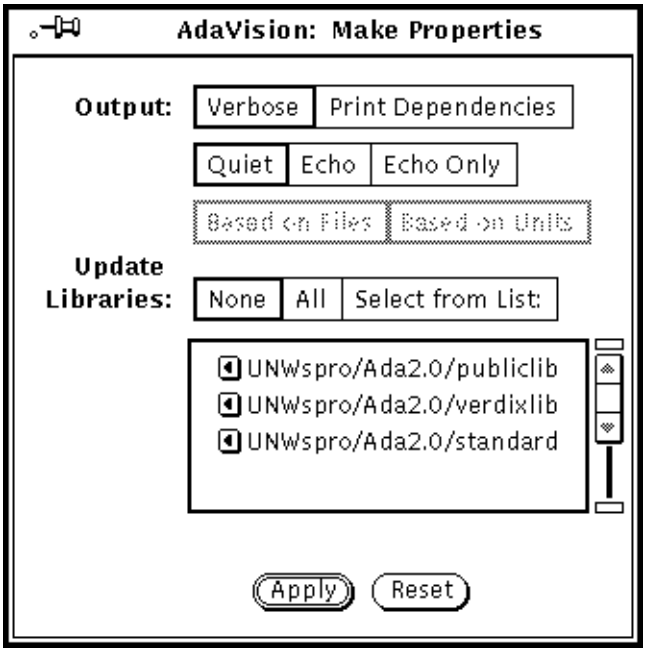
The Make Status window also has a Redo button. Use this button to execute Make quickly after fixing any problem with the previous `make`.

4.10.2 Make Option Settings

The Make facility has a Make Properties window with controls for the three options to Make: Verbose, Print Dependent, and Update Libraries.

To check or set Make facility options:

1. **Choose Make from the Props menu in either LU or CU mode to open the Make Properties window.**



2. Click SELECT on the setting you want for each of the controls and press Apply.

4.10.2.1 Output Options

Notice that the Verbose option and the Print Dependencies option are *mutually exclusive.* By default, the Verbose option is set to Quiet. The Print Dependencies values are inactive (dimmed). When you choose to set a Print Dependencies option, the Verbose option values are dimmed, and the Print Dependencies values become active.

The Verbose option prints or “echoes” to the output pane each of the recompile commands as Ada processes them. The default setting is Quiet, which instructs Make *not* to echo the unit-by-unit commands. When you set Echo, Make recompiles the units *and* echoes the commands to the Output pane. When you set Echo Only, Make echoes the recompilation commands it would process, but does *not* actually recompile any units.

The Print Dependencies option prints either unit or file-to-file dependencies. The default is Based on Files.

4.10.2.2 Update Libraries Option

The Update Libraries option tells `make` to update units, that is, check dependencies and recompile out-of-date units that are in libraries higher up in the selected unit's ADAPATH. The default is None. Make does not recompile units in other libraries unless you set this option.

If you set All, `make` updates dependencies for the selected LU in every library on the ADAPATH of the LU. For large Ada libraries, this could take some time.

If you set Select from List, you can then select from the scrolling list those libraries on the ADAPATH for the selected library that you want AdaVision to update.

4.11 Linking Program Units

The link job in AdaVision runs as a separate process in the background. AdaVision displays a job status window for Link, called the Link Status window. Like all job status windows, it is initially closed to its icon. (See "Compilation of Selected Units" on page 4-21 for more information about job status windows.)

4.11.1 Steps for Linking Units

Executable main program

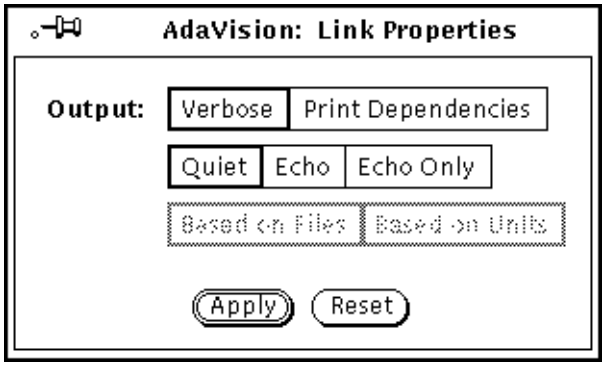
To link the units in a main program:

1. **Switch to LU mode.**
2. **Click SELECT on the subprogram spec of the unit that is to become the main program.**
3. **Choose Link from the Commands menu.**

When AdaVision successfully links a program, it places a Sun logo in the main program.

4.11.2 Link Option Settings

Link also has a Link Properties window which is displayed when you choose Link from the Props menu in either of the unit modes.



You can set the Verbose or Print Dependencies options. Be sure to press Apply after choosing your settings. These options behave in the same way as they do for the Make facility. (See “Make Option Settings” on page 4-29.)

4.12 Running a Program

You can run a program from AdaVision in one of two ways:

- By selecting the main program icon and choosing Run in the Commands menu or the floating menu
- By selecting, dragging, and dropping the main program icon onto the workspace

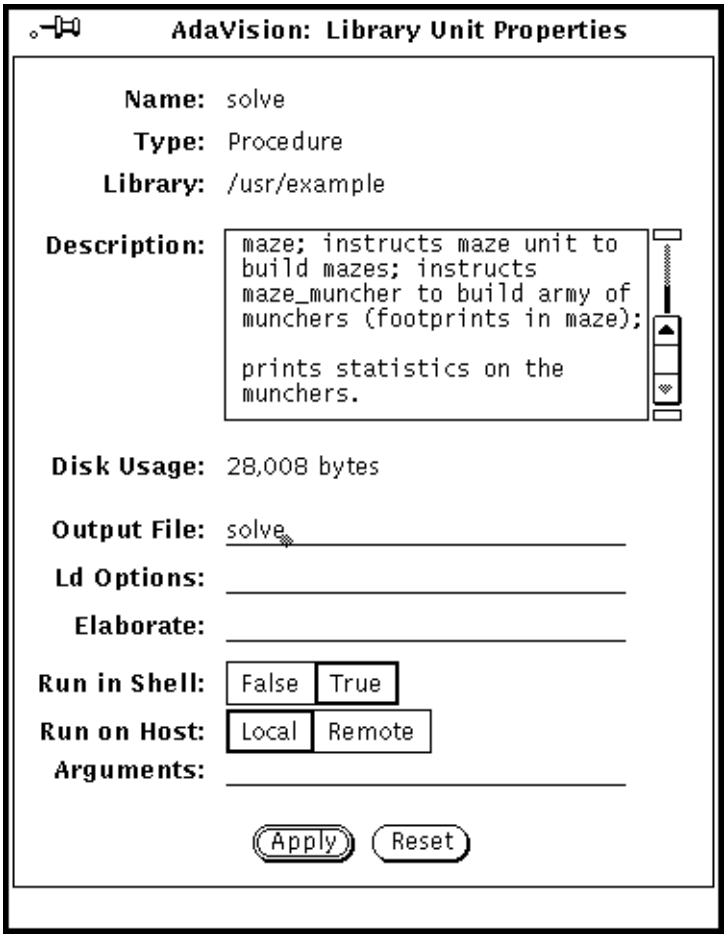
AdaVision marks main program icons, that is, program executables, with a Sun logo, as shown in the previous section.

4.12.1 Main Program Properties

Before learning how to run a program, look at the Library Unit Properties window for the main program.

To open the Library Unit Properties window:

1. **Switch to LU mode.**
2. **Click SELECT on the main program icon.**
3. **Choose Selection from the Props menu to open the Library Unit Properties window.**



Note the choices you must make for the Run in Shell and Run on Host controls. Whether you need to run the program in a Shell Tool depends on whether the program makes use of standard input (`stdin`), standard output

(`stdout`), or standard error (`stderr`). If the program makes use of any of these features, it should run in a Shell Tool. The default for this control is True.

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

At the Run on Host control, you check Local or Remote to indicate whether you want to run the program on your local machine (*swada_host*) or the remote machine (*ada_host*). Local is the default. Remote applies only when you have selected Remote for the Jobs control in the AdaVision Library Properties window. (Refer to "Remote Execution Use" on page 2-16.) The additional setup requirements for remote execution are covered in the next section.

You can also enter arguments to be passed to your program in the Arguments text field.

4. **Press Apply.**

4.12.2 Program Execution from AdaVision

If your main program is an OpenWindows application, and you would like to use remote execution to run it, you must first set your environment for remote display of your main program windows. Use the following commands at the system prompt of a Command or Shell Tool before starting AdaVision, where the variables, *ada_host*, *swada_host* and *display_machine,* are the machine names.

```
swada_host% xhost ada_host
swada_host% setenv DISPLAY display_machine:0
```

Note that *swada_host* and *display_machine* might be the same machine.

After you start AdaVision, remember to set the AdaVision Jobs control setting to Remote and enter the *ada_host* on the AdaVision Properties window. "Remote Execution Use" on page 2-16 shows you how to do so.

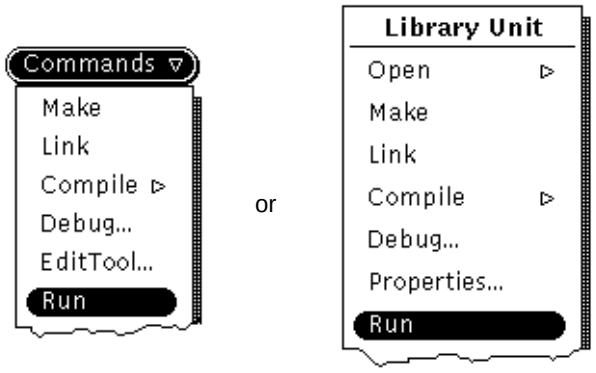
4.12.2.1 How to Run

Run a program in one of three ways.

Choosing Run from the Commands Menu

Here are the steps:

1. **Switch to LU mode.**
2. **Click SELECT on the main program, that is, the program executable.**
3. **Choose Run from the Commands menu or the floating menu.**



Using Drag-and-Drop

Follow steps 1 and 2 above, then drag and drop the main program icon onto the workspace.

Using AdaDebug

"Program Execution" on page 5-15 in Chapter 5, "Using AdaDebug" provides the details.

4.12.3 Program Execution in AdaDebug from AdaVision

To run the program in AdaDebug, the Ada visual debugger, click SELECT on the icon for the main program in LU mode and then choose Debug from the AdaDebug Commands menu, or drag and drop the program icon onto the AdaDebug display pane.

Chapter 5, “Using AdaDebug,” and Chapter 6, “Debugging Tasks with AdaDebug,” describe AdaDebug in detail, including how to set up for remote execution and remote display.

4.13 Sorting and Filtering Units

If a library contains many units, you may want to sort and/or filter them. The default sort order is alphabetical by *name*.

In CU mode, you can sort the units in one of three ways:

- Alphabetically by unit *name*
- Alphabetically by unit *type*—tasks, generic subprogram, subprogram, generic package, package
- Alphabetically by unit *kind*—specification unit (spec), secondary unit (body), or subunit.

In LU mode, the objects are not individual units, but composite objects representing specs, bodies, and subunits. You can sort them by *name* or *type*, but not by *kind*.

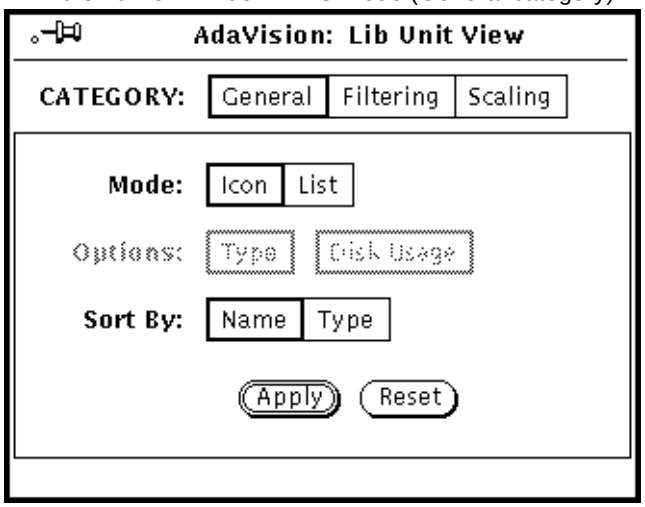
The following subsections describe how to sort and filter icons. AdaVision saves the settings that you change in the property windows for these facilities. When you restart AdaVision, these settings remain the same as that for the previous session, that is, the icons are sorted and filtered in exactly the same manner as before.

4.13.1 Object Sorting in LU or CU Mode

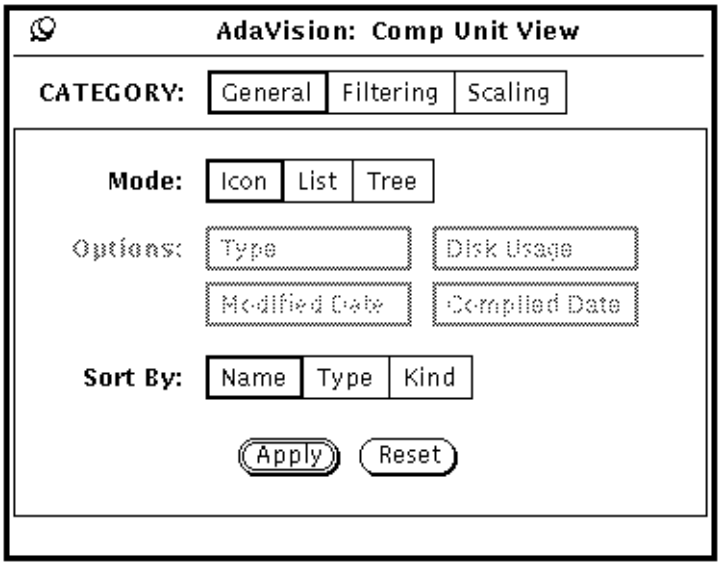
To sort unit objects in either LU or CU mode:

1. **Choose Options from the View menu to open either the Lib Unit View window or the Comp Unit View window.**
   Be sure General is selected under CATEGORY.

Lib Unit View window in LU mode (General category)



Comp Unit View window in CU mode (General category)



2. **Click SELECT on the desired value in the Sort By control and press Apply.**

4.13.2 Type Sorting in CU Mode

When you sort CUs by *type*, AdaVision displays units:

- First, *by type*, in the following order:
  - Tasks
  - Generic subprograms
  - Subprograms
  - Generic packages
  - Packages
- Second, within each type listed, *alphabetically by name*
- Third, within each group of alphabetically sorted units, *by kind*:
  - Specs
  - Bodies
  - Subunits

Figure 4-1 summarizes the sort-by-type sort order. It also serves as a matrix display of the main types of AdaVision CU icons.

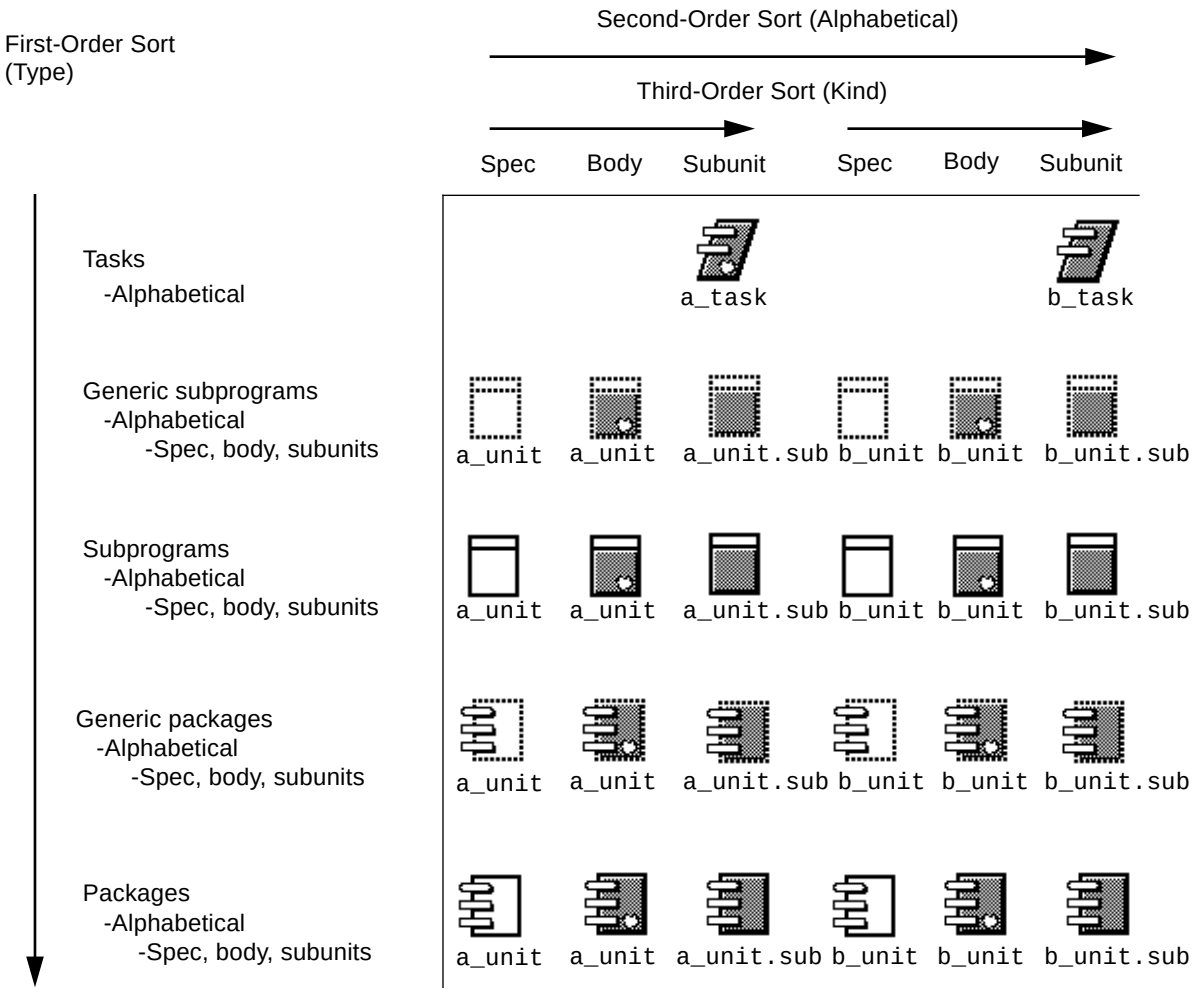
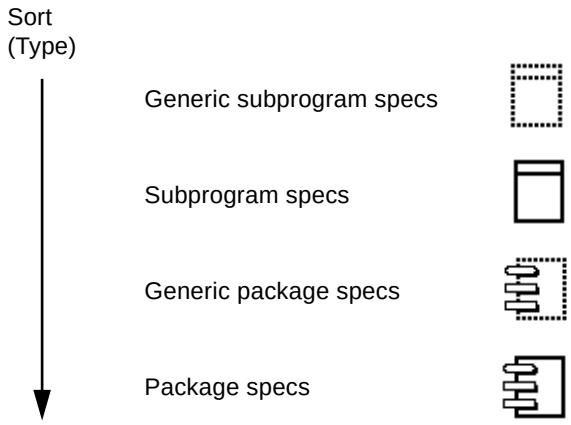


Figure 4-1 Sort order for compilation units by type in CU mode

4.13.3 Type Sorting in LU Mode

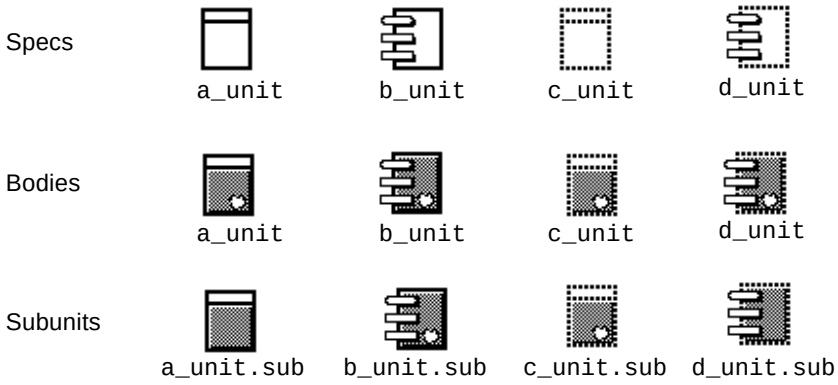
You sort objects by type only in LU mode. The sorting order is the same as in CU mode, first by type, then alphabetically by name. The pattern is less complicated because there is no third-order sorting by kind.



4.13.4 Kind Sorting in CU Mode

When you sort by *kind* in CU mode, AdaVision lists:

1. Spec units of all types, alphabetically by name
2. Then body units of all types, alphabetically by name
3. Then subunits of all types, alphabetically by name



4.13.5 Object Filtering

Filtering is another way to sift through large numbers of units in an Ada library. In CU mode, you can filter objects by unit type or kind. In LU mode, you can filter objects by type only. You can also specify a pattern which AdaVision uses to display units matching the pattern, that is, it filters out all units except those with the pattern in their names.

The filtering controls, Show and Pattern, are in the Filtering subwindow under CATEGORY on the Options window that is invoked from the View menu in either LU or CU mode. See the diagram under “Object Sorting in LU or CU Mode” on page 4-36.

4.13.5.1 Filtering Subwindow

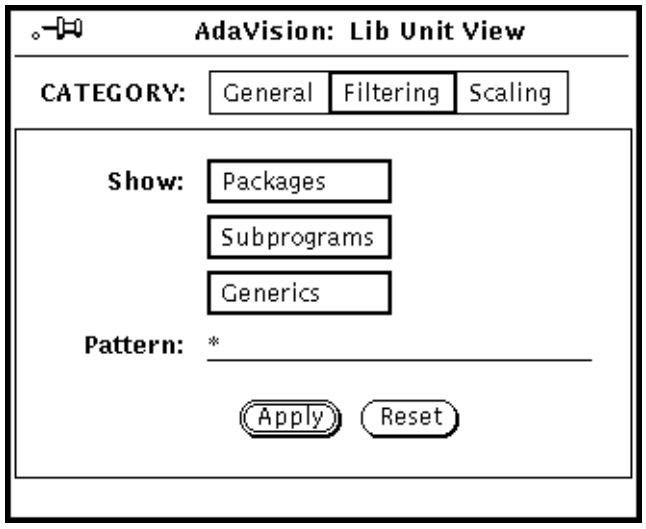
To filter objects with the Show control in either LU or CU mode:

1. **Choose Options from the View menu to open the Lib Unit View window or the Comp Unit View window, depending on which mode AdaVision is in.**
2. **Click SELECT on Filtering under CATEGORY.**
   The Filtering subwindow is then displayed.

