Building Programs with MakeTool

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

Part No.: 801-5095-10
Revision A, December 1993

Contents

Preface

This manual describes how to use MakeTool to compile source code and browse makefiles. MakeTool is an integrated component of the SPARCworks™ toolset, which also includes:

- Analyzer
- Debugger
- FileMerge
- Manager
- SourceBrowser

Before You Begin

This manual is written for software developers who write and test programs coded in text (ASCII) source, including ANSI C, FORTRAN, Pascal, C++, and Assembler.

This manual assumes you are familiar with

- Sun® operating system commands and concepts
- The OPEN LOOK® interface and the OpenWindows™ environment, particularly the use of the mouse to activate a window, select text, and click on buttons

If you are not familiar with the OPEN LOOK interface, see Chapter 4, "The OPEN LOOK GUI," in *Managing SPARCworks Tools*. For more information on the OPEN LOOK GUI, see the *OPEN LOOK Application Style Guidelines*.

For more information on the OpenWindows environment, see the *OpenWindows Developer's Guide 3.0.1 User's Guide*.

Operating Environment

MakeTool runs under the Solaris® 2.x operating environment.

Solaris 2.x implies:

- Solaris 2.2 or later
- SunOS™ 5.2 (or later) operating system
- A SPARC® computer (either a server or a workstation)
- The OpenWindows 3.x application development platform

The SunOS 5.x operating environment is based on the System V Release 4 (SVR4) UNIX[1] operating system, and the ONC™ family of published networking protocols and distributed services.

How This Book Is Organized

This manual is organized as follows:

Chapter 1, "Introduction to MakeTool," presents an overview of MakeTool.

Chapter 2, "Building Programs," describes how to start MakeTool and load makefiles.

Chapter 3, "Browsing Makefiles," describes how to browse makefiles and evaluate makefile rules and macros.

Appendix A, "Troubleshooting," describes how to proceed if MakeTool fails or produces errors.

1. UNIX is a registered trademark of UNIX System Laboratories, Inc.

What Typographic Changes and Symbols Mean

The following table describes the typographic conventions and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The 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.
♦	A single-step procedure	♦ **Click on the Apply button.**

Code samples are included in boxes and may display the following:

%	UNIX C shell prompt	`system%`
$	UNIX Bourne and Korn shell prompt	`system$`
#	Superuser prompt, all shells	`system#`

How to Get Help

SPARCworks tools include the following on-line help facilities:

- **AnswerBook® system** displays all SPARCworks tools manuals. You can read this manual on line and take advantage of dynamically linked headings and cross-references.

  To start the AnswerBook system, type: `answerbook`

- **Magnify Help**™ messages are a standard feature of the OpenWindows software environment. If you have a question, place the pointer on the window, menu, or menu button and press the Help key.
- **Notices** are a standard feature of OPEN LOOK. Some notices inquire about whether or not you want to continue with an action. Others provide information about the end result of an action and appear only when the end result of the action is irreversible.
- **SunOS Manual Pages** (man pages) provide information about the command-line utilities of the SunOS operating system. Each tool has at least one man page. To access the man page for MakeTool, type:

  ```
  man maketool
  ```

Related Documentation

This manual is part of the SPARCworks document set. Other manuals in this set include:

- *Installing SunPro Software on Solaris*
- *SPARCworks Tutorial*
- *Browsing Source Code*
- *Debugging a Program*
- *Managing SPARCworks Tools*
- *Merging Source Files*
- *Performance Tuning an Application*

You can find these and other related documents in the on-line AnswerBook system. AnswerBook products available for the SPARCworks 3.0 release are:

- **SPARCworks documents —** *SPARCworks 3.0 AnswerBook*
- **Common tools document**s **—** *SPARCworks/SPARCompiler 3.0 Common Tools & Related Material AnswerBook*
- **Installation guide —** *SPARCworks/SPARCompiler 3.0 Installation AnswerBook*

To access the AnswerBook system, refer to *Installing SunPro Software on Solaris*.

Introduction to MakeTool 1

MakeTool is an OpenWindows interface to `make`, the SunOS utility that oversees program compilation and ensures that programs are compiled from the newest sources. In addition, MakeTool contains a browser that helps you interpret makefiles by expanding macros and rules so that you can understand them more easily.