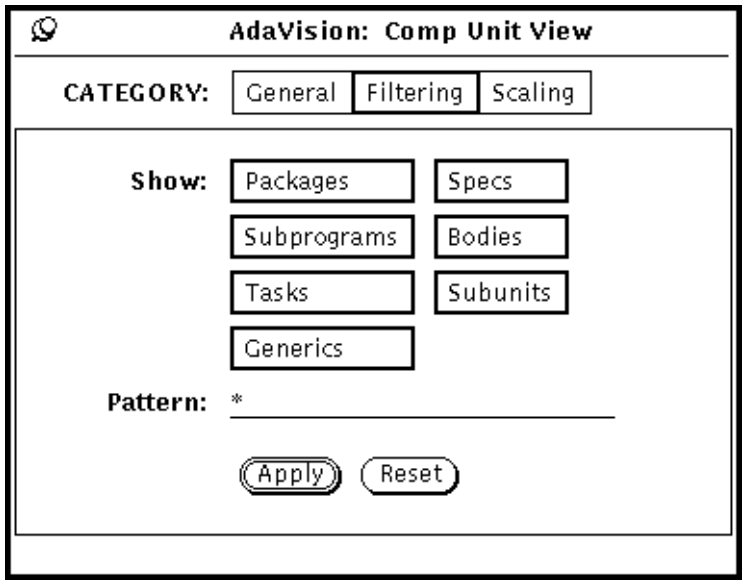
Here is what that window looks like in both LU and CU modes.

The Filtering subwindow

LU mode



CU mode



3. **Click SELECT on the settings for the types and kinds of units you want displayed.**
   By default, all settings are selected, so you see all the units in the library when you first switch to either LU or CU mode. If a value is set, its frame is thick and dark. You can set any or all of the values independently of one another. Each setting is a toggle switch.

4. **Apply the settings.**

4.13.5.2 Show Control

The Show control in LU mode is the same as the control in CU mode. The only exception is:

- The LU mode control does not have values for specs, bodies, or subunits (because you cannot filter by *kind* in LU mode).

4.13.5.3 Pattern Control

By using the Pattern control, you can restrict the display objects to those whose names match the pattern specified. AdaVision filters out of the display all objects *except* the ones matching the pattern.

Note that Pattern operates in conjunction with the Show control filters. Pattern further restricts whatever filters are set in the Show control. For example, if you filter the display to show body units only, you can further restrict the display to body units whose names match a specified pattern.

To filter library objects with the Pattern control in either LU or CU mode:

1. **In the Pattern text field in the Filtering subwindow, type a string that matches a pattern in the names of objects you want AdaVision to display.**
   Initially, the Pattern text field displays an asterisk (*), the wildcard character. This control accepts any regular expression. For example:

2. **Apply the setting.**

4.14 Printing Files and Screens

AdaVision supports printing of source files for selected specs, bodies, or subunits. You can also print AdaVision screens, that is, the contents of the AdaVision display pane.

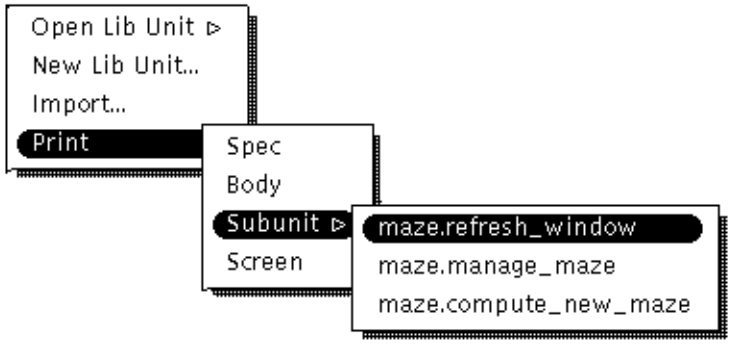
AdaVision checks the `PRINTER` environment variable to set the initial value of the Print Script control. You can change this script to have AdaVision use any other printer your workstation supports. (See “Printer Name Assignment” on page 2-13 on how to change the Print Script control.)

4.14.1 Spec, Body, Subunit, or Screen Printing

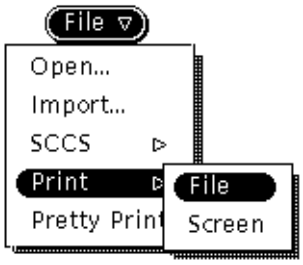
To print a spec, body, or subunit of an LU:

1. **Switch to LU mode.**
2. **Click SELECT on the LU object containing the units you want printed.**
3. **Pull to the right on the Print item on the File menu to open the Print submenu.**
4. **Select Spec, Body, Subunit, or Screen as you want.**
   Spec is the default choice.

   You can print a subunit only if there are one or more subunits in the LU you selected, in which case you can pull to the right to highlight the subunit you select. If not, this item is inactive and dimmed.



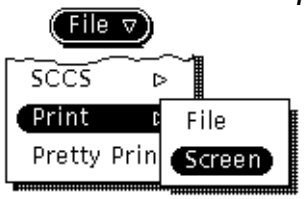
4.14.2 File Printing



To print source files for selected CUs:

1. **Switch to CU mode.**
2. **Select the unit(s) you want printed.**
3. **Pull to the right on Print from the File menu to open the Print submenu.**
4. **Choose File.**

4.14.3 Display Pane Content Printing



To print the contents of an AdaVision display pane in LU or CU mode:

1. **Display all the objects you want printed. Resize the AdaVision main window if necessary.**
2. **From the File menu, choose Screen from the Print submenu.**

"Printer Scaling" on page 2-24 explains how to scale AdaVision icons for printing.

Note that the Tree graph window and the Dependencies graph window have their own Print buttons. See "Tree Graph of Units in an LU" on page 4-13 and "Dependencies Graph of Units in an LU" on page 4-14.

4.15 Using Tags from the Commands Menu

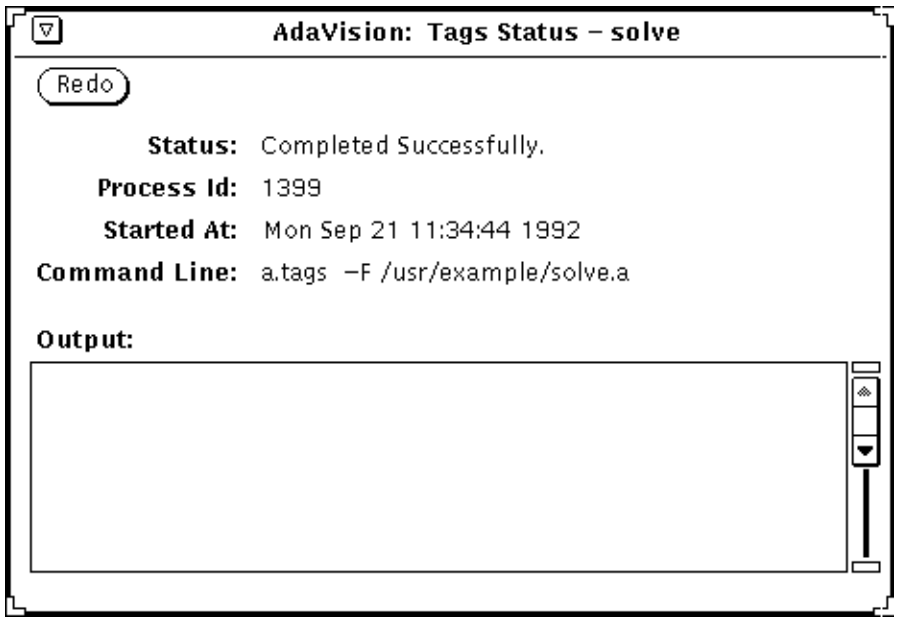
While in CU mode, you can select a CU and execute the Ada `a.tags` command by choosing Tags from the Commands menu. This process creates a `tags` file for that unit in the directory in which the Ada library resides. (For details on the `a.tags` command and the `tags` file, refer to the SPARCompiler Ada documentation.)

To create a `tags` file:

1. **Switch to CU mode.**
2. **Click SELECT on a CU.**
3. **Choose Tags from the Commands menu.**

The Tags job runs in a Tags Status window, which is initially closed to an icon. If the job completes successfully, the icon background clears. If the job is unsuccessful, the icon appears broken.

Double-click SELECT to open the icon; you will see a window displaying the status, the process ID, the date and time the Tags job started at, as well as the Ada command line used for the job. An output pane shows error messages, if any.



4.16 Using Pretty Print

AdaVision supports the reformatting of Ada code. This feature is called “pretty printing.”

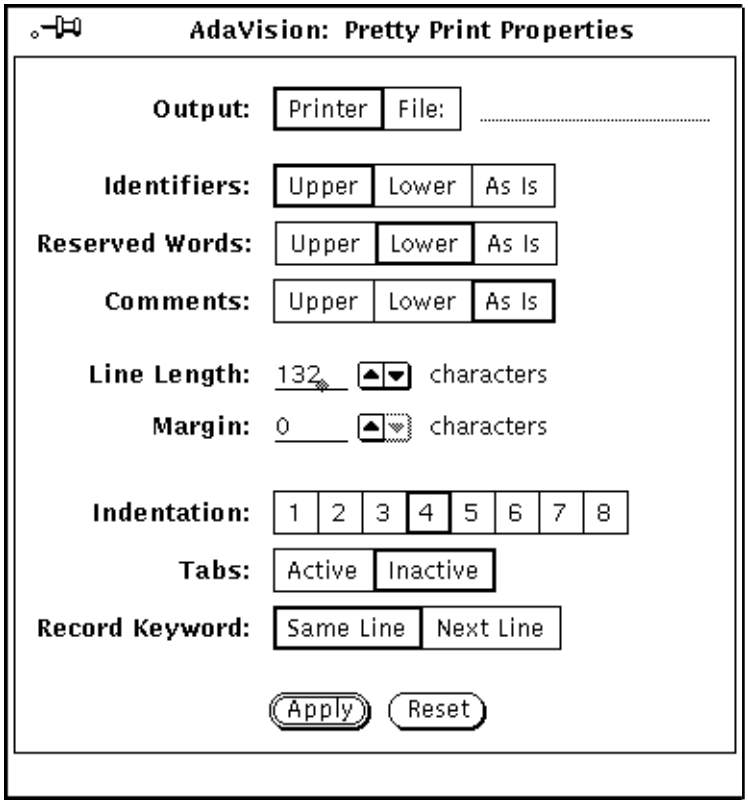
4.16.1 Pretty Print Properties

The Pretty Print facility offers a property window from which you can customize the options for the pretty print output.

To check or change Pretty Print options:

1. **Switch to CU mode.**

2. **Choose Pretty Print on the Props menu to open the Pretty Print Properties window.**



Here is a brief description of the settings.

At the Output control, you choose to send the reformatted code to a separate file or send it directly to the printer. If you choose File, be sure to type in the full path name in the text field provided.

Next, you can specify whether to print occurrences of the following Ada language structures in uppercase (Upper), lowercase (Lower), or leave them as they are in the original source code (As Is):

- Identifiers
- Reserved Words
- Comments

Line Length is set to 132 by default, the maximum number of characters of source code in each line. You can lower or raise this number by editing it, or by clicking SELECT on the Up or Down scrolling button.

The Margin setting specifies the left indent in characters for the topmost level in the code. The default is 0, that is, flush left.

Indentation sets the number of spaces to indent for each level in the code. The default is 4.

If you want to substitute a tab character for each sequence of eight consecutive space characters, select Active in the Tabs field.

Record Keyword controls the placement of the Ada keyword `record`, placing it either on the same line as the reserved words `type` or `for`, or on the next line.

3. **Click SELECT on or type in the value for the settings you want in each control.**

4. **Press Apply.**

4.16.2 Steps for Pretty Printing

To reformat Ada code using pretty printing:

1. **Switch to CU mode.**

2. **Click SELECT on the unit containing the source to be reformatted.**

3. **Choose Pretty Print on the File menu.**

Using AdaDebug 5

This chapter is the first of two chapters that describe how to use SPARCworks/Ada AdaDebug. It explains how to debug single-tasking programs from within the AdaDebug Program View window. Chapter 6, “Debugging Tasks with AdaDebug,” describes the Task Browser and how to debug specific tasks in Task View windows.

This chapter is organized into the following sections:

5.1 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 OPEN LOOK GUI features and facilities while preserving the full range of the `a.db` functionality.

5

In AdaDebug, you browse and debug single-tasking programs in a *Program View* window. For multitasking programs, AdaDebug also provides a *Task Browser* and multiple *Task View* windows. (Chapter 6, “Debugging Tasks with AdaDebug,” details the Task Browser.) In both Program View and Task View windows, a Commands menu contains the most commonly used debugging task items. You can also issue `a.db` commands from a command line located to the right of the Custom menu button.

AdaDebug supports a separate Program I/O window and a Messages Log window.

AdaDebug opens into a Program View window. Program View shows you the program from the top-down perspective of its main subprogram.

The Program View window is an OPEN LOOK base window, so you can resize it, move it, or close it to an icon. You can leave it open on the workspace for as long as you like. You can also start more than one AdaDebug. Each AdaDebug session has its own Program View window.

You can customize AdaDebug properties in a Properties window. This window is the same for all three main AdaDebug windows: Program View, Task Browser, and Task View.

Figure 5-1 shows the AdaDebug Program View window.

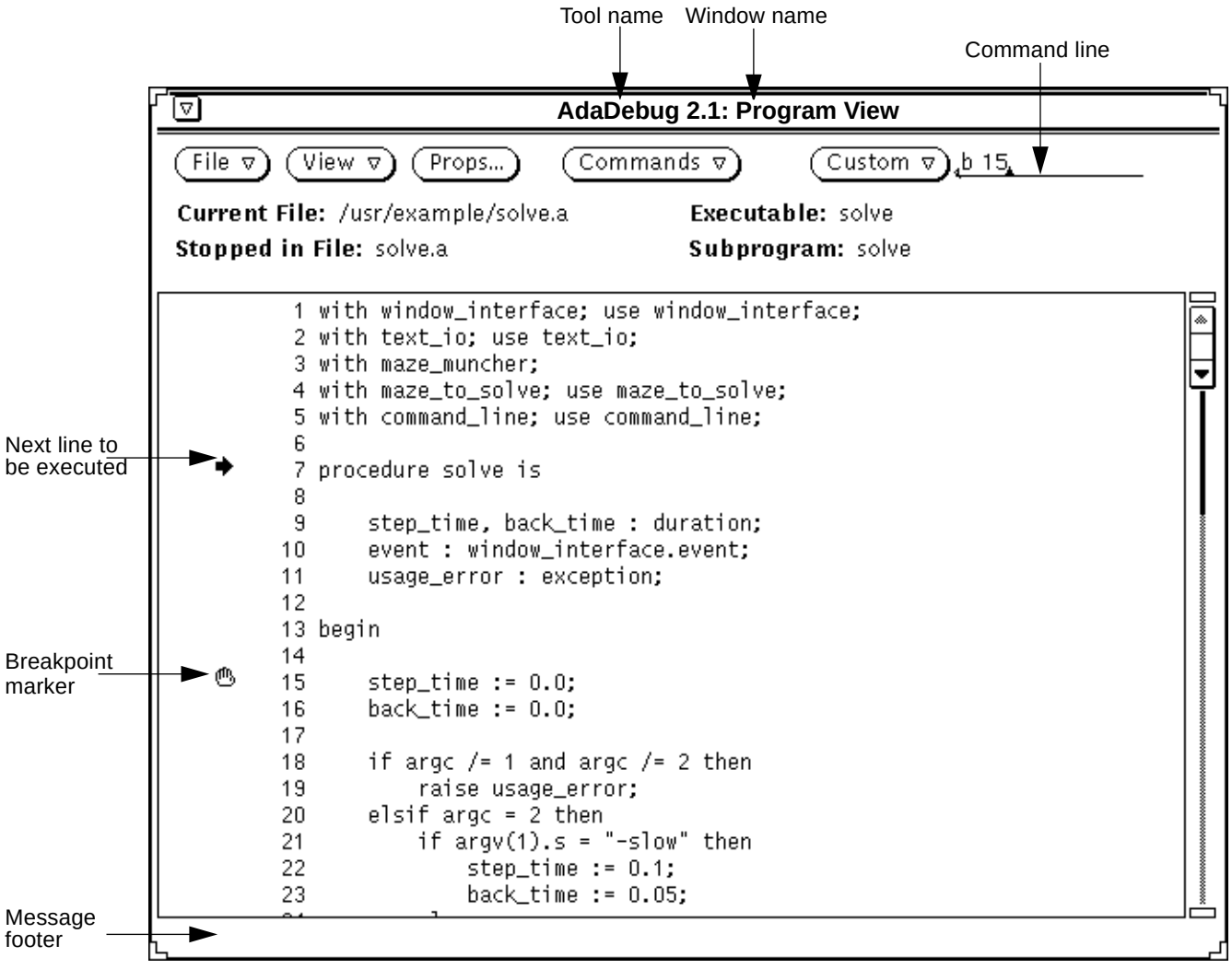


Figure 5-1 Program View window for AdaDebug

Tool name
Name of tool you are using; in this case, AdaDebug.

Window name
Name of window; in this case, Program View.

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

Next line to be executed
Line marked with an arrow, representing the next line to be executed. This point is also known as "Current Focus."

Breakpoint marker
Glyph representing the palm of a hand held up as if to signal "stop," denoting the location of the breakpoint.

Message footer
Area where AdaDebug displays messages. Error or warning messages are printed on the left footer.

Also displayed in the control panel is the following information:

Current file
Name of the file being displayed.

Executable
Name of the executable file.

Stopped in file
Name of the file at which the program stopped after reaching the breakpoint.

Subprogram
Name of the subprogram or unit at which the program is currently stopped.

5.2 Working in AdaDebug

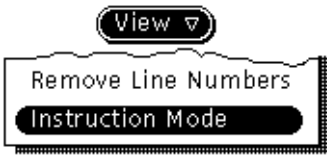
AdaDebug operates in two modes: source mode and instruction mode.

5.2.1 Instruction Mode and Source Mode

The Program View window is always in source mode when you start AdaDebug. In source mode, AdaDebug displays high-level source code. In instruction mode, AdaDebug displays the machine instructions for each statement. Thus, you can debug a program at the level of the machine instructions.

In Program View source or instruction mode, you can:

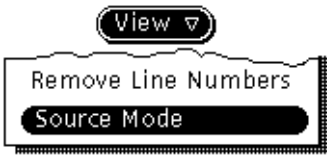
- Issue debugging commands by choosing from the Commands menu (also available as the floating pop-up menu in the display pane)
- Issue Ada `a.db` commands on the command line to the right of the Custom menu button



To switch to instruction mode:

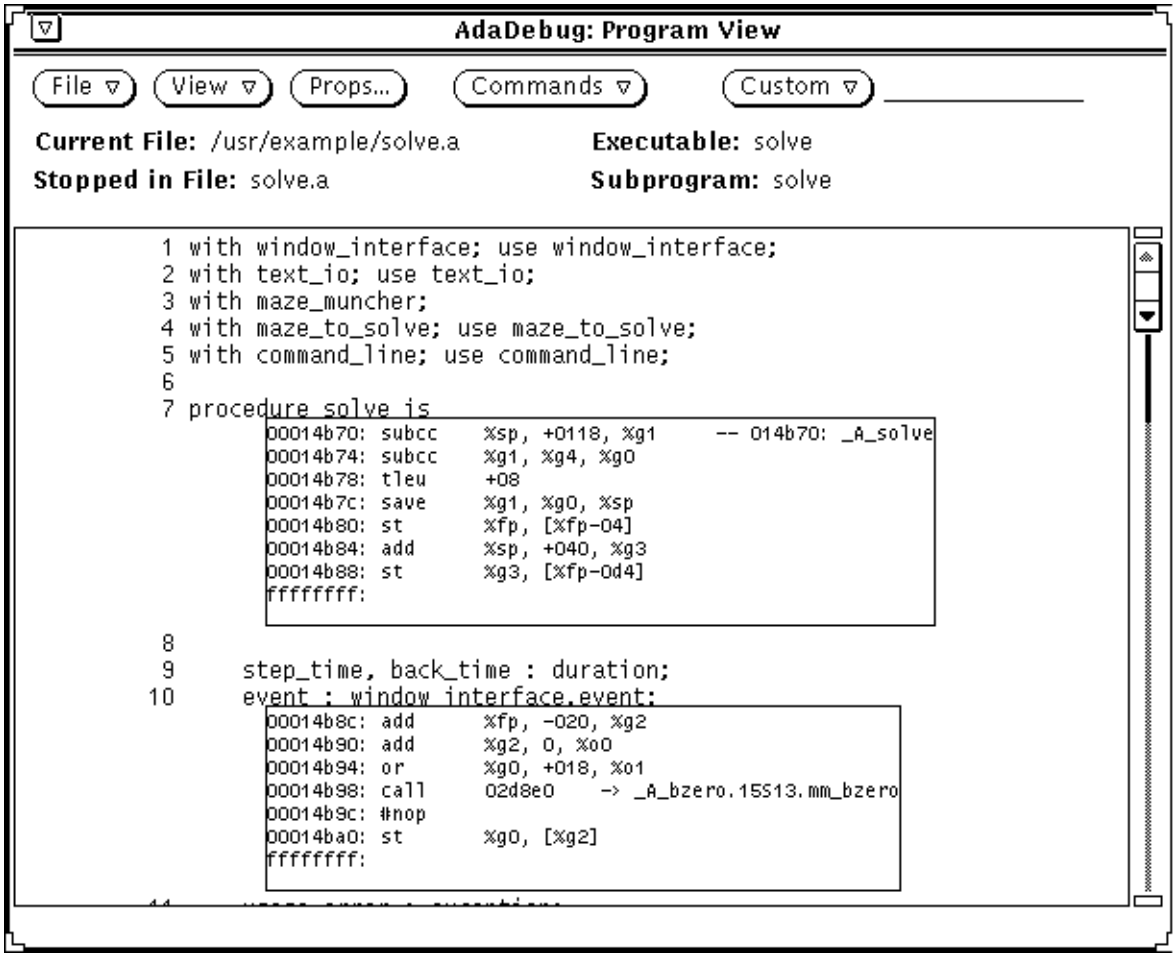
♦ **Choose Instruction Mode from the View menu.**

As soon as instruction mode is in place, the second menu item under View toggles to Source Mode, and vice versa. Therefore, to switch out of instruction mode back to source mode:



♦ **Choose Source Mode from the View menu.**

The following figure shows the Program View window in instruction mode.



5.2.2 Unit and File You Are Viewing

When you issue a debugging command that requires AdaDebug to display code not currently in view, AdaDebug automatically scrolls the display.

If you step into a call that requires AdaDebug to display a line in a different file, AdaDebug replaces the current file with the required file.

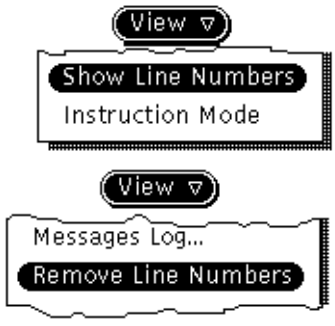
Information fields along the top and bottom of the display pane tell you at which file, subprogram, and line the current line glyph ➧ is pointing.

5.2.3 Command Execution

In AdaDebug, you execute commands by pressing menu buttons or choosing menu items. When they are chosen, some menu items trigger pop-up windows that offer more options. After setting your options in the pop-up window, you then press a [*Command*] button in the window, where *Command* is a variable denoting the command to be executed.

In pop-ups with a single text field and, with the cursor in the pop-up window, pressing the Return key on the keyboard is equivalent to pressing the [*Command*] button. When two or more buttons are present, the default command is executed. The default button is encircled in double rings, as shown in the following example:

5.2.4 Line Numbers



When a program is first loaded onto the Program View window, no line numbers are displayed for the source code.

To show line numbers:

♦ **Choose Show Line Numbers under View.**

As soon as line numbers are shown, this menu item toggles to become Remove Line Numbers. So, to remove line numbers:

♦ **Choose Remove Line Numbers under View.**

5.3 Starting AdaDebug

You can start AdaDebug in three ways. In the following sections, remote debugging setup is explained before you are shown how to start AdaDebug.

5.3.1 Remote Debugging of an OpenWindows Application

If the main program you want to debug is an OpenWindows application, and you would like to use remote execution for the debugging process, you must first set your environment for remote display.

To do so, execute the following commands at the system prompt of a Command or Shell Tool on the machine on which you run AdaDebug, *swada_host*. The variable *ada_host* is the name of the machine licensed to run SPARCompiler Ada; *display_machine* is the name of the machine where the main program windows are displayed.

```
swada_host% xhost ada_host
swada_host% setenv DISPLAY display_machine:0
```

Note that *display_machine* may be the same as *swada_host*.

This command sequence allows AdaDebug to run on *swada_host*, `a.db` (the SPARCompiler Ada debugger) and your main program to run on *ada_host*, and your windows to be displayed on *display_machine*.

When you start AdaDebug, be sure to use the `-h` option for host name. See the next section.

Note – Once you have started AdaDebug, you cannot switch back and forth between local and remote execution.

5.3.2 AdaDebug Startup

You can start AdaDebug:

- From a Command or Shell Tool
- From AdaVision
- From the SPARCworks Manager palette

Each of these processes is described in the following subsections.

5.3.2.1 From a Command Tool or Shell Tool

To run AdaDebug from a Command or Shell Tool:

♦ **Type** `adadebug` *[AdaDebug_options] [program_name]* **at the system prompt.**

Aside from its own options, AdaDebug also supports most of the generic OpenWindows options. All other options that you specify at the command line are passed to `a.db`, the underlying Ada debugger. This is an example:

```
% adadebug -Wx large -h ada_host -L path_name program_name
          \_____________/ \_____/
          AdaDebug options  a.db options
```

where:

`-Wx large` is the OpenWindows option that increases the font size
`-h` is an AdaDebug option, followed by the host name variable, *ada_host*
`-L` *path_name* is an option for the underlying debugger
program_name is the name of the main program to debug

Note – The `a.db` option commands are described in the SPARCompiler Ada documentation. The option commands for OpenWindows are described in the `man` page for `xview`. AdaDebug accepts all of the OpenWindows options except for those which change the window size and icon labels, namely, `-Ww`, `-Wh`, `-Ws`, `-WL`, and `-WI`.

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

5.3.2.2 From AdaVision

To start AdaDebug from the AdaVision Commands menu:

AdaVision Commands Menu

- **Select the program executable and choose Debug from the AdaVision library unit (LU) mode Commands menu.**
  AdaDebug loads the program as it opens.

  If you do not select the executable, you can still open AdaDebug by choosing Debugger from the AdaVision Commands menu. AdaDebug is then launched with an empty source display pane. To start a debugging session, you must load a program into AdaDebug.

5.3.2.3 From SPARCworks Manager

To start AdaDebug from the SPARCworks Manager palette:

- **Double-click on the AdaDebug icon in the palette, or drag and drop that icon onto the workspace.**

5.3.3 Program Loading into AdaDebug

Most of the time, you will probably specify a program to debug when you first start AdaDebug. If you do not, you will want to load the program by one of the following methods:

- Dragging and dropping from AdaVision or File Manager
- Starting a debugging session from within Program View

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

5.3.3.1 Dragging and Dropping from AdaVision or File Manager

To load a program from AdaVision:

- **Drag and drop an executable program icon onto the Program View source display pane.**
  AdaDebug replaces the current program, if any, with the new one.

To load a program from File Manager:

- **Drag and drop an executable Ada file icon onto the Program View source display pane.**
  Again, if there is already a program in AdaDebug, the new program replaces it.

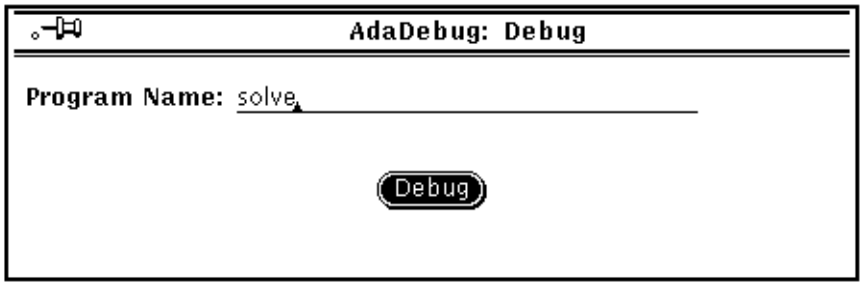
5.3.3.2 Starting a Debugging Session from within Program View

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

1. **Choose Debug on the File menu.**
   The Debug pop-up window is then displayed.

2. **Type the path name of the program you want to debug in the Program Name text field.**

3. **Press the Debug button or the Return key.**
   AdaDebug replaces the current program, if any, with the new one.



The Debug pop-up window is especially convenient when AdaVision or File Manager is not displaying the executable unit or file.

5.4 Browsing a File or Subprogram

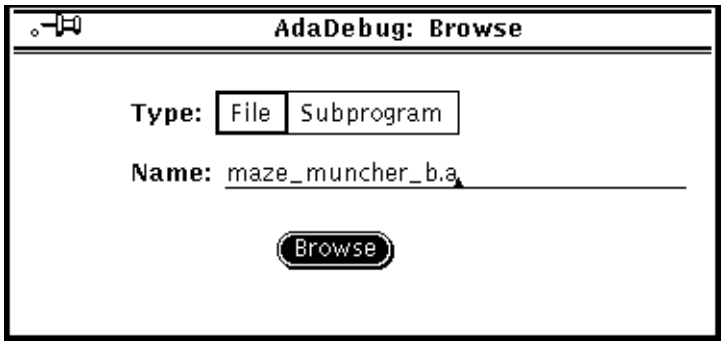
To browse a file or subprogram in the Program View source display pane:

1. **Choose Browse from the File menu.**
   The Browse pop-up window is then displayed.

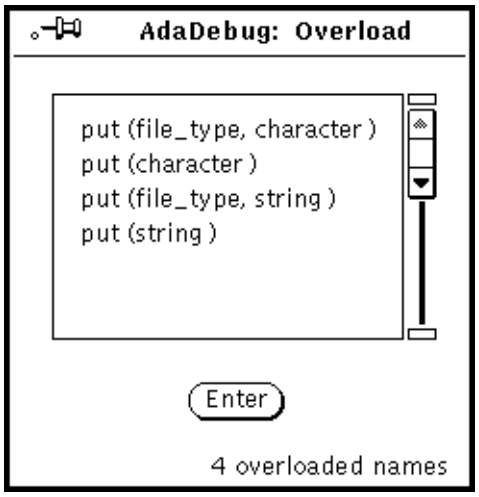
2. **Click SELECT on File or Subprogram to specify the type.**
   File is the default.

3. **Type the name of the file or subprogram you want to browse in the Name text field.**
   AdaDebug replaces the current program, if any, with the new one.

4. **Press the Browse button or the Return key.**
   The source for the file or subprogram is then loaded in the Program View display pane. This display pane is a read-only code-browsing and command entry window; you cannot edit the source code displayed in it.



When you press Browse or Return, if there are two or more subprograms by the same name, AdaDebug pops up the Overload window. Listed in the output pane in that window are the overloaded subprograms and their parameters. Following is an example:



Subsequently, to browse a subprogram:

1. **Click SELECT on the subprogram name.**
2. **Press the Enter button.**
   The source for the subprogram is then loaded in the Program View display pane.

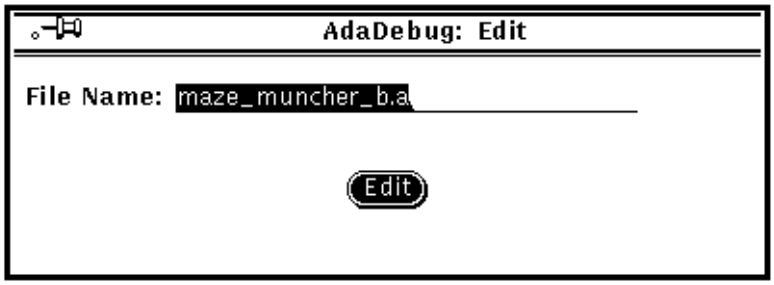
5.5 Editing a File

To edit a file in the Program View source display pane:

1. **Choose Edit from the File menu.**
   The Edit pop-up window is then displayed. The current file name is displayed in the File Name text field as a default entry. For editing convenience, it is highlighted. Change the name as you wish.

2. **Press the Edit button or the Return key.**
   Your editor of choice is then invoked, and the source file is loaded in its display pane for editing. In the text pane of some editors, including that of EditTool, the cursor is positioned at the same line of the source file as that being displayed in the Program View window.

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



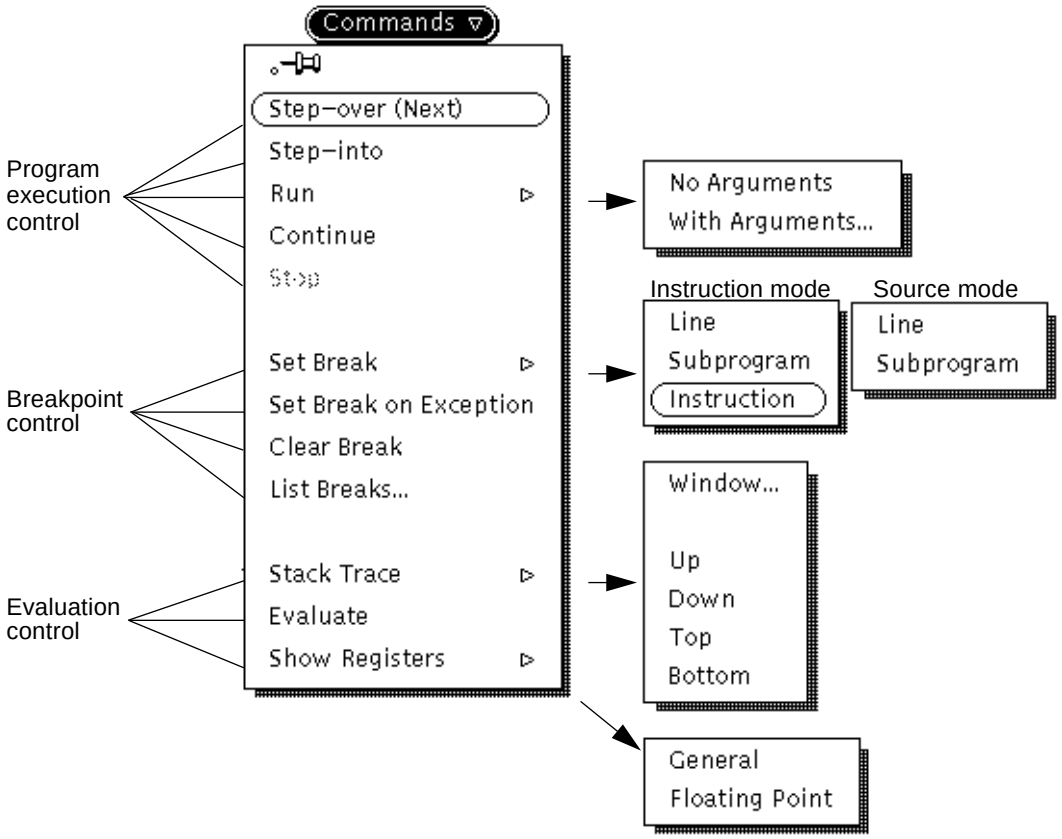
Note – To change your editor of choice, change the option setting for the Editor control in the AdaDebug Properties window. See “Customizing AdaDebug Properties” on page 5-28 for details.

5.6 Using the Commands Menu

The next few sections describe how to use the items on the Program View Commands menu. How to use the command line is described in “Using the Command Line” on page 5-27.

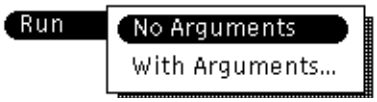
Here is a picture of the Commands menu. The four submenus are “opened” to show you where each of these basic debugging menu items is located. In actual use, you can have only one submenu open at a time.

Note – Certain items are dimmed when they do not apply. For example, the last item on the menu, Show Registers, is active only in instruction mode, dimmed in source mode.



5.6.1 Program Execution

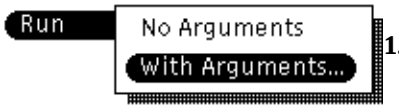
To run a program in AdaDebug:



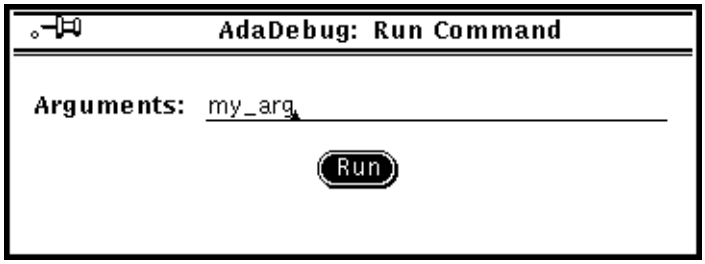
- **Choose Run from the Commands Menu.**
  No Arguments is the default choice on the submenu.

5.6.1.1 Program Execution with Arguments

To run a program with arguments:



1. **Choose With Arguments from the Run submenu to open the Run Command pop-up window.**
2. **Type the arguments in the Arguments text field.**
3. **Press the Run button or the Return key.**



5.6.1.2 Program Continuation

After a program has stopped at a breakpoint, choose Continue on the Commands menu to continue program execution from the location of the current breakpoint.

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

5.6.1.3 Program Stop

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

This menu item is useful in special circumstances—for example, when you want to stop the program to break out of an infinite loop without killing the program and the debugger.

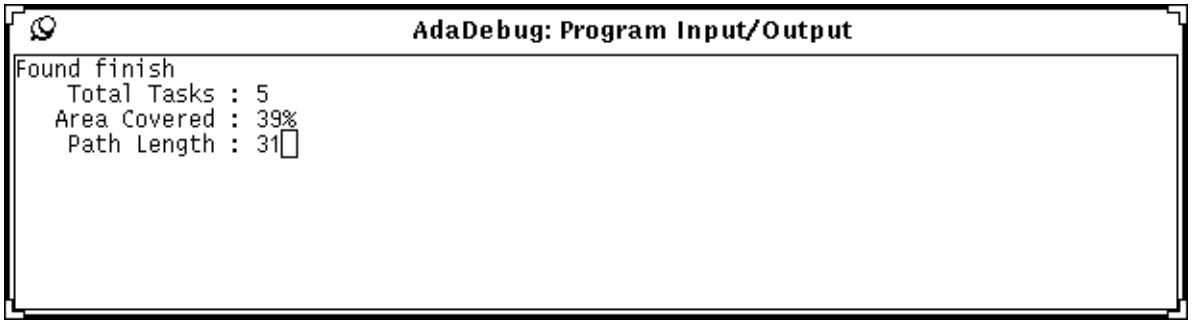
5.6.1.4 Program I/O

Under certain conditions, AdaDebug automatically pops up a Program I/O window.

- When a program writes to the standard I/O, the Program I/O window is opened, displaying the output.
- If user input is required while a program is being run, the Program I/O window opens for input entry.

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.

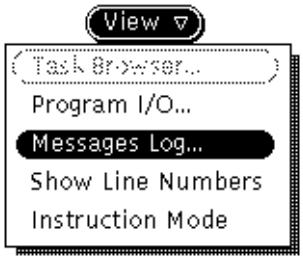
Here is an example of the output displayed in the Program I/O window after running `solve`, the program executable in the `maze` example.



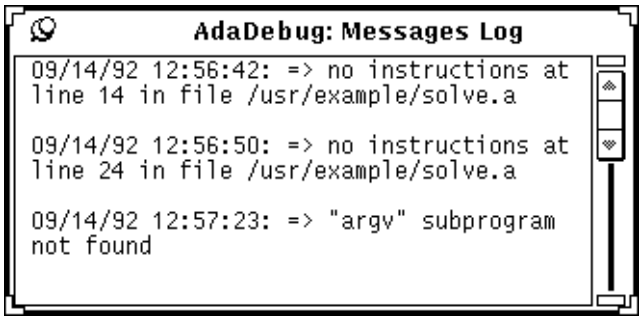
5.6.1.5 *Messages Log*

Sometimes, AdaDebug error and warning messages, which are listed in the left message footer at the bottom of the Program View window, are too long to be displayed completely. On other occasions, a second message overwrites the space too quickly for you to read the first message. You can view these messages later. AdaDebug keeps a file of error and warning messages that are displayed in the footer.

To display a log of AdaDebug error and warning messages:



- **Choose Messages Log from the View menu.**
  The Messages Log window then displays the messages.



5.6.2 *Single-Stepping*

By choosing the Step-over (Next) and Step-into menu items, you can step incrementally through a program. Use Step-over (Next) to advance through source code statement-by-statement or machine instruction-by-instruction. Step-over (Next) steps over subprogram calls. Use Step-into to step into subprogram calls.

AdaDebug places a black arrow glyph ➧ to the left of the next line to be executed.

5.6.2.1 *Step-over (Next)*

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

♦ **Choose Step-over (Next) from the Commands menu.**

Each time you choose Step-over (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.

5.6.2.2 *Step-into*

When you use Step-into, AdaDebug executes the next statement, just as with Step-over (Next). However, if the next statement includes a subprogram call, Step-into displays the call source code 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 Step-into from the Commands menu when the next line to be executed (as indicated by the pointing-arrow glyph) contains a subprogram you want to examine or step through.**

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 pane fields, Stopped in File and Subprogram, will also change, if necessary, to show the new file and subprogram names.

5.6.3 *Breakpoints*

In debugging, it is common practice to preset a breakpoint before running the program. Thus, the program stops at the line where you suspect a bug may exist.

Choosing Commands ⟹ Set Break, you can set breakpoints that will stop the program if it reaches the line where the breakpoint is set. While in source mode, you can specify a breakpoint at a line or at a subprogram. Line is the default choice. While in instruction mode, you can set a breakpoint at an instruction. Breakpoints can also be set at all exceptions in an Ada program.