This chapter is organized into the following major sections:

1.1 How `make` *Works*

`make` initiates a program build by reading a *makefile* for the program and executing the commands it finds there. The makefile, written by a software developer or project administrator, usually describes explicitly how to build each module of the program as well as the final executable.

1.1.1 Some `make` *Basics*

Although a complete discussion of `make` is beyond the scope of this manual (you can type `man make` to access the man page), a few characteristics of `make` and makefiles are necessary in order to understand MakeTool.

Makefile Search Algorithm

`make` searches the current working directory for a makefile with the name `makefile`. If no `makefile` exists, `make` searches for a file named `Makefile`. MakeTool uses this same algorithm to search for makefiles. If your makefile has a different name, you can load it into MakeTool after it has started.

Depending on a project's organization, a makefile may include other makefiles and may direct `make` to descend into subordinate directories and use makefiles it finds there.

Target

A *target* is a module (usually a file) that is to be constructed by `make`. It may be the result of a long series of compilations (a *high-level* target, such as an executable binary) or an intermediate result (a *secondary* target, such as an object file with a `.o` suffix).

Some targets cause a series of commands to be executed but do not actually build a module. For example, the `clean` target usually removes all derived files in a directory.

Dependency

A *dependency* is the relationship between a target and a module that `make` uses to build it. Common usage has extended the meaning to identify the module itself in addition to the relationship between it and its target. Each target is typically built from several dependencies.

New `make` users are sometimes confused when they hear *dependency* being used to identify what is (mathematically speaking) an independent variable: the target depends on the modules from which it is built, not the other way around. Such confusion is entirely justified, but any attempt to redefine the term more rationally would produce even greater confusion among experienced `make` users.

Rule

A *rule* describes how to build a target from a list of dependencies. Typically, a rule is a command line in a makefile that invokes a compiler or linker. `make` also maintains a list of implicit rules so that compilations can be initiated even

when no makefile exists. For example, an implicit rule invokes the C compiler to build an intermediate object (`.o`) file from any source file that has been named with the `.c` suffix convention.

Macro

A makefile *macro* is a variable that is used in dependency lists and rules. Makefiles that employ macros are usually more flexible and more easily maintained than makefiles without macros.

In makefiles, any line of the form `STRING1 = string2` is a macro definition. `STRING1` is the name of the macro, and `string2` is the definition of the macro. For example, consider the following macro definition:

```
OBJECTS = block.o hand.o block_draw.o
```

Following this definition, whenever `make` encounters the string `$(OBJECTS)` in the makefile, it expands the string to the value `block.o hand.o block_draw.o`.

1.2 Graphical Overview

This section explains the controls and features of the MakeTool graphical user interface.

1.2.1 MakeTool Animated Icons

MakeTool in its closed form is represented by a series of icons, as shown in Figure 1-1 on page 4. The icons cycle through, with each icon representing a different condition of a `make` build process. By looking at the icon, you can tell at a glance whether or not a long `make` task is finished. The cycling of the icons produces an *animation* effect.

- The first icon indicates that a `make` has successfully completed. This icon is also displayed before the first `make` of a session.
- The second icon indicates that `make` has been started but is not yet finished. This icon is animated: It shows source files being fed into a “`make` machine” and rolling out on a conveyor belt as compiled objects.
- The third icon indicates that `make` has failed due to an error.

Figure 1-1 MakeTool Animated Icons

1.2.2 MakeTool Window

The MakeTool window (Figure 1-2) opens when you start MakeTool. MakeTool attempts to load a makefile automatically using the same search algorithm as `make`—that is, according to the following rules:

- First, MakeTool searches for a file named `makefile` in the directory in which MakeTool was started (the current working directory).
- If MakeTool cannot find `makefile`, it searches for `Makefile` in the same directory.

If MakeTool cannot find either `makefile` or `Makefile`, then you must load the file manually by clicking on Load.