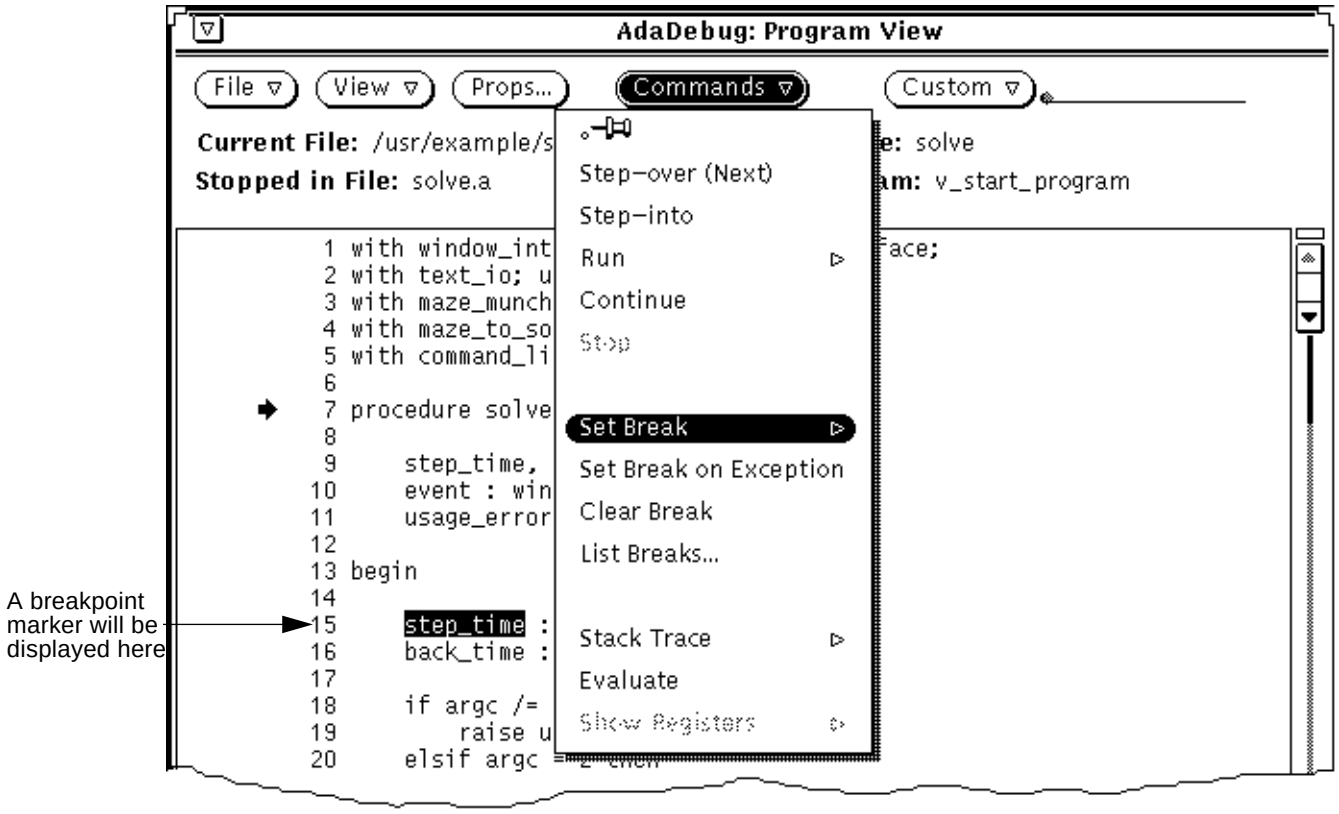
Each breakpoint is marked with a glyph representing the palm of a hand held up as if to signal “stop.”

5.6.3.1 *Breakpoint at a Specific Line*

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

1. **Double-click SELECT on a word in the line where you want to set the breakpoint.**
   Alternatively, triple-click SELECT to select the entire line.

2. **Choose Set Break from the Commands menu.**
   Line is the default choice, so you need not open the submenu.



5.6.3.2 *Breakpoint at a Subprogram*

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

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

```
attack( step_time, back_time );
```

2. **Choose Set Break from the Commands menu, then pull right to choose Subprogram.**

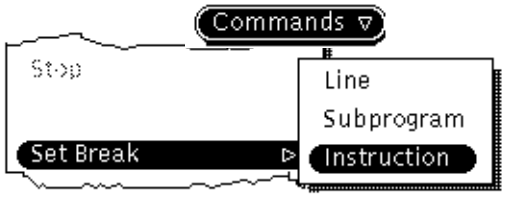
5.6.3.3 *Breakpoint at an Instruction*

If Program View is in instruction mode, another item, Instruction, becomes active.

Note – "Working in AdaDebug" on page 5-4 describes how to switch between source and instruction modes.

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

1. **Double-click SELECT on a word in the line where you want to set the breakpoint for the instruction.**
2. **Choose Set Break from the Commands menu, then pull right to choose Instruction.**



5.6.3.4 *Breakpoint at Exceptions*

AdaDebug can set breakpoints at all exceptions in an Ada program before it is run. You can, for example, check possible errors while the program is being run.

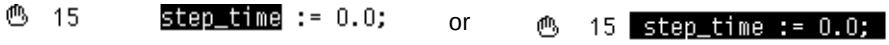
To set breakpoints at all exceptions:

- **Choose Set Break on Exception in the Commands menu.**

5.6.3.5 *Breakpoint Clearing*

To clear a breakpoint:

1. **Select a word in the line containing the breakpoint you want to clear.** Alternatively, select the entire line. For example:



2. **Choose Clear Break from the Commands menu.**

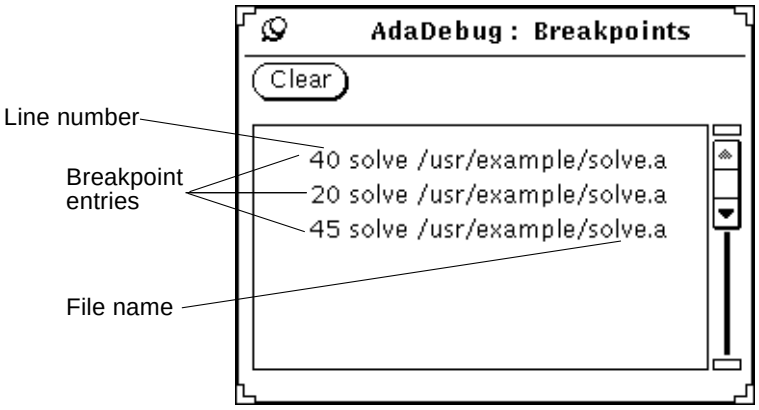
5.6.3.6 *Breakpoint Listing*

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:

- 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 List Breaks from the Commands menu to open the Breakpoints window that lists all of the breakpoints currently set.**
  The Breakpoints window is then displayed.



You can select a line with the break point you want and double-click to go there. To clear a breakpoint that is listed in the window:

- **Click SELECT on the breakpoint entry and press the Clear button.**

5.6.4 The Call Stack

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 currently executing when the program halts at a breakpoint or after a single step is said to be at the *top* of the stack, at level 1. The subprogram that called the currently executing subprogram is pushed down one level on the stack to the next lower-level position. 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 single-tasking program, the main subprogram is always at the bottom of the stack.

5.6.4.1 *Stack Trace Window*

To examine a call stack in AdaDebug, open the Stack Trace pop-up window from the Commands menu.

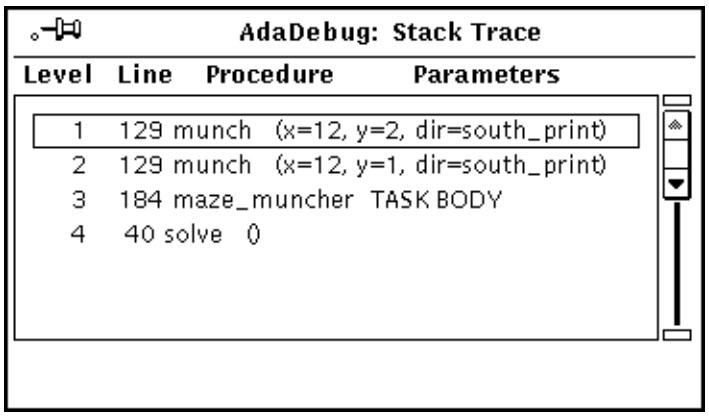
The Stack Trace window lists each stack frame in its own row, called an entry. Each entry shows the call level, line number, and subprogram name. The Stack Trace window also shows the parameters for each subprogram, which help you discriminate among subprograms with overloaded names.

Opening the Stack Trace Window

To open the Stack Trace window:

- **Choose Stack Trace from the Commands menu, then pull right to choose Window.**

Here is a sample Stack Trace window.



Showing the Stack Trace Window Automatically

If you prefer, you can customize AdaDebug, so that the Stack Trace window automatically pops up whenever the program hits a breakpoint. (Refer to “Customizing AdaDebug Properties” on page 5-28.)

5.6.4.2 Code Associated with Each Stack Frame

You can display the code in the Program View or Task View window associated with any stack frame in one of two ways, as follows:

- Click SELECT on a stack frame entry in the Stack Trace window
- Choose the Up, Down, Top, or Bottom menu item located on the Stack Trace submenu under the Commands menu

The Stack Trace window has the advantage of letting you jump to any level in the stack. By choosing the Up or Down menu item, you can display only the code associated with the next frame higher or lower than the current frame; and by choosing the Top or Bottom menu item, you can display only the code associated with the topmost or bottommost frame.

Displaying Code from the Stack Trace Window

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

1. **Open the Stack Trace window by choosing Window from the Stack Trace submenu.**
2. **Click SELECT on a stack frame entry.**

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. The stack frame glyph reminds you that the subprogram you are viewing is not at the top of the stack.

```
  206          first_muncher := start_muncher( start_x, start_y, start_dir );
  207          first_muncher.finish( found_finish );
```

The arrow glyph points to the code associated with the frame at the top of the stack.

Choosing Up, Down, Top, or Bottom

Another way to trace a call on the stack is by choosing one of the items on the Stack Trace submenu from the Commands menu.

To trace calls on the stack using Stack Trace:

1. **Choose Stack Trace from the Commands menu and pull right to display the items.**
2. **Choose one of the items: Up, Down, Top, or Bottom.**

The following diagram illustrates these steps.

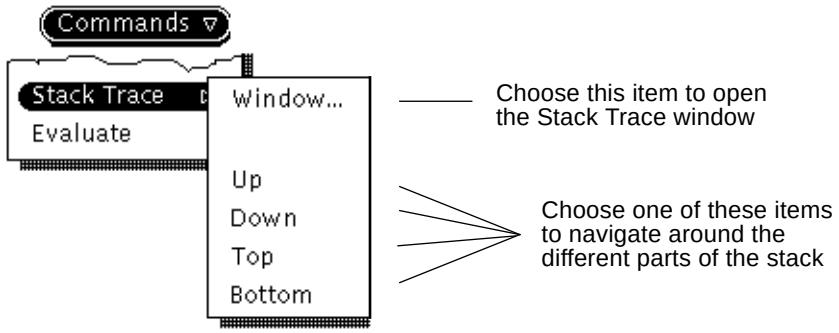


Table 5-1 contains descriptions of the Stack Trace submenu items.

Table 5-1 Stack Trace Submenu Items

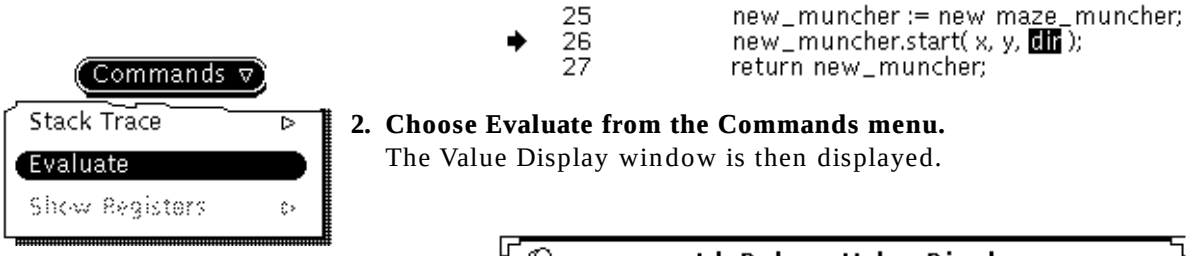
Menu Item	Description
Up	Display the call at the next higher level on the stack, that is, the “callee.”
Down	Display the call at the next lower level on the stack, that is, the “caller.”
Top	Display the call at the top of the stack: the currently executing subprogram.
Bottom	Display the call at the bottom of the stack: in Program View, the main subprogram.

5.6.5 *Expressions*

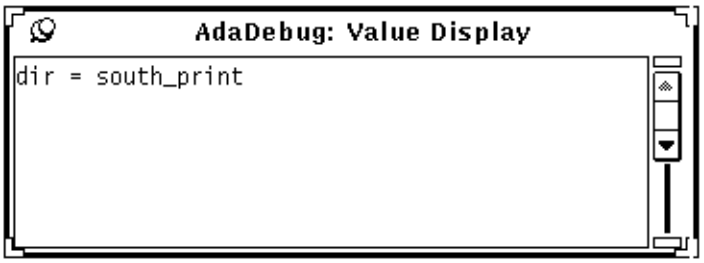
AdaDebug displays the value of selected expressions in a Value Display pop-up window. Each Program View or Task View window has one Value Display window. You can pin the window open and scroll to see the value of expressions evaluated at different times during a debugging session.

To evaluate an expression in AdaDebug:

1. **Select the expression in the Program View (or Task View) display pane.**



2. **Choose Evaluate from the Commands menu.**
   The Value Display window is then displayed.

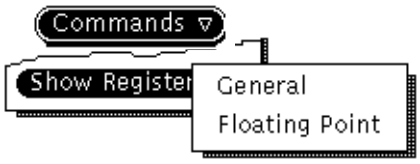


5.6.6 Register Values

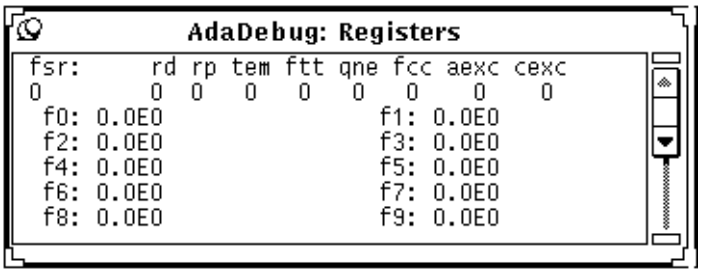
In instruction mode, AdaDebug adds the Show Registers item to the Commands menu. You can choose that menu item to view the current register values of general and, when applicable, floating-point registers in Registers pop-up windows.

To view register values in a Registers window:

1. **Switch to instruction mode. (See “Instruction Mode and Source Mode” on page 5-4.)**
2. **Choose General or Floating Point from the Show Registers submenu under Commands.**



The Registers window is displayed. The following example shows floating point registers.



5.7 Using the Command Line

The Program View window has a text field to the right of the Custom menu button. This field is called the command line. You can enter an `a.db` line command on the command line. (Refer to the SPARCompiler Ada documentation for the various commands.)

5.7.1 Command Entry

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

1. **Click SELECT in the control area to obtain the input focus, if the control area does not already have the focus.**
   The input caret is dimmed if the area does not have the focus, highlighted if it does.

Custom ▽ /accept start

2. **Type the line command in the command line.**

3. **Press Return.**

5.7.2 Command to Assign a Value

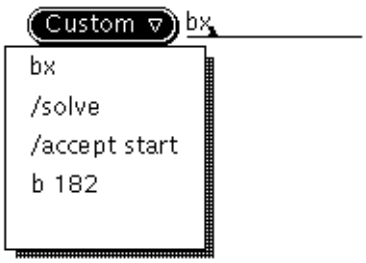
In addition to the `a.db` command :=, AdaDebug accepts an assign command:

Custom ▽ assign foo:=3

1. **Type `assign` before the syntax** *name* `:=` *expression* **in the command line.**
   An example is shown on the left margin. Refer to the SPARCompiler Ada documentation for the assignments that are supported.

2. **Press Return.**

5.7.3 Custom History



After you have entered an `a.db` command at the command line and pressed Return, AdaDebug adds the item to the Custom button menu.

Custom keeps a history list of up to 10 commands that you have entered at the command line. Pull down to choose the item you want to execute the command again.

The list of commands is a list, not a stack. For example, if you enter the same command twice, it is displayed only once on the list.

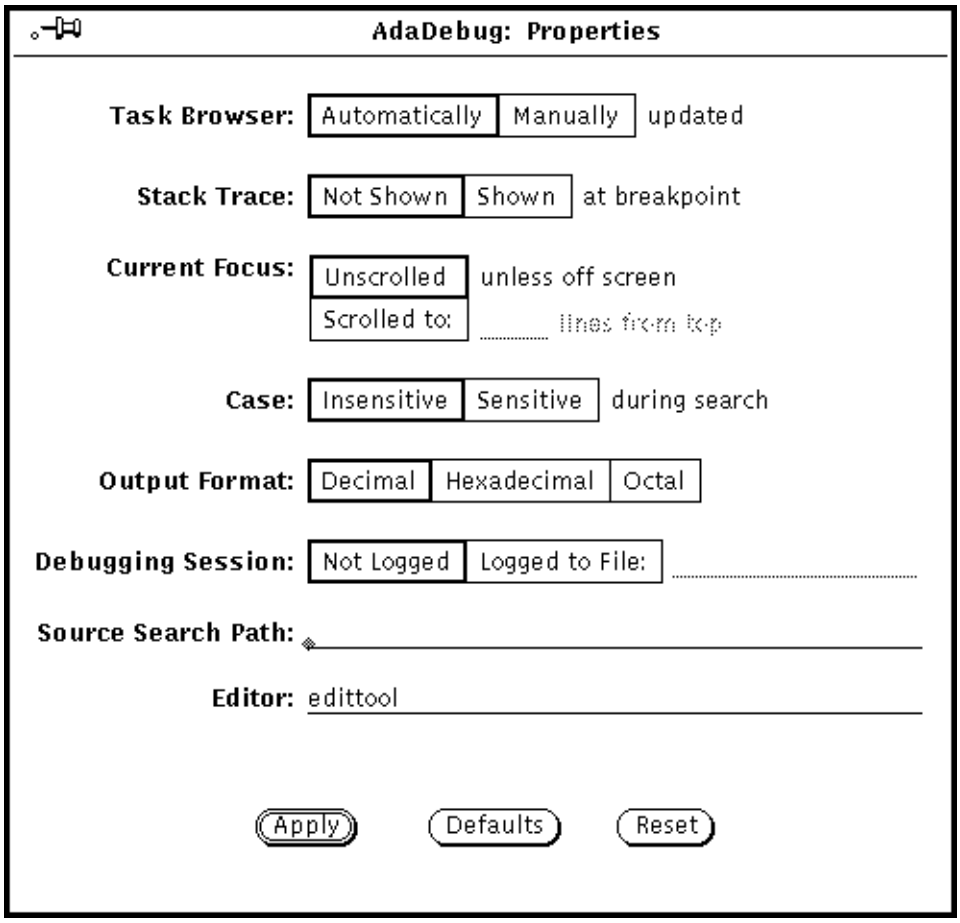
5.8 Customizing AdaDebug Properties

The Properties window contains controls for a number of features you can customize. Default or current settings are shown with thick dark borders.

Note – The Properties window is the same for the three main AdaDebug windows—Program View, Task Browser, and Task View. Task Browser and Task View are discussed in Chapter 6, “Debugging Tasks with AdaDebug.”

To open the Properties window:

- **Click SELECT on the Props menu button on the Program View window.** The Properties window is then displayed.



Be sure to press Apply after setting your options.

AdaDebug saves all property settings once they have been modified. Next time you restart AdaDebug, these settings remain the same as they were set for the previous session.

Table 5-2 summarizes the properties that you can set.

Table 5-2 Property Controls for AdaDebug

Property Controls for AdaDebug		
Task Browser	Automatically updated	Update the Task Browser display pane when the program reaches a breakpoint or after each single step.
	Manually updated	Update the Task Browser display pane when you close and then reopen the Task Browser, or when you issue an `lt` (list task) command from the command line.
Stack Trace	Not shown at breakpoint	Do not automatically pop up the Stack Trace window when the program reaches a breakpoint. You choose Commands ⟹ Stack Trace ⟹ Window to open.
	Shown at breakpoint	Automatically pop up the Stack Trace window when the program encounters a breakpoint.
Current Focus	Unscrolled unless off screen	Display the line corresponding to the current focus somewhere in the display pane. Faster than the alternative option.
	Scrolled to:*n* lines from top	Scroll display pane until the line of code corresponding to the current focus is *n* lines from the top of the display pane.
Case	Insensitive during search	The `a.db` search command ignores uppercase/lowercase distinction.
	Sensitive during search	The `a.db` search command respects uppercase/lowercase distinction.
Output Format	Decimal, Hexadecimal, Octal	Display formal numerical output as specified when evaluating expressions.
Debugging Session	Not Logged	Do not keep an `a.db` log file for the current session.
	Logged to File	Write an `a.db` log for this session to the file name entered in the text field.
Source Search Path		Specify a search path name for use with AdaDebug.
Editor		Specify your editor of choice. The default is `edittool`.

Debugging Tasks with AdaDebug 6

This chapter describes how to debug Ada tasks by means of the AdaDebug Task Browser and Task View facilities.

Note – This chapter assumes that you know the principles and techniques of debugging Ada tasks and have read the previous chapter on using AdaDebug. This chapter explains how to use the AdaDebug interface to the Ada `a.db` tools for debugging tasks.

The chapter is organized into the following sections:

6.1 Introducing the Task Browser

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

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

Subsequently, you obtain detailed information about any specific task by selecting it and then opening its Task Status window. This functionality is described in greater detail in “Displaying Task Status” on page 6-20.

6.1.1 Task Browser Startup

Open the Task Browser from the View menu in Program View. If more than one task is active, then AdaDebug activates the Task Browser item and makes it the default choice on the View menu. If no tasks are active other than the main task, this menu item is inactive (dimmed).

Note that the Task Browser displays tasks, not task units. AdaVision displays task units in AdaVision compilation unit (CU) mode. See “Three Types of Icons” on page 4-11.

To open the Task Browser:

♦ **Choose Task Browser from the View menu.**

Figure 6-1 shows the Task Browser window in icon mode, the default display mode when this window is first opened. In this example, the Task Browser is displaying task objects that were active while debugging `solve`, the executable in the `maze` example.

Note – “Tasks in Icon Mode or List Mode” on page 6-9 discusses the two display modes for the Task Browser.

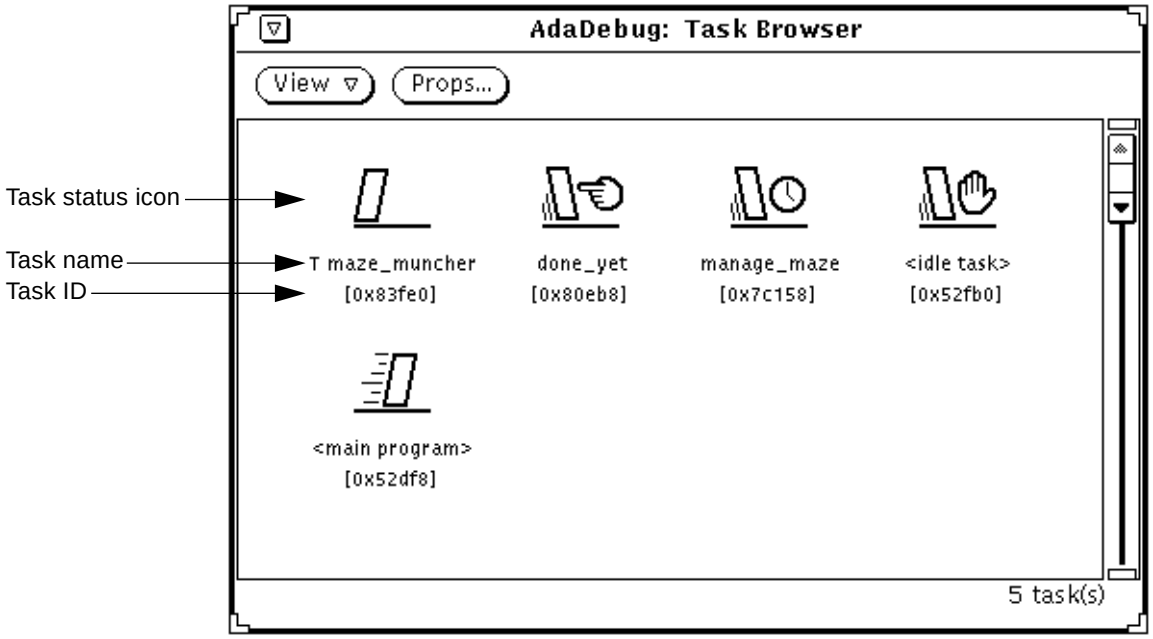


Figure 6-1 Task Browser window

The Task Browser can be closed to an icon.

6.1.2 Task Objects and Icons

The Task Browser represents each task as a task object, displaying an icon which reflects the task object itself, along with the task name and task ID number. Each task state has a different task icon image. “Understanding Task Status Icons” on page 6-4 shows a complete list of task state icons.

You handle task objects in much the same way that you handle icons in AdaVision. You can display task objects in icon mode or in list mode. You can also sort and filter task objects by state or name and search for tasks that match a specified pattern. The controls for these features are items on the View menu. Their uses are described in “Changing the View of Task Objects” on page 6-8.

6.1.3 Properties Window

Pressing the Properties button in the Task Browser window invokes the Properties window.

This window is the same for the three main AdaDebug windows—Program View, Task Browser, and Task View. Task Browser and Task View are discussed in this chapter; Program View is discussed in Chapter 5, “Using AdaDebug.”

“Customizing AdaDebug Properties” on page 5-28 gives a full description of the controls provided in the Properties window.

6.2 Understanding Task Status Icons

The Task Browser uses a different icon to represent each of the states a task may be in. A few states have substates, but there are no special icons for the substates.

6.2.1 List of Task Status Icons

This section identifies and briefly describes each icon and the task state it represents.

`attempting rendezvous`

The task is attempting 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, "called in rendezvous" 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 "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.

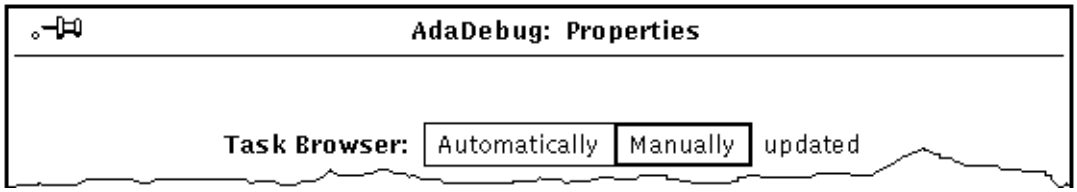
`waiting to exit`

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

6.2.2 Task Object Display Update

By default, AdaDebug automatically updates the list of task objects each time the program stops at a breakpoint. When the program reaches a breakpoint, AdaDebug checks with `a.db` to see if any tasks are still active. It then revises the list of task objects.

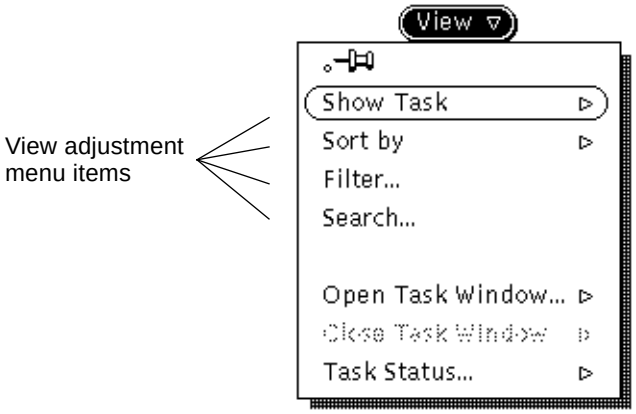
You can turn off the automatic update facility by setting the Manually updated option in the Task Browser control located in the Properties window in either Program View, Task Browser, or Task View.



See “Customizing AdaDebug Properties” on page 5-28 for a full description of the Properties window.

6.3 Changing the View of Task Objects

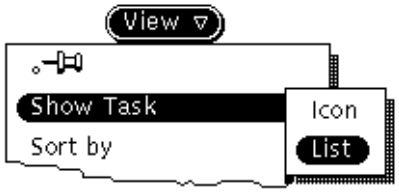
To adjust the view of task objects, choose from the first group of items on the Task Browser View menu.



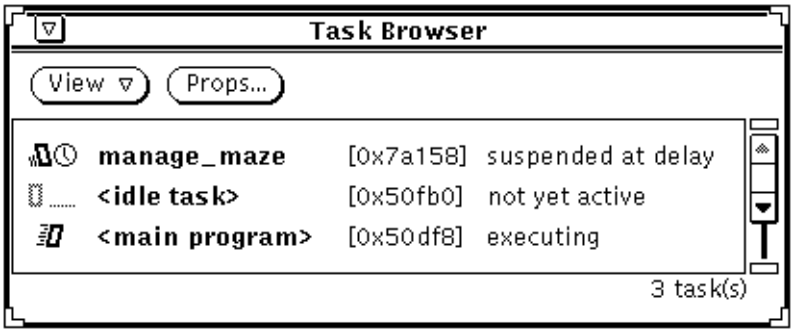
6.3.1 Tasks in Icon Mode or List Mode

Tasks can be displayed in icon or list mode by choosing either of the following:

- View ⟹ Show Task ⟹ Icon
- View ⟹ Show Task ⟹ List

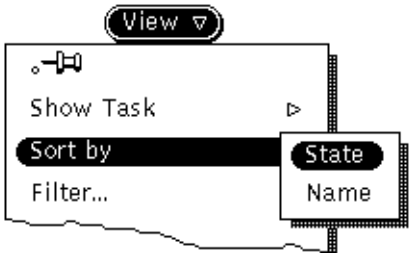


The following figure illustrates list mode where, for each task, the Task Browser shows a down-sized version of the task icon, name, ID, and the task state.

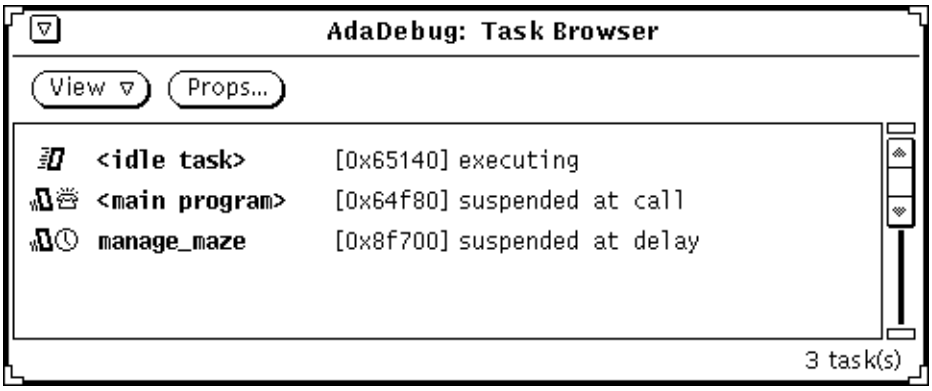


6.3.2 Task Object Sorting Facility

When the Task Browser first opens, tasks are displayed in the order in which they appear in the task queue. The Sort by item sorts task objects by *state* or *name*. The choices are on the submenu. State is the default.



The following figure shows some tasks being displayed in the Task Browser window in list mode, sorted by state.

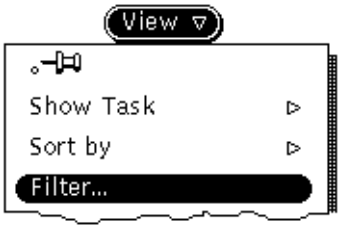


6.3.3 Task Object Filtering Facility

You can also use the Task Browser to filter a list of tasks by *state*, by *pattern*, or by a *pattern within* a list filtered by state.

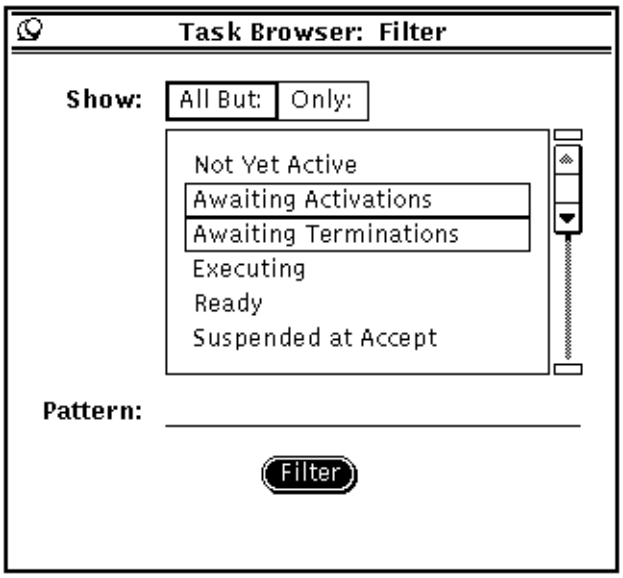
To filter by task state, use the Show control on the Filter pop-up window. When you set Show to the All But option (the default setting), you filter out tasks in those states you have selected. When you set Show to Only, the Task Browser displays only those tasks in selected states. Use the Pattern control to filter tasks with a specific pattern, typically the task name.

6.3.3.1 Filtering Out Tasks in Selected States



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

1. **Choose Filter from the View menu.**
   The Filter popup window is displayed.



2. **Set the Show control to All But, if not already set.**
3. **In the scrolling list, click SELECT on the state(s) for which you want corresponding tasks removed from the display.**
4. **Press the Filter button.**

6.3.3.2 Showing Only Those Tasks in Selected States

Show Only is the reverse of the Show All But form of the filtering facility.

To show only those tasks in the states you select:

1. **Choose Filter from the View menu to open the Filter pop-up window.**
2. **Set the Show control to Only.**
3. **In the scrolling list, click SELECT on the state(s) for which you want corresponding tasks in the display.**
4. **Press the Filter button.**

6.3.3.3 *Filtering by Task Name*

By setting the Pattern control in the Task Browser Filter, you can display task objects whose names match the pattern entered in the text field. Type in any regular expression as the pattern.

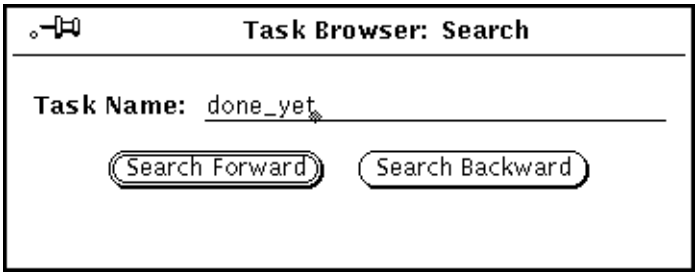
To redisplay all tasks after filtering for those that match a pattern, toggle the Show control setting and press the Filter button.

6.4 *Searching for a Task*

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

To search for a task:

1. **Choose Search from the View menu to open the Search pop-up window.**
2. **Enter the name of the task.**
3. **Press either the Search Forward or Search Backward button.**
   Pressing the Return key activates Search Forward, the default.



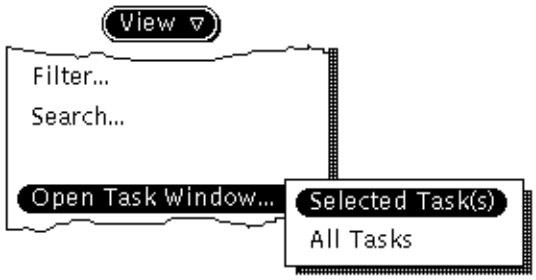
As soon as the task is found, it is selected in the display pane.

6.5 *Opening the Task View Window*

The Task View window displays the code which that task was executing. You can open as many Task View windows as there are tasks, one for each task.

To view a task in a Task View window:

1. **Click SELECT on the task object of your choice.**
2. **Choose Open Task Window from the Task Browser View menu, then pull to the right to choose Selected Task(s).**
   A double-click on a task icon acts as an accelerator to open the Task View window for that task.



The submenu also offers another choice, All Tasks, for you to view all the tasks shown in the Task Browser.

Note – If you do not select a task before pulling to the right from Open Task Window, the Selected Task(s) item is inactive.

The Task View window is displayed. Figure 6-2 is an example.

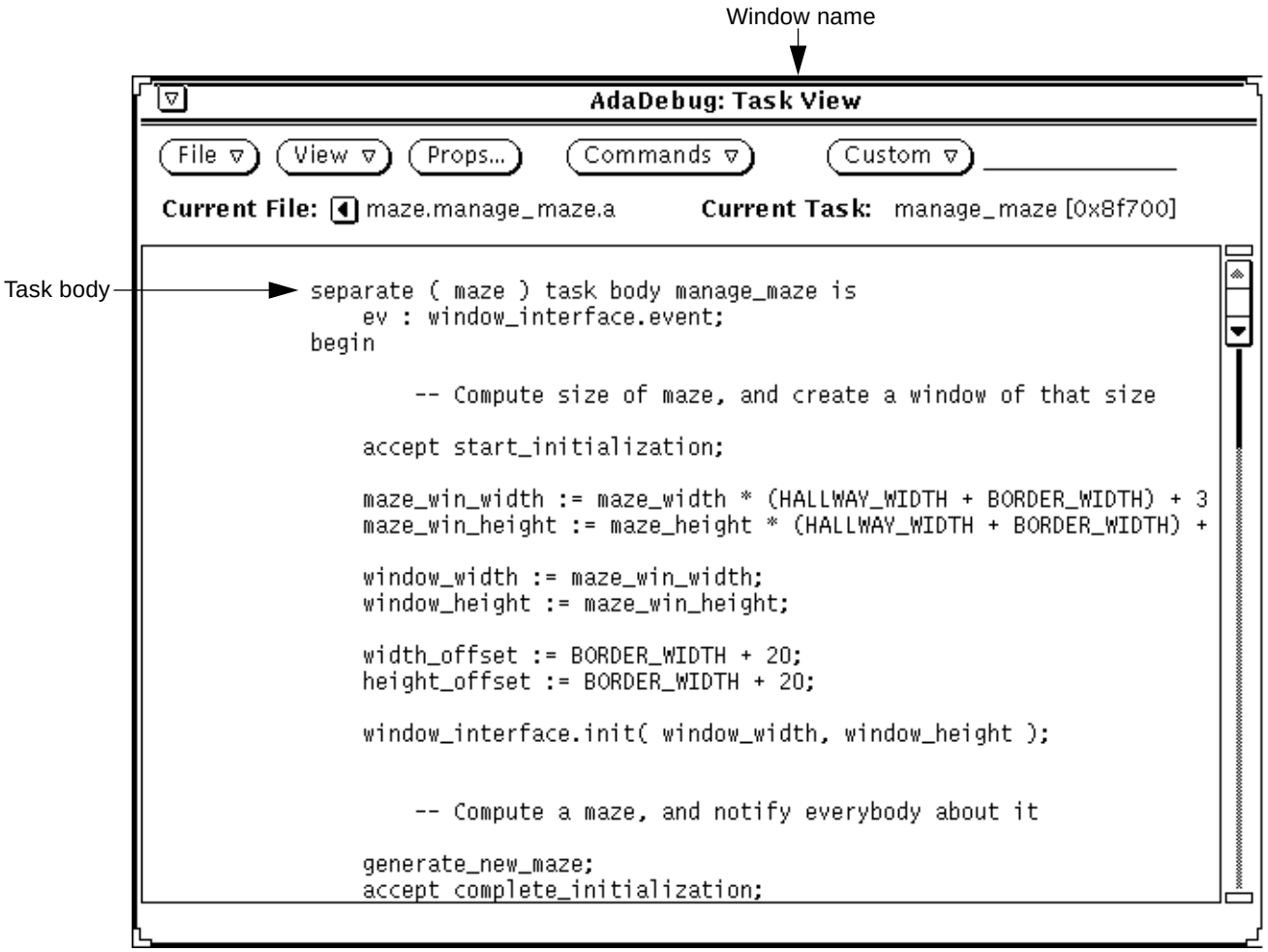


Figure 6-2 Task View window in AdaDebug

The Task View window can be closed to an icon.

Once a Task View window is opened, an item in the View menu, Close Task Window, becomes active. Choose that item and pull right to close the Task View window(s) for either the selected task(s) or all tasks.

6.6 Debugging Tasks

You debug specific tasks from within Task View windows. 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 treat each separate thread of control independently of the others.

6.6.1 Task View Startup

To debug a task, first open the task into a Task View window by doing either of the following:

- Double-click SELECT on its icon in the Task Browser.
- Click SELECT on the task icon and choose Open Task Window ⟹ Selected Task(s) from the View menu.

This process is described in “Opening the Task View Window” on page 6-13.

With few exceptions, a Task View window looks and behaves just like a Program View window. The difference is 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 window and the Task View window simultaneously. For instance, if you open a task and set a breakpoint in the Task View window, then, 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 window and the Program View window.

6.6.2 Breakpoints in Task View

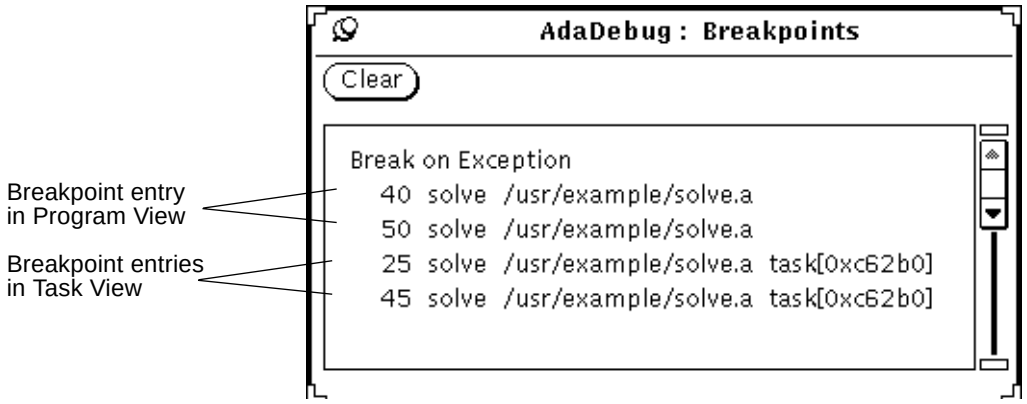
A breakpoint set in a specific Task View applies to that task only. All other tasks that execute the same line of code will continue as though there was no breakpoint. A breakpoint set in the Program View applies equally to all tasks. The Task View window shows *only* those breakpoints you set for that task.

To see a list of breakpoints, open the Breakpoints window:

♦ **Choose List Breaks under the Commands menu.**

Following is an example of a Breakpoints window that lists a breakpoint entry in Program View and two breakpoint entries in Task View.

Aside from the file name and line number, a breakpoint entry in Task View shows the task name and task ID.



6.6.3 Task View Commands and Custom Menus

In a Task View window, the Commands and Custom menu items are the same as those on the Program View window. When pinned up, the Task View Commands menu header includes the task ID that distinguishes this menu from the Program View Commands menu.

The following diagram illustrates this difference between the two pinned-up menus.

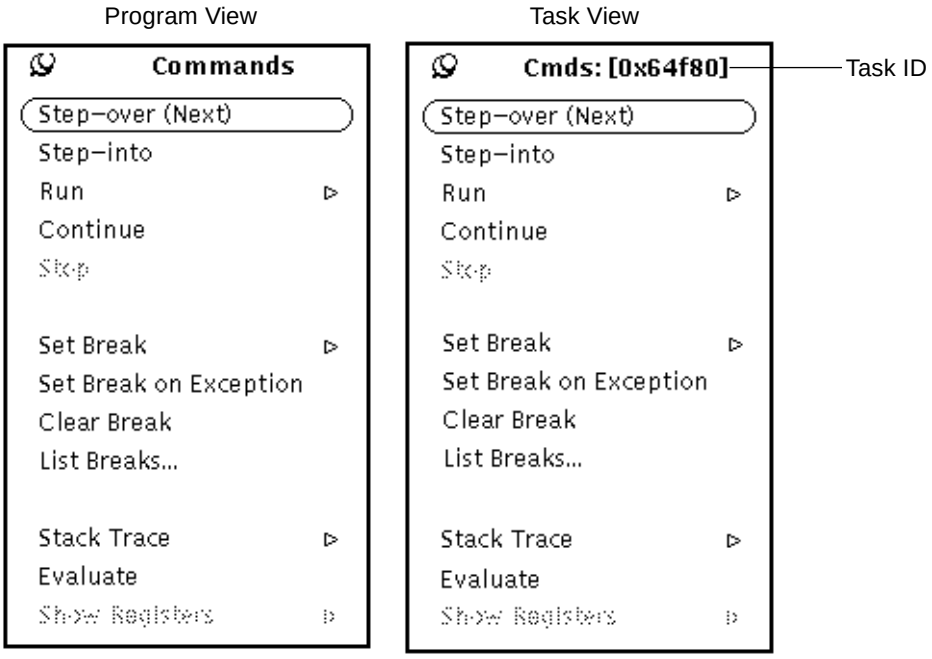


Figure 6-3 shows the items on the Commands menu, together with those on the four submenus. The four submenus are “opened” to show you where each of these basic menu items is located. In practice, you can open only one submenu at a time.