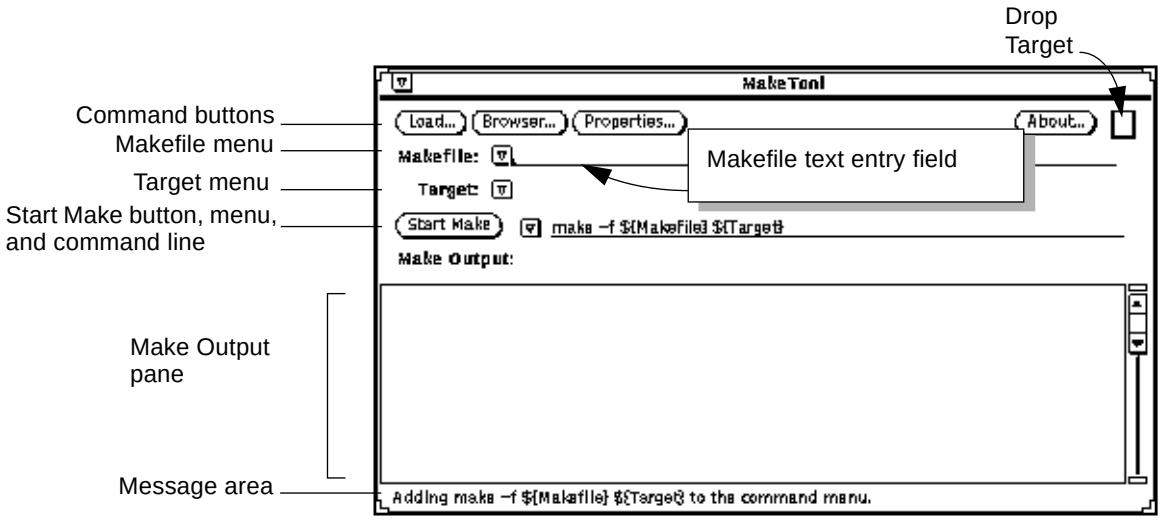


Figure 1-2 MakeTool Base Window

If MakeTool finds a makefile, it loads the makefile. The fully qualified name of the makefile is shown in the Makefile text entry field of the MakeTool window.

Load Button

When you click on Load, MakeTool opens the Makefile Load Window from which you can select and load the makefile you want. See “Loading Makefiles” on page 2-14 for details.

Each time you load a makefile, the name of the makefile is added to the Makefile menu.

Browser Button

The Browser button displays the Makefile Browser window, shown in Figure 1-3 on page 6.

When you select a statement in the Makefile Statements pane, the complete source and an expanded version of that statement are shown in the Statement Source and Expanded Source pane. See Chapter 3, “Browsing Makefiles” for details on how to browse files.