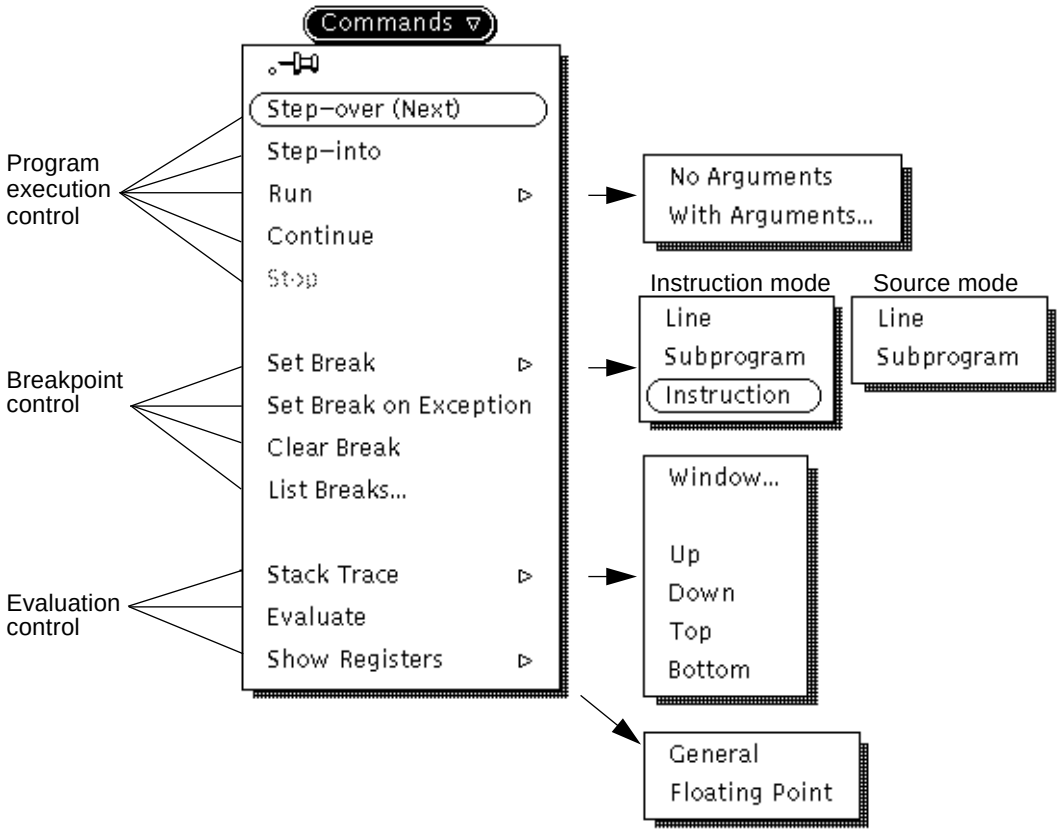


Figure 6-3 Program and Task View Commands Menu

Note – Certain items are dimmed when they do not apply. For example, the last item on the menu, Show Registers, is active only in instruction mode, dimmed in source mode.

The following items in the Commands menu apply to all tasks regardless of whether they are issued from a Task View or a Program View window:

- Step-over (Next)
- Step-into
- Run
- Continue

- Stop
- List Breaks
- Show Registers

The other items, with the exception of Evaluate, apply only to the task displayed in the Task View window when chosen from that window. Evaluate applies to the selection only. The selection might or might not be task-specific.

6.6.4 Task Stack Frames

To display the stack for a particular task:

- **Choose Stack Trace ⟹ Window from the Task View Commands menu.** The Stack Trace window is then displayed, showing the stack for the task.

  You can have multiple Stack Trace windows open at the same time, one for each Task View. (See “Stack Trace Window” on page 5-23 for more information.)

6.7 Displaying Task Status

To display information about a selected task in a Task Status window:

1. **Click SELECT on a task in the Task Browser display pane.**
2. **Choose Task Status from the View menu and pull to the right to choose Selected Task(s) to open the Task Status window.**

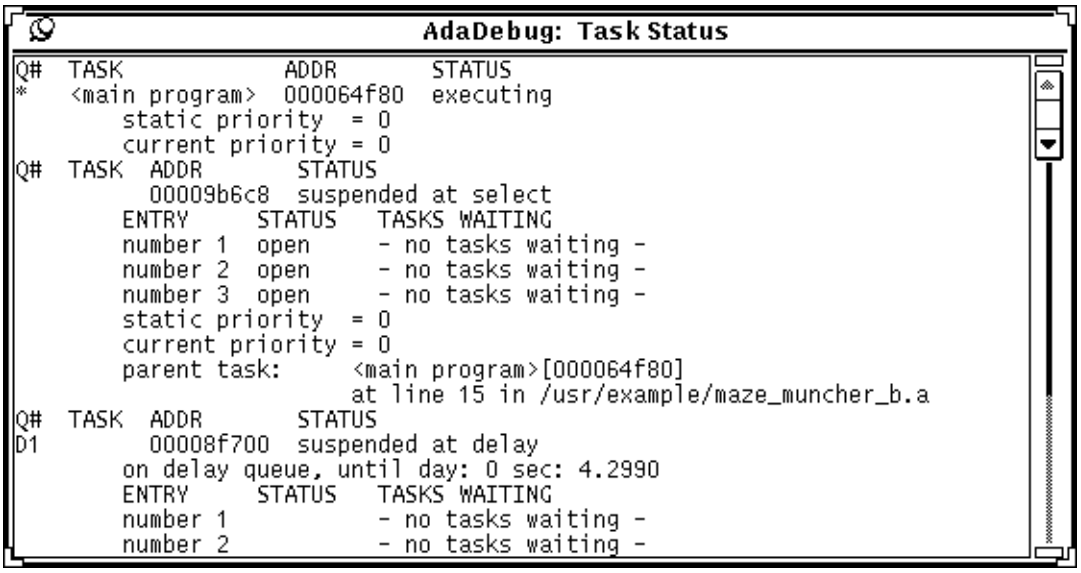
If you do not select a task before opening the Task Status submenu, then the Selected Task(s) item is inactive.

You can view the status information of all the tasks in one Task Status window:

- **Under View, pull right on Task Status to choose All Tasks.**

6.7.1 Task Status Window

Here is an example of a Task Status window.



This window displays the following information:

- Name, address, and status for the selected task
- 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.

You can pin the Task Status window to the Desktop. Use the scrollbars to display information about the tasks.

6.7.2 Life History of Tasks

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

6.7.2.1 Task Creation

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

In Program View, you can place a breakpoint at the `begin` statement in a task body. When you run the program, it will reach this breakpoint when a task first attempts to execute the task body statements. Because you placed this breakpoint in the task body from Program View, 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; you do so in a Task View window. See “Opening the Task View Window” on page 6-13.

6.7.2.2 Task Termination

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*.

6.7.2.3 Main Program Task

In Ada programs, the main program is itself a task, so the Task Browser always displays an object for it. In the Task Browser, the main program task has 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 Browser display pane. The *Ada Language Reference Manual* provides more details on the life history of tasks.

Using LRMTool 7

The *Reference Manual for the Ada Programming Language*—known informally as the Language Reference Manual, or LRM—contains the complete, official definition and description of the Ada language. LRMTool presents the manual on-line as one continuous document. It displays the manual in one or more text pane(s) of an OpenWindows base window. LRMTool consists of a set of user-interface features and facilities that help you navigate easily through the document, displaying sections that you are interested in reading.

LRMTool features include:

- Active table of contents—Displays the part of the manual you select from the table of contents
- Split-pane multiple views of the manual—Displays several portions of the manual at the same time in separate panes
- Linked reference and index entries—Displays any chapter or section referenced in the text or listed in the index
- Find expression facility—Locates and displays occurrences of any word or pattern of characters (any regular expression)
- Index search facility—Searches the index and lists all occurrences of an expression; you select from the list to display the index entry.
- History facility—Tracks your search path and allows you to return to a recently visited location in the manual.

7

This chapter describes how to use LRMTool, the SPARCworks/Ada on-line version of the *Reference Manual for the Ada Programming Language*.

The chapter is organized into the following sections:

7.1 Starting LRMTool

Start LRMTool from a Command or Shell Tool, from the AdaVision Commands menu, or from the SPARCworks Manager palette.

To start LRMTool from a Command or Shell Tool:

- **Type the following at the system prompt and press Return:**

```
lrmtool &
```

To start LRMTool from within AdaVision:

- **Choose LRMTool from the Commands menu.**

To start LRMTool from within the SPARCworks Manager palette:

- **Double-click the LRMTool icon in the palette, or drag and drop the icon onto the workspace.**
  See "SPARCworks Manager" on page 1-2 for details.

When you first start LRMTool, it always displays the first page of the manual. Figure 7-1 shows the LRMTool base window with that page being displayed.

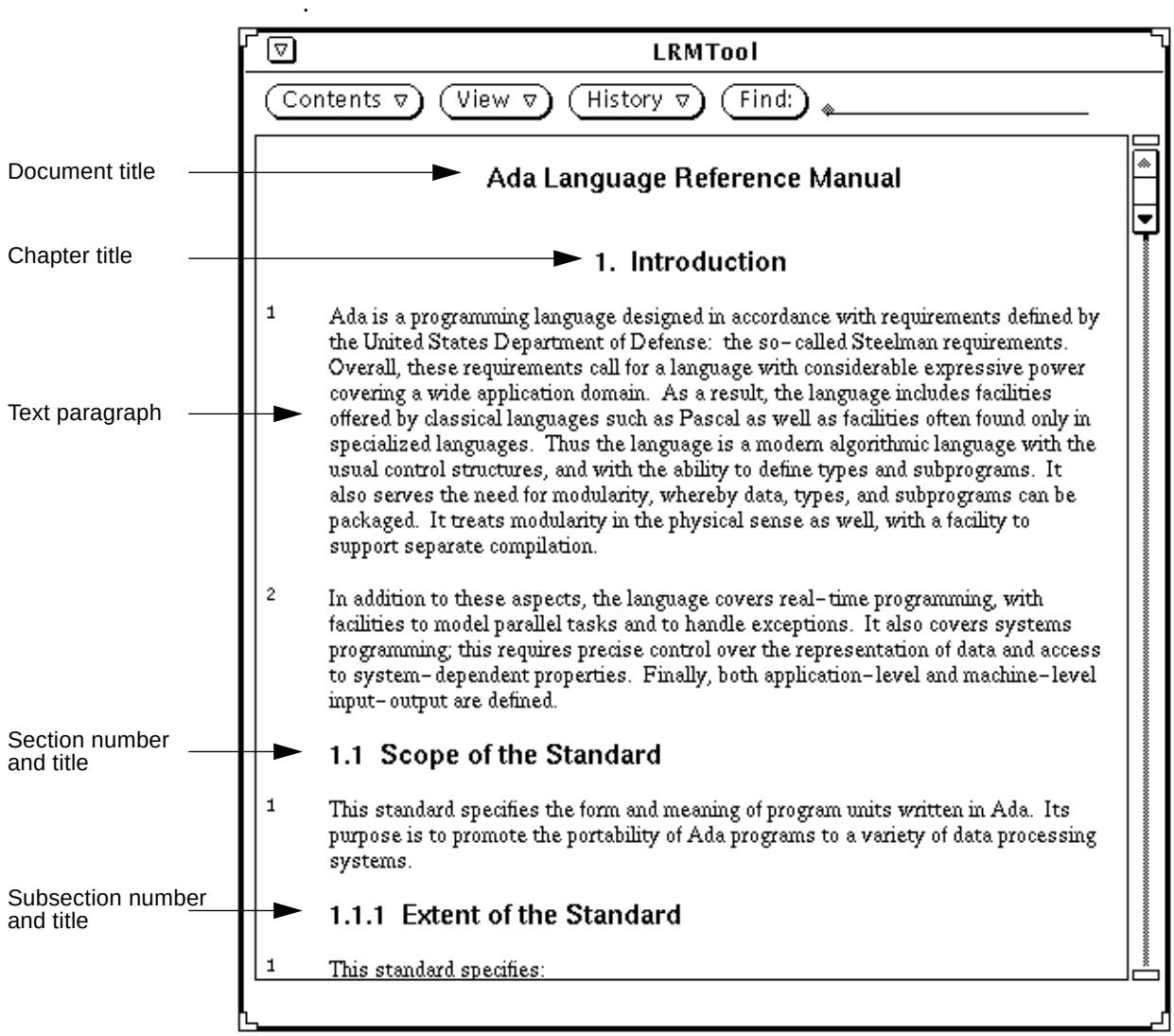


Figure 7-1 Base window for LRMTool

Like the other SPARCworks/Ada tools, LRMTool runs independently in its own OpenWindows base window. You can move, resize, and scroll it, and close it to its icon. You can leave it running on your workspace for as long as you want, independently of the other SPARCworks/Ada tools.

7.2 Understanding the Features and the Interface

This section presents a quick summary of how to use LRMTool. "Understanding Details of the LRMTool Interface" on page 7-9 provides details on how to use each feature.

7.2.1 Entries in the Table of Contents

The LRM is divided into numbered chapters, sections, subsections, appendices, an index, and Appendix F, which contains SPARCompiler Ada-specific language implementation information. These divisions are reflected in a complete table of contents.

When you choose a table of contents entry, LRMTool displays that section. It also records the move on the History menu.

LRMTool presents the table of contents in two styles:

- Menu entries on the Contents button menu (also on the display pane floating pop-up menu).
- Entries in a hierarchical (multilevel) scrolling list. (Choose Table of Contents from the View menu.)

Both styles offer the same functionality. The section "Table of Contents" on page 7-9 describes these facilities in detail.

7.2.2 Links

The LRM has a large index at the end of the document. It also lists references at the end of sections. LRMTool creates *links* from index and reference entries to the sections to which they refer.

There are two kinds of links:

- *Section number* links—When you activate a section number link, LRMTool displays the section associated with it.
- *Word or phrase* links—When you activate a *word or phrase* link, LRMTool displays the index entry for it.

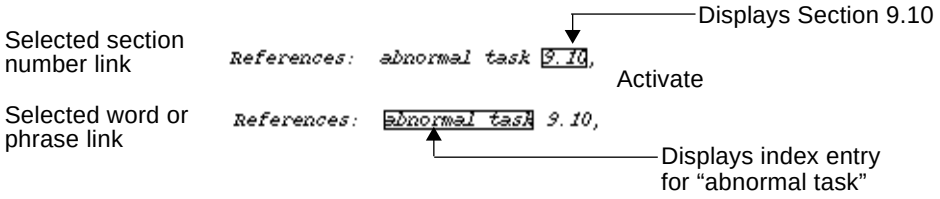
You activate a link in one of two ways:

- Double-click SELECT on a link to replace the text in the current pane with the linked text.

  To choose a selected link, you can also position the pointer over the link-box, click MENU to open the submenu, and choose Goto, which echoes the selection.

- Select a link and then drag and drop it to a second display pane. LRMTool then displays the linked text in the drop pane.

  When you select a link, LRMTool draws a box around the selection. The box distinguishes a link selection from a standard text selection (OpenWindows uses reverse video by default).



When you split the display pane into multiple views of the manual, you can select a link and then drag and drop it onto another pane. LRMTool displays the section or index entry associated with the selected link.

7.2.3 Find Facility

LRMTool scrolls the text to the next occurrence of the word or expression you ask it to find. Also, if you enter a valid section number or appendix letter (A-F) in the Find text field, LRMTool takes you to that section or appendix when you press the Find button.

7.2.4 Index Search

The Index Search pop-up window (from the View menu) lets you instruct LRMTool to list all index entries containing the word or phrase you enter in the Search Pattern text field. You then click SELECT on an entry, and LRMTool displays the selected index entry in the active LRMTool window pane. (The active pane is the last pane you clicked in.)

7.2.5 History Feature

Each time you display a different section of the manual, History records your movements on a menu that works like an 8-slot stack, with the current location listed at the top. You can return to any of the last seven locations recorded.

7.2.6 Scrolling Facility

You can scroll from start to finish. The scrollbar elevator indicates your location in the document as you move the elevator up and down on the cable. History ignores line-by-line (click) scrolling.

7.3 Displaying Multiple Views of the Manual

LRMTool is especially effective when you use its multiview facility. Beneath the control area is a large, scrollable display pane where LRMTool displays the text of the manual. When you want to browse or search for information on a number of related topics, you can split the display pane into two or more smaller panes, showing multiple views of the manual. For example, you can display the index or a set of references in one pane, and then use another pane to display the sections to which the index or reference entries refer.

7.3.1 Split View Facility

To split the display pane:

1. **Position the pointer in the Scrollbar window at the location where you want the pane to split. Press MENU for the Scrollbar menu.**
2. **Choose Split View.**

Alternatively, split the pane by using the *anchor bar* at the top of a scrollbar. Grab the anchor by positioning the pointer over it and pressing SELECT. Drag the arrow down to where you want to split the window and release the mouse button. (As you move the pointer, a line is displayed that shows where the window will split when you release the button.)

Figure 7-2 illustrates the display pane before and after it was split.

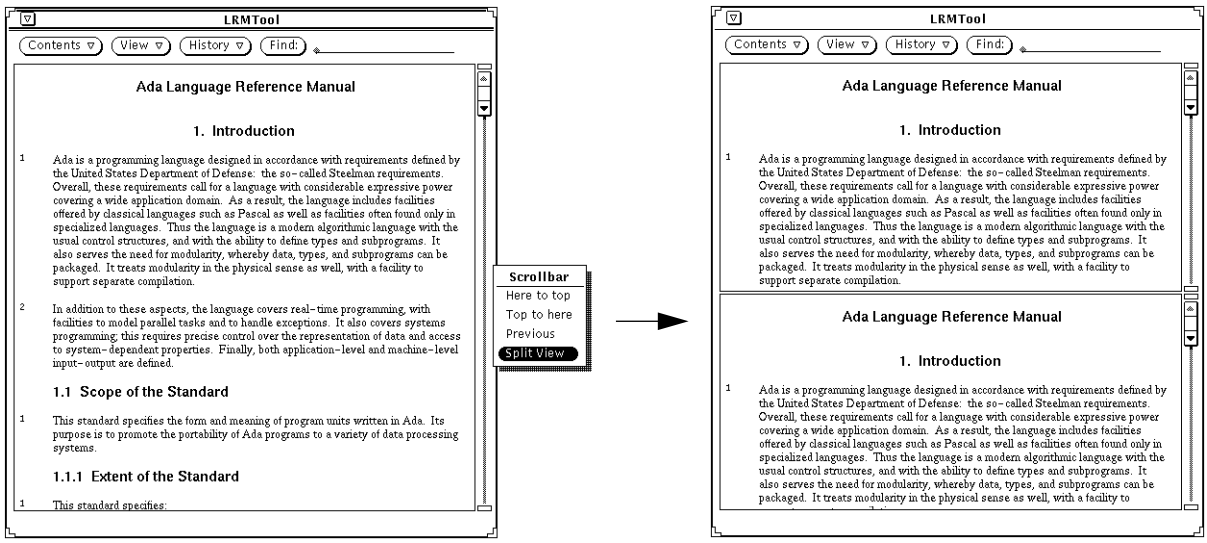


Figure 7-2 Splitting the LRMTool display pane for multiple views

7.3.2 Dragging and Dropping Links from One Pane to Another

To display text associated with a selected link in a pane, use the link drag-and-drop feature:

- **Click SELECT on a link and then drag and drop it onto another pane.** The text is then displayed in the pane to which the link was dropped.

7.3.3 Split Pane Joining

Join two split panes from the scrollbar menu:

- **Choose Join Views from the Scrollbar menu *from the pane you no longer want to view.***
  When you join views, the text in the pane from which you selected the menu item disappears.

When you join two split panes, the pane beneath the one from which you chose Join Views recovers the space—except when you join the panes from the bottom pane, in which case the pane above it recovers the space.

Figure 7-3 illustrates joining of a split pane.

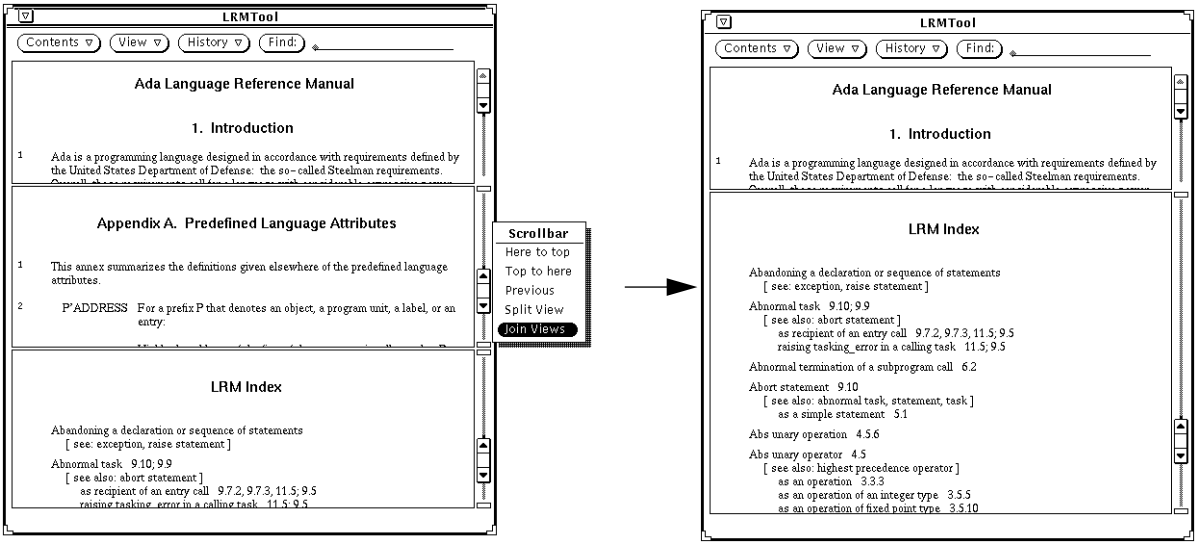


Figure 7-3 Joining two panes in LRMTool

7.3.4 Active Display Pane

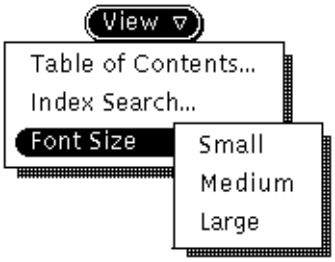
When you split the display pane into multiple panes, only one pane is active at any time; the active pane is always the last pane in which you clicked the mouse.

The active pane comes into play when you are using the Index Search facility, the Table of Contents pop-up window, and the Find button. Index Search, for example, displays an index entry in the active pane, displacing whatever was displayed there previously. (See “Index Search” on page 7-6.)

7.4 Understanding Details of the LRMTool Interface

This section describes in more detail how to use the LRMTool features and facilities summarized in the previous section.

7.4.1 Font Size Changes in the Display Pane



You can change the font size of the text in the LRMTool display pane.

To change the font size:

♦ **Choose Font Size from the View menu. A submenu lets you choose among Small, Medium, and Large fonts.**

7.4.2 Table of Contents

LRMTool displays all of the entries in the LRM table of contents in two forms:

- *Menu items* on the Contents menu button, which is located in the control area of the main LRMTool window; the table of contents in menu form is also available on the display pane floating pop-up menu.
- *Entries* in a hierarchical scrolling list. The scrolling list is displayed in a pop-up window when you choose Table of Contents from the View menu.

Both forms present the same table of contents and provide the same functionality. When you choose a table of contents item from the Contents menu or an entry from the scrolling list in the pop-up window, LRMTool displays the section of the manual you chose. The only difference between the two tables of contents is the style of the interface. LRMTool offers both styles for your convenience.

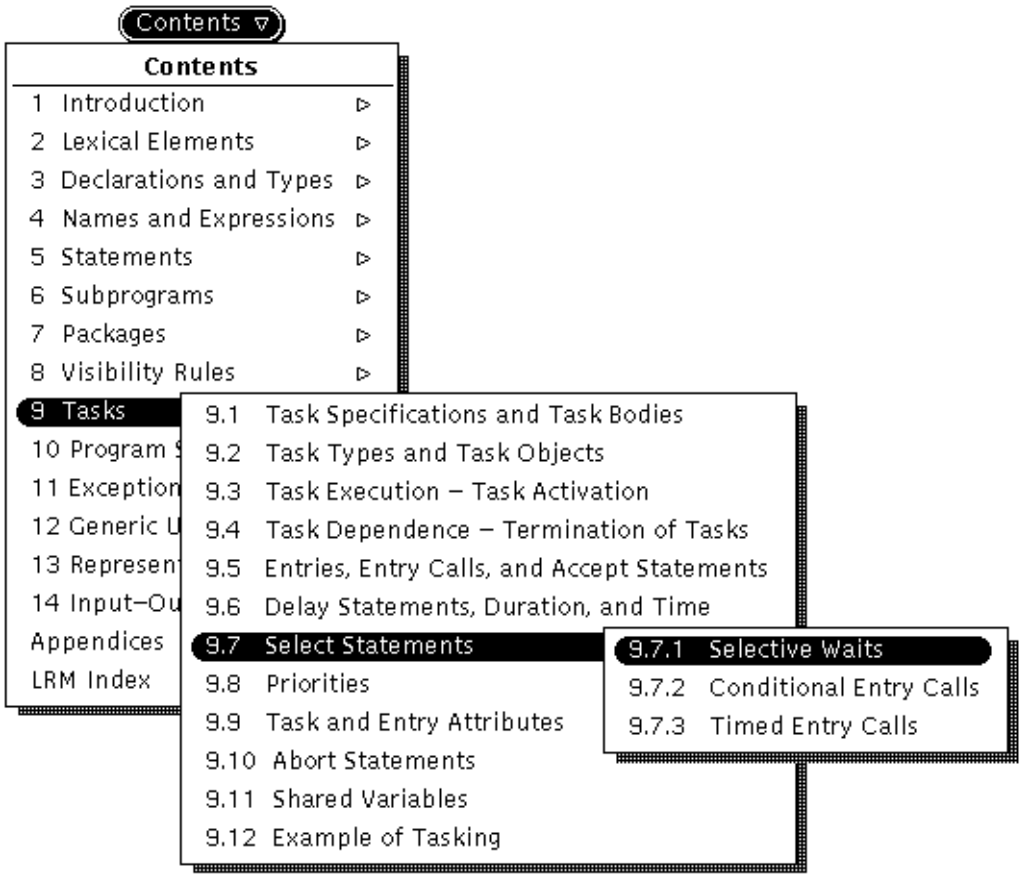
7.4.3 Contents Menu

The Contents pull-down menu lists chapters, appendices, 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.

To use the Contents menu:

- **Choose the entry that corresponds to the part of the manual you want to display.**

When you choose a section, LRMTool displays the section in the active pane and enters the new location at the top of the History stack.



7.4.4 *Table of Contents Scrolling List*

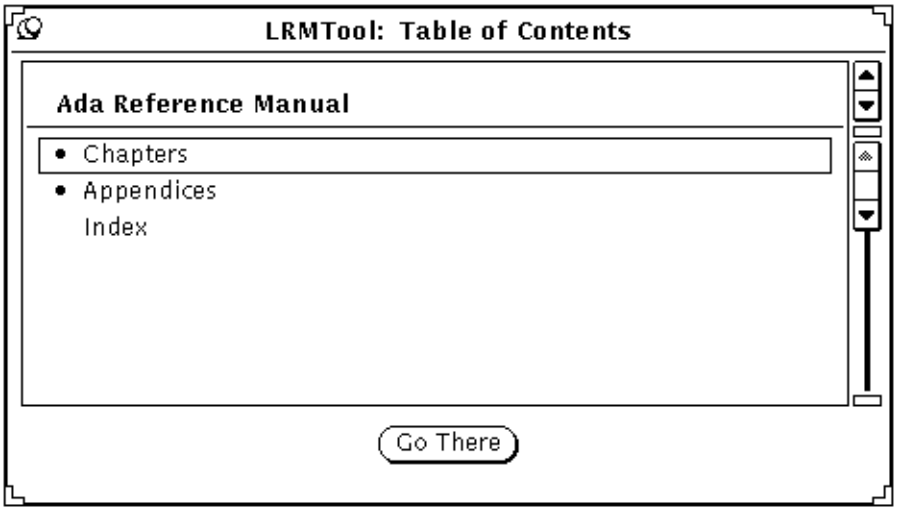
The Table of Contents pop-up window displays the table of contents in an OPEN LOOK control called a *hierarchical scrolling list*. Hierarchical scrolling lists work like regular scrolling lists, except that they handle nested lists like a table of contents with entries, subentries, sub-subentries, and so on. Hierarchical scrolling lists are compact; they show you entries at one level only and represent with markers the level in the hierarchy you are looking at.

To view the Table of Contents hierarchical scrolling list:

- **Choose Table of Contents from the View menu.**

Here is the table of contents at the top level as represented in a hierarchical scrolling list:

A bullet beside an entry means that the entry has subdivisions



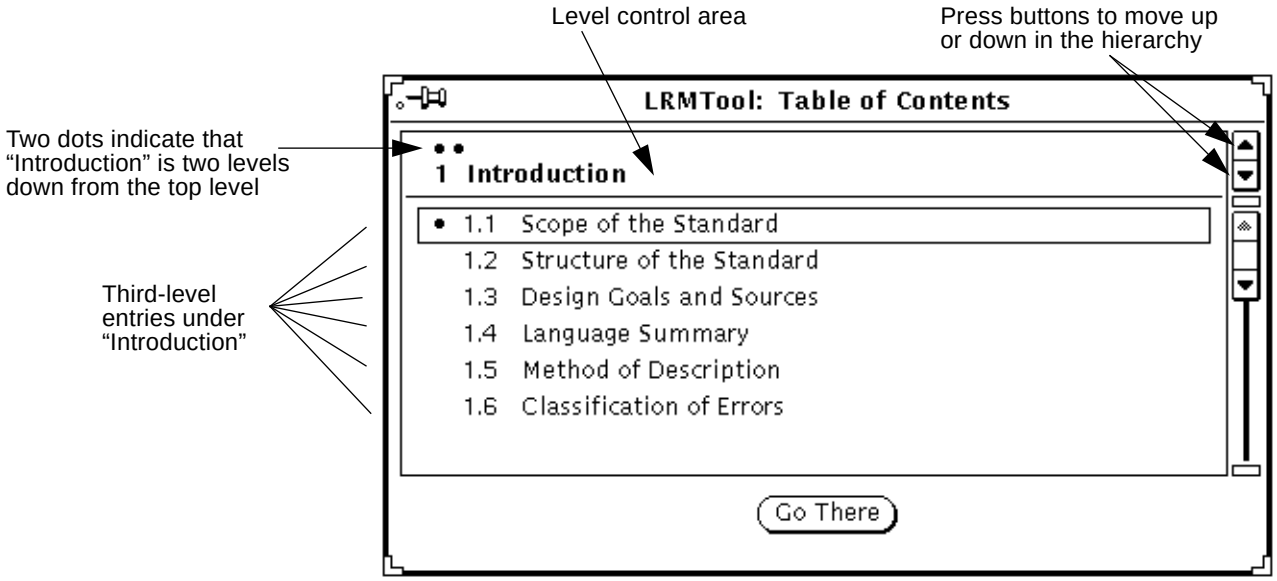
To use the hierarchical scrolling list:

- **Double-click SELECT on an entry listed.**
  If you double-click SELECT on an entry that has subdivisions (like Chapters, in the above example), then LRMTool displays the subentries section of the selection. You can then choose a subentry and either double-click SELECT or press the Go There button to have LRMTool display the associated text.

If you double-click SELECT on an entry that does not have subdivisions (like Index in the above example), or if you select that entry and press the Go There button, LRMTool displays the associated text in the bottom pane.

A *bullet* (•) to the left of an entry indicates that the entry has subdivisions. To view the subdivision entries, double-click SELECT on an entry.

Here is how the list looks when you double-click SELECT on Chapters and then double-click SELECT again on Introduction to reach the third level down.



Notice the two bullets above "1 Introduction" in the Level Control area. These bullets indicate that the Introduction is two levels down from the top of the hierarchy.

To move up and down the hierarchy:

♦ **Press the Up or Down button in the level control area.**

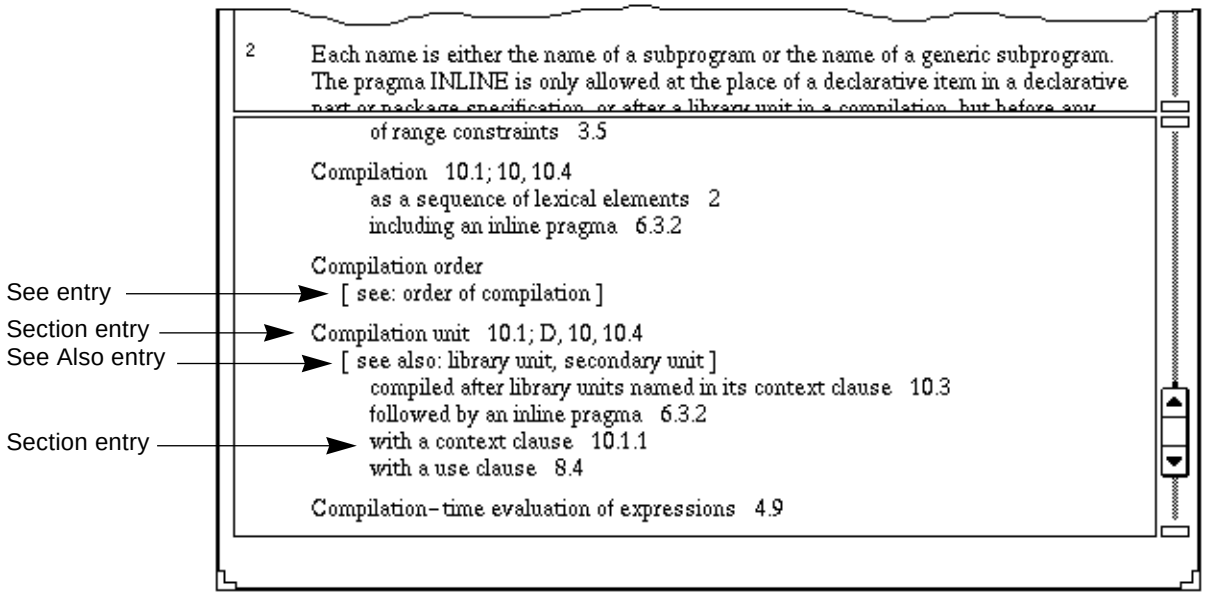
7.4.5 Index

The LRM has a large, thorough index. All technical terms used in the manual are listed in the index.

The three types of index entries are:

- *Section* entries 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 [] contain words or phrases that the index references using different terms or the same terms in a different order. (See the example in the following figure.)
- *See also* entries also in brackets [] alert you to other index entries that refer to sections with related information.

The following example illustrates the three types of index entries.



7.4.5.1 Index Entry Links

The index supports two types of links: *section number* links and *word* or *phrase* links. Section number links display sections; word or phrase links display the index entry for the word or phrase itself.

7.4.5.2 Referenced Section

To display a section referenced in an index or reference entry, use one of the following ways:

- Double-click SELECT on a section number to display the section in the active pane.
- Highlight a section number by clicking SELECT on it once, then drag and drop the link box to another display pane, where LRMTool will display the section. “Displaying Multiple Views of the Manual” on page 7-6 explains this functionality.

7.4.5.3 Index Entries

To display the part of the index containing the entry associated with a word or phrase reference or index link:

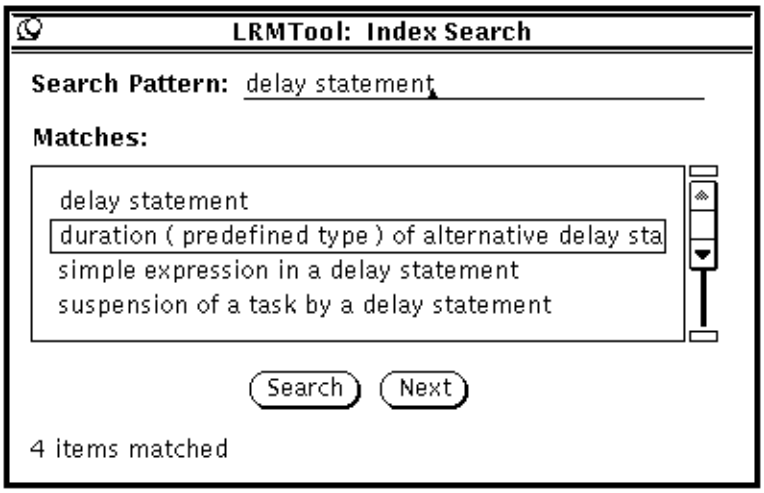
1. **Double-click SELECT on a word or phrase link to display the index entry in the active pane.**
2. **Select the word or phrase link by clicking SELECT on it once, then drag and drop the link-box to another display pane.**
   There, LRMTool displays the part of the index containing the selected link.

7.4.6 Index Entry Searches

You can activate the Index Search pop-up window by choosing Index Search from the View menu. If you enter a word or phrase in the Search Pattern text field and press the Search button or the Return key accelerator, LRMTool searches for all of the index references containing that word or phrase.

The window lists the entries found in a scrolling list. Click SELECT on an entry, and LRMTool scrolls the manual to the location in the index where that index entry occurs. Press the Next button to go to the index location of the next entry on the scrolling list.

The search is *not* case-sensitive.



7.4.7 Reference Links

The LRM lists references in separate paragraphs at the end of many sections and subsections. LRMTool displays these references in *italic* type, as the manual does.

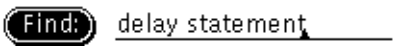
LRMTool treats all of these references as links. Note, however, that LRMTool does not handle isolated references appearing in regular text paragraphs as links. References that contain links are those listed in separate paragraphs that start with *References.*

Reference entries consist of two parts: a word or phrase followed by a section number. LRMTool treats each part as a different type of link: a word or phrase link and a section number link. These two types of links behave exactly as their counterpart index entry links; that is, activated word or phrase links display index entries, and activated number links display the section. A few reference words or phrases cannot be looked up. When you select one that cannot be looked up, it is highlighted as normal selected text, instead of having a box around it.

Activate reference links the same way you do index links, either by using the double-click accelerator or by selecting them and then dragging the link-box to a different pane from the one in which you selected the link. For details, see the discussion of links in “Links” on page 7-4, or “Index” on page 7-12.

7.4.8 Find Facility

You can put the Find facility to work for you in several ways. It locates the first occurrence of any word, phrase, or number you enter in its editable text entry field.



Find accepts any string; 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 (but see the exceptions discussed in the following section about using Find to display sections or appendices).**
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.**

7.4.8.1 Finding Sections by Numbers and Appendix Letters

You can also use Find to display a section or appendix by entering its number or appendix letter. This slight modification of the basic Find functionality makes it easier for you to move around in the LRM.

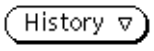
Find interprets an entry that corresponds to a section number or appendix letter (uppercase A through F) as a request to display the corresponding section or appendix rather than to find the first occurrence of that number or letter embedded in the text. For example, if you Find “A,” LRMTool displays Appendix A, not the next occurrence of that uppercase character. The same applies to section numbers. If you Find “9.2,” LRMTool displays section 9.2, not the next occurrence of that number, which may be in a reference paragraph.

To see all the places where the number 9.2 appears as a reference, use Search Index or the OpenWindows Find function key. (See the next section for additional information.)

7.4.8.2 LRMTool Find Button Versus OpenWindows Find Key

The Find button (and its Return key accelerator) in LRMTool operates independently of the Find facility provided by the OpenWindows Find function key. However, since you can use the OpenWindows selection service to select any text in an LRMTool display pane, you can also use the Find function key to locate words or phrases appearing in the text.

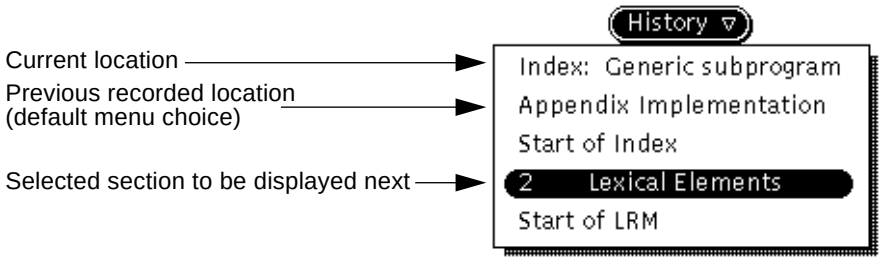
7.4.9 History Facility



The History facility lists the eight most recently displayed sections as items on the History menu, including the currently displayed section.

To redisplay a section appearing on the History menu:

♦ **Choose the item from the list.**



History stacks items on the list from the top; the top item listed is always the currently displayed section. The second item lists the most recently displayed section, which is also the default choice. Thus, you can redisplay the last section you viewed prior to the currently displayed section by simply pressing the History button.

7.4.9.1 Recorded History

History enters the item on the History menu when you use one of the following methods to display a section:

- Choosing from the table of contents
- Activating a reference or index link
- Scrolling events

7.4.9.2 Unrecorded History

History does *not* add items to the menu stack for sections you display when you use the following display methods:

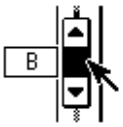
- Scrolling using the Up and Down arrows without stopping
- Finding expressions or sections with the Find facility
- Displaying sections from the Index Search pop-up window
- Displaying sections using the History facility

7.4.9.3 History across Multiple Views

History is blind to the number of views of the manual you have open. It maintains one stack of menu items only. The single list applies across views.

If you have several views open, History redisplays the section in the active pane when you choose an item from the History menu.

7.4.10 Scrolling



Because LRMTool implements the on-line LRM as one continuous document, you can scroll through the manual from beginning to end, stopping anywhere you like. LRMTool supports a location-sensitive scrollbar elevator. As you move the drag box up or down the scrollbar cable, the scrollbar elevator reports the location to the nearest top-level division (chapter number, appendix letter, or index). For example, in the figure shown in the margin, the scrollbar elevator is positioned alongside Appendix B.

Using EditTool 8

In AdaVision, when you open a unit for editing, SPARCworks/Ada displays the source file to which the unit belongs in an EditTool window—unless you have changed the default editor using the Editor control in the AdaVision Properties window. EditTool provides all of the functionality of the standard Text Editor, and it works in the same way as Text Editor.

In addition to providing standard editor functionality, EditTool also helps you correct errors detected by the SPARCompiler Ada compiler. When a compile job fails, you can press the EditTool button in the Compile Status window to start the editing process.

When you do so, EditTool does the following:

- Opens the source file containing the error(s)
- Positions the cursor at the location of the first error
- Highlights the first error
- Displays the error message in the EditTool footer

This chapter describes how to use EditTool. EditTool is the AdaVision default editor. It is a specialized version of the OpenWindows standard editor, Text Editor. EditTool runs in a separate OpenWindows base window.

The description of EditTool is organized into the following sections:

8

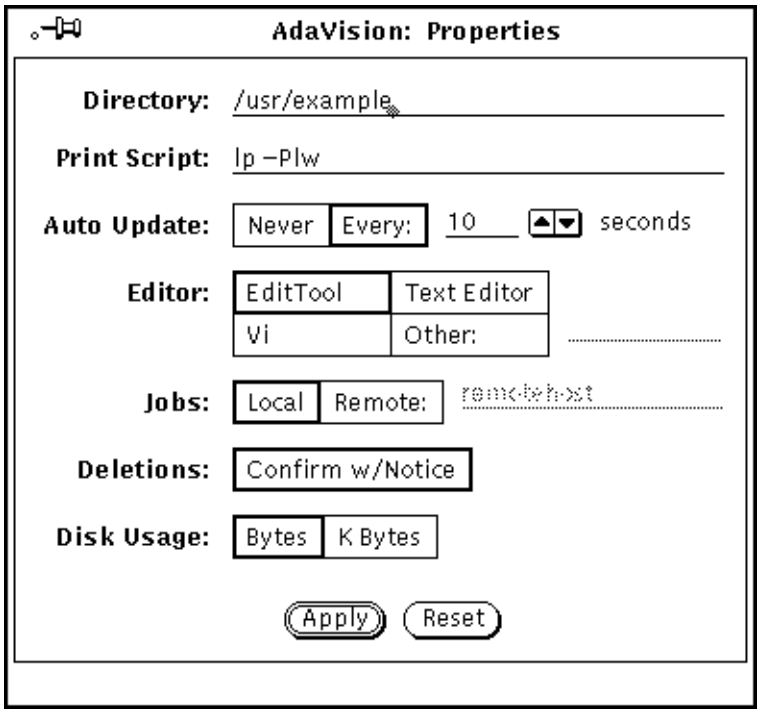
8.1 Starting EditTool

Initially, EditTool is the default AdaVision editor. The instructions in “Five Ways to Start EditTool” on page 8-3 assume that it is still the default editor. Before learning how to start EditTool, you are shown in the following section how to reset the Editor control to EditTool.

8.1.1 EditTool as the Default Editor Setting

To make EditTool the AdaVision default editor:

1. **Choose AdaVision from the Props menu (any mode) to open the AdaVision Properties window.**



2. **In the Editor control, click SELECT on the EditTool option.**

3. **Press Apply.**

8.1.2 Five Ways to Start EditTool

All of the methods for starting EditTool provide access to the compiler-error correction features. You can start EditTool by:

- Entering the EditTool startup command in a Command or Shell Tool
- Opening an AdaVision LU or CU
- Choosing EditTool from the AdaVision Commands menu
- Pressing the EditTool button in the Make, Compile, or Import Status window of a job that failed to compile successfully
- Double-clicking the EditTool icon on the SPARCworks Manager palette or dragging and dropping the icon onto the workspace

With EditTool running, you can drag and drop an LU or CU icon from the AdaVision display pane onto the EditTool drop target. EditTool replaces the currently displayed file with the file you drop onto the drop target.

8.1.2.1 From a Command or Shell Tool

To open EditTool from a Command or Shell Tool, at the system prompt, type:

```
edittool &
```

Note that this command only opens EditTool with a blank display pane. By supplying an argument, such as *file_name*`.a`, after the command, you can open the source file for browsing or editing. To access the source file containing the errors after a compile job has failed, add the extension `.err` to the file name, as in:

`edittool` *file_name*`.err` **`&`**

Alternatively, with EditTool running, you can drag and drop the `.err` file icon from File Manager onto the drop target, hence loading the file without using the command line. Do not drag and drop the `.err` file onto the EditTool display pane; doing so only loads the `.err` file text without the functionality for error correction that is described in “Correcting Compilation Errors” on page 8-5.

8.1.2.2 From AdaVision

To view or edit an Ada LU or CU:

1. **Click SELECT on the LU or CU icon.**
2. **Select a unit icon in display pane first.**
3. **Choose Open Lib Unit (LU mode) or Open (CU mode) from the AdaVision File menu.**
   Double-clicking SELECT on a unit icon (in icon display mode) or on the unit name (in list display mode) acts as an accelerator to open the unit in AdaVision.

Alternatively, you can choose EditTool from the AdaVision Commands menu to start an EditTool session.

This EditTool item is convenient when you want to start an edit session without loading a source file into the display pane. If no LU or CU icon is selected when you choose Edit, then AdaVision starts an EditTool session with a blank display pane. You can then load any file the same way as you would using Text Editor, or you can drag and drop units into it.

8.1.2.3 From the Job Status Windows

For use when compiling, AdaVision has an EditTool button in the Compile, Make, and Import Status windows. If a compilation terminates unsuccessfully, the EditTool button in any of these windows becomes active. Press it to launch EditTool.

8.1.2.4 From the SPARCworks Manager Palette

As soon as SPARCworks Manager is invoked, the EditTool icon is one of those displayed in the palette.

- **Double-click SELECT on the EditTool icon to launch the tool, or drag and drop the icon onto the workspace.**

“SPARCworks Manager” on page 1-2 describes this interface.

8.2 Correcting Compilation Errors

EditTool differs from Text Editor only in that it has a compilation-error correction tool, which Text Editor does not.

When you start EditTool after an unsuccessful compilation, EditTool does the following:

- Opens with the file containing the error(s)
- Positions the cursor at the location of the first error
- Highlights the first error

Note – Depending on the type of problem, the highlight may not be on the exact spot in the source code that is in error. The error is usually nearby, often the line above the one indicated.

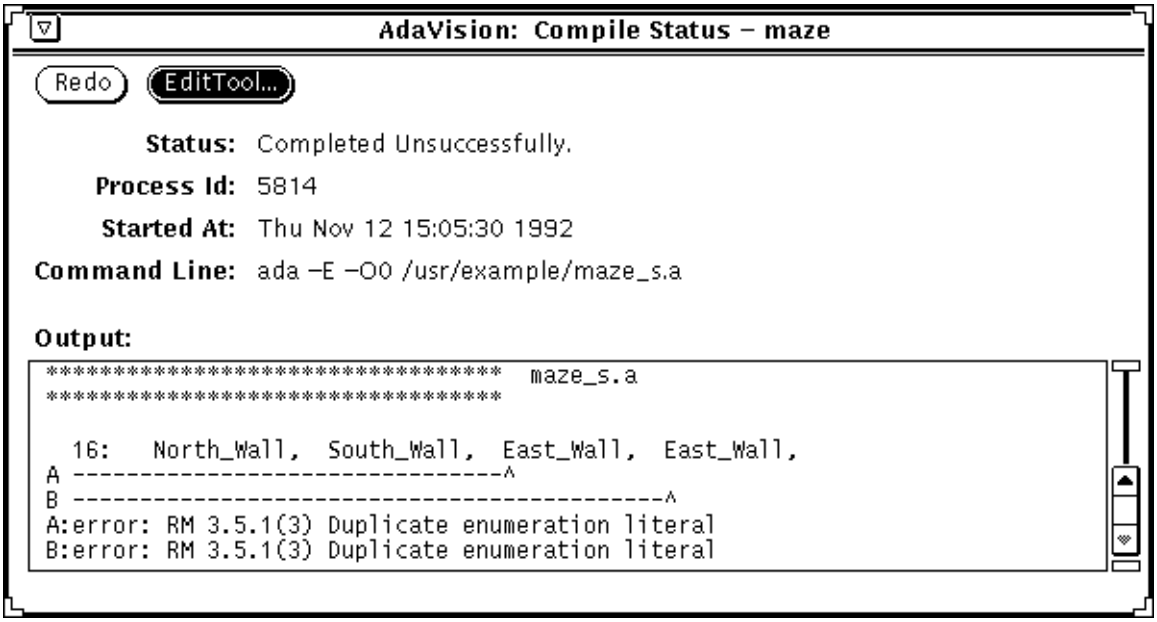
- Displays the error message in the EditTool footer

8.2.1 Example of Error Correction

Here is an example of fixing errors reported by the compiler.

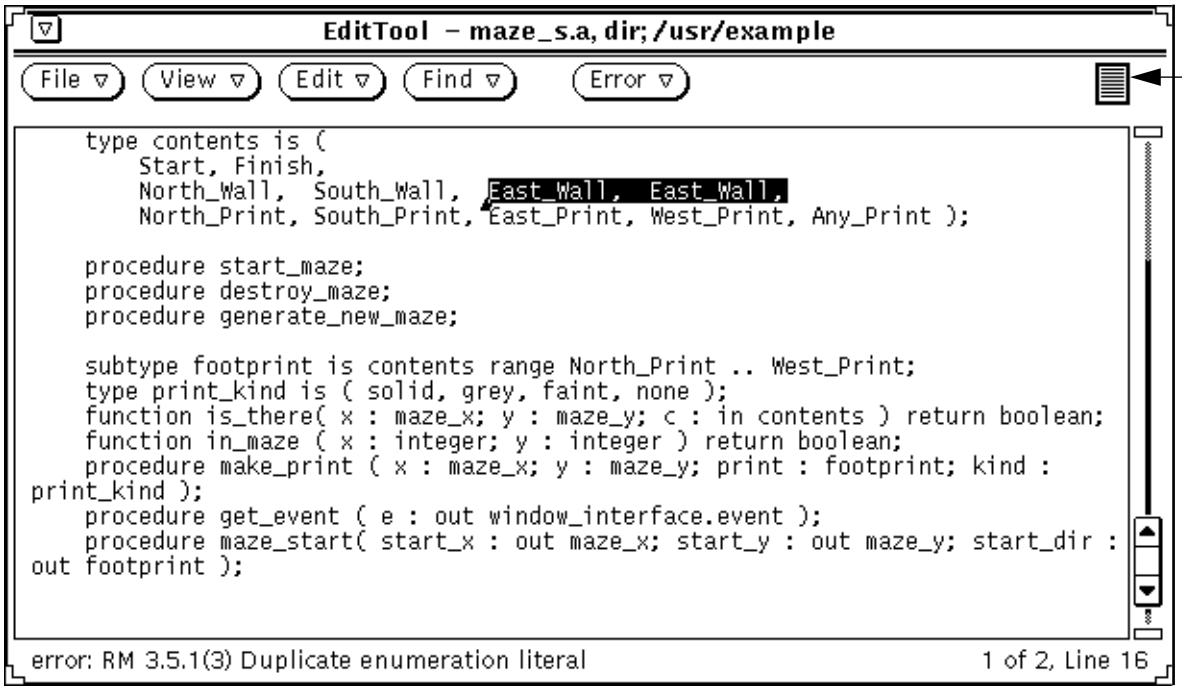
A trivial error has been introduced into the `maze` specification unit. Then a compile is attempted. The compile job icon (shown in the margin) indicates that the compile terminated unsuccessfully.

The Compile Status window is opened by double-clicking SELECT on the job icon. The window reports the error in the Output pane, as shown in the following figure.



To start EditTool from a Compile Status window after a failed compile:

- **Press the EditTool button.**
  AdaVision then starts EditTool in a separate window, as follows.



To correct the error, simply edit the line—in this case, replacing the extra string `East_Wall` with `West_Wall`.

Once the error is corrected, save the file in the usual manner by choosing Save Current File from the File menu.

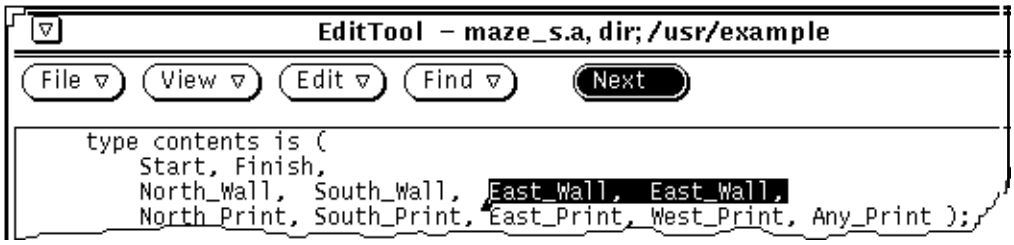
The unit can now be compiled again by pressing the Redo button in the Job Information window. This time, the unit compiles successfully.

8.2.2 Error Menu

EditTool has a special button, Error, with a pull-down menu consisting of four items: Next Error, Next File, LRM Citation, and Error Browser.

8.2.2.1 Next Error

When you click SELECT on the Error button, you choose the default item, Next Error. When you have inserted or deleted new lines of text to correct an error, EditTool is capable of finding the next error further along in the file. Next Error scrolls the text to the next error reported by the compiler *in the currently displayed file* and highlights the error. Using Next Error, you can cycle through all of the errors detected in any given compilation.



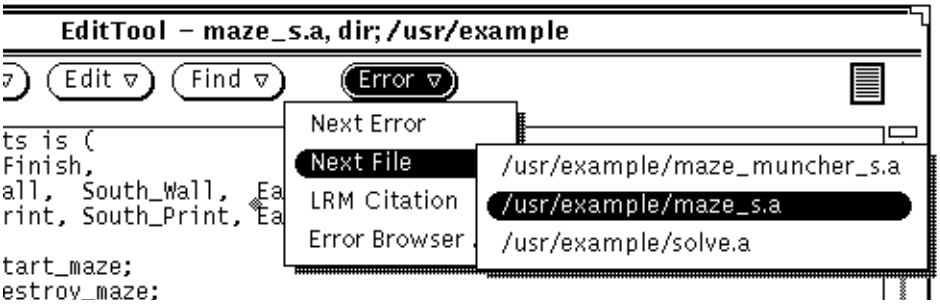
EditTool cycles from error to error within the first file in which the compiler reported errors, returning to the first one. To have EditTool locate and display errors in other files, choose Next File.

8.2.2.2 Next File

The Next File item on the Error menu has a submenu that lists the files containing reported errors.

To see errors in other files:

1. Choose the name of the file you want EditTool to display.



2. **After the file has been loaded, choose Next Error to cycle through the error(s) in the newly displayed file.**

8.2.2.3 LRM Citation

In many instances, compilation errors are due to incorrect semantics, and so on, that are explained in the *Reference Manual for the Ada Programming Language*—known informally as the Language Reference Manual (LRM). For these errors, the Ada compiler prints in the footer an error message that contains a reference to the LRM. This reference points to one of the following:

- A chapter, as in `RM 4`
- A section, as in `RM 4.3`
- A subsection, as in `RM 4.3.2`
- A paragraph, as in `RM 4.3.2(1)`
- An appendix, as in `RM A`

When you see such a reference in the error message, you can elect to browse the LRM text on-line. Do the following:

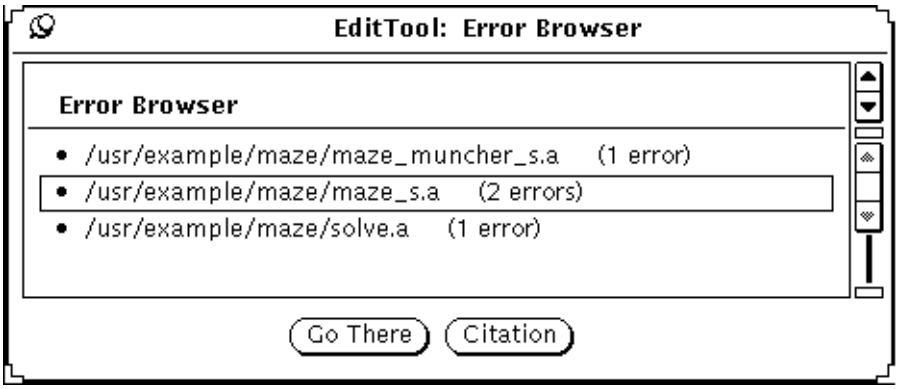
♦ **Pull down on the Error menu and choose LRM Citation.**
If it is not already open, LRMTool is then launched. The text referred to in the error message is displayed in the LRMTool text pane.

The Error ⟹ LRM Citation menu item works only if the error message contains a reference to the LRM. Otherwise, choosing this item produces an error message from EditTool that is displayed in the footer.

Note – If you scroll along the EditTool display pane and choose Error ⟹ LRM Citation, LRMTool displays the citation as referenced in the error message. EditTool does not reposition its display pane to the error in question.

8.2.2.4 Error Browser

Choosing Error Browser from the Error menu opens a pop-up window that lists the full path name of each file with compilation errors. It also tells you how many errors are in each file in parentheses, next to the path name.



Opening Source Files Containing Errors in EditTool

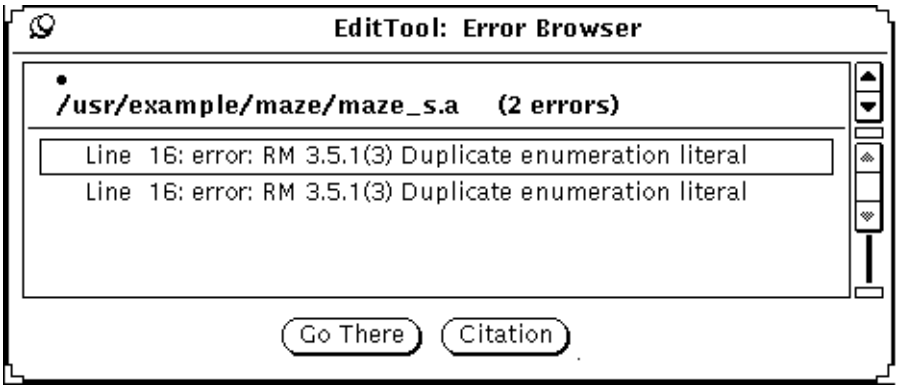
There are two ways to load a file containing compilation errors in EditTool.

The first way is by using the Go There button.

1. **Click SELECT on the path name of the file to be loaded.**
2. **Press the Go There button.**
   The source file is then loaded onto the EditTool display pane, and the first error highlighted. The path name information in the Error Browser display pane remains unchanged.

Alternatively, you can:

1. **Double-click SELECT on the path name of the file to be loaded.**
   The path name is then moved to the title area of the scrolling list. A list of compilation errors in that file is displayed in the Error Browser display pane, replacing the path names of the files containing errors, as shown in the following figure.



2. **Either click SELECT on an error entry and then press the Go There button, or double-click SELECT on the error entry.**
   The source file containing the error is then loaded onto the EditTool display pane with the error highlighted.

Browsing the LRM Citation

The Citation button in the Error Browser provides another avenue for browsing LRM citations that pertain to compilation errors.

If an error entry contains an LRM citation, you can click SELECT on that entry, then press the Citation button to invoke LRMTool (if it is not already open). The referenced LRM text is then displayed on screen.

EditTool prints an error message on the footer when the Citation button is pressed in either of the following conditions:

- The selected error entry does not contain an LRM citation.
- The Error Browser is displaying path names of the files that contain errors, *not* the error entries themselves. An example is shown in the graphic on page 8-10.

8

Implementation Notes A

This appendix describes some details of implementation. The advanced user may be curious, and this information should prove helpful if a problem occurs.

This appendix consists of the following section:

A.1 File called `.options`

The AdaVision property windows and SPARCompiler Ada tools use a `.options` file to remember command-line options, such as those issued to compile and link each unit of an Ada library. For example, the name of an executable is not maintained in the Ada library or anywhere else; it exists only in the `-o` option specified to the Ada linker.

The `.options` file serves other purposes as well:

- The format of a compilation command in the `.options` file is:

  `c:`*source_file_name*`:ada` *commandline*`:`

- The format of a link command is:

  `l:`*unit_name*`:a.ld` *commandline*`:`

`The` colons serve as field separators. So an example file might be:

```
c:hello.a:ada -O2 $ARG:
l:hello:a.ld -o hello $ARG:
```

The format of a default command, or of the user options stored in the file `$HOME/.options.user`, is the same, but with a null argument field. For example:

```
c::ada -E:
```

Glossary

Active pane

The last pane in a window that the user clicked in.

ADAPATH

An Ada library search list associated with each SPARCompiler Ada library. If an Ada unit is not found in the current Ada library, then the Ada libraries named in the ADAPATH will be searched in order until the unit is found.

AdaDebug

A SPARCworks/Ada tool that provides an object-based and window-based interface to `a.db`, the SPARCompiler Ada debugger.

AdaVision

A SPARCworks/Ada tool that provides a graphical interface for the SPARCompiler Ada toolset. AdaVision functions as a library browser and manager, as well as a workplace for editing, compiling, and debugging Ada units and programs.

Automounter

The `automount` program that enables users to mount and unmount remote directories on an as-needed basis.

AutoRead

An AdaVision feature that updates the visual representation of an Ada library at regular intervals.

Body

An Ada construct that describes the implementation of a subprogram, package, or task.

Call stack

The current state of all active subprograms that AdaDebug displays in a Stack Trace window during debugging.

Command line

An editable text field in the AdaDebug Program View window, where the user enters an Ada debugger command.

Compilation unit

A program unit, displayed as a graphical icon in AdaVision, which is compiled as an independent text. It may be the specification or body of a subprogram or package, including generic units and subunits. (*See also* **Library unit**.)

Compilation unit (CU) mode

The mode of AdaVision where Ada compilation units for a selected library are represented as separate objects and displayed in icon, list, or tree mode. The user browses and manages compilation units in this mode, including executing make library, compile, tags, and search jobs.

Cross-development

The process of using 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.

EditTool

A specialized version of the OpenWindows Text Editor, which includes a window-based application of the Ada `a.error` facility.

Generic unit

A template for either a subprogram or package.

Job Status window

A separate window where AdaVision executes an import, make or make library, compile, link, or tags job. A delete job in library unit or compilation unit mode also runs in such a window, provided that the Verbose control is set to Echo in the Delete Properties window.

Library mode

The mode of AdaVision where Ada library names are displayed. In this mode, the user examines and manages Ada libraries, as well as sets certain AdaVision defaults, including library directives and the ADAPATH.

Library unit

A compilation unit that is a specification, not a body. It is part of an Ada library, displayed as a graphical icon in AdaVision. (*See also* **Compilation unit**.)

Library unit (LU) mode

The mode of AdaVision where Ada library units for a selected library are represented as separate objects and displayed in either icon or list mode. The user browses and manages library units in this mode, including executing make, compile, and link jobs.

LRMTool

An on-line version of the *Ada Language Reference Manual*.

Package

An Ada construct consisting of a specification and an optional body that can be separately compiled.

Program

A main subprogram compilation unit and the collection of compilation units upon which it depends.

Program executable

The result of linking a main subprogram.

Program View window

An AdaDebug display pane where the user browses and debugs the source code of a program.

Remote display

The process of running SPARCworks/Ada on a remote SPARC system and directing the windows display to a local machine.

Remote execution

The process of running AdaVision on a local machine while submitting Ada jobs to a designated compile and debug server.

Spec

The specification portion of a compilation unit which contains visible declarations.

Stack Trace window

The pop-up window where AdaDebug displays a call stack.

Subprogram

An executable Ada construct, that is, a procedure or a function.

Subunit

A portion of a body or another subunit that can be separately compiled.

Tags

The AdaVision version of the Ada `a.tags` tool, which provides the capability to cross-reference Ada units and types.

Task

An Ada construct that may operate in parallel with other tasks.

Task Browser

An AdaDebug display pane that presents a global view of program active tasks.

Task Status window

An AdaDebug window that displays information about a selected task during debugging.

Task View window

An AdaDebug display pane where the user browses and debugs the source code and call stack of a single task in a multitasking program.

Index

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W