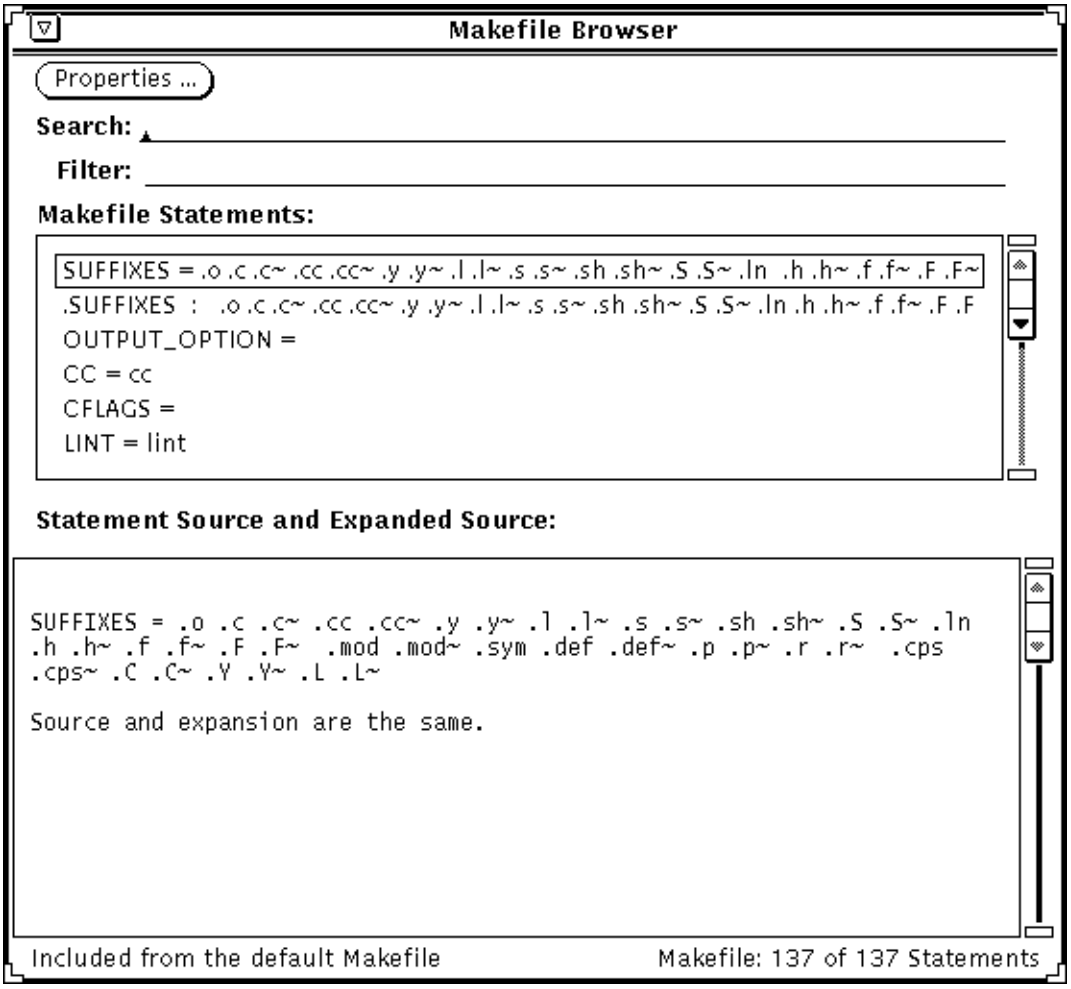


Figure 1-3 Makefile Browser Window

Properties Button

Clicking SELECT on the Properties button brings up the MakeTool Properties window in which you can specify commands to be executed with each `make`. See “Executing Commands in Response to make Events” on page 2-17 for instructions.

MakeTool Properties Window

The MakeTool Properties button opens a pop-up window (shown in Figure 1-4) in which you can specify shell commands that will execute each time one of the following actions occurs:

- `make` starts a build
- `make` fails to build the requested target
- `make` successfully completes a build

Note – Changes to the properties are only in effect during the current session. To save customizations permanently, see Section 2.4.1, “Saving MakeTool Customizations,” on page 2-17.

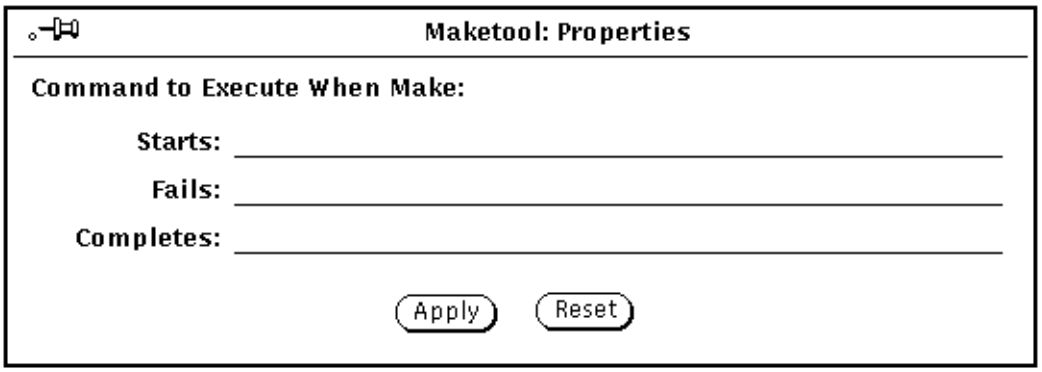


Figure 1-4 MakeTool Properties Pop-Up Window

About Button

The About button displays a pop-up window that gives the version number, copyright, and a brief description of MakeTool (see Figure 1-5 on page 1-8). Clicking on the Comments button displays another window from which you can send comments about the tool to SunPro.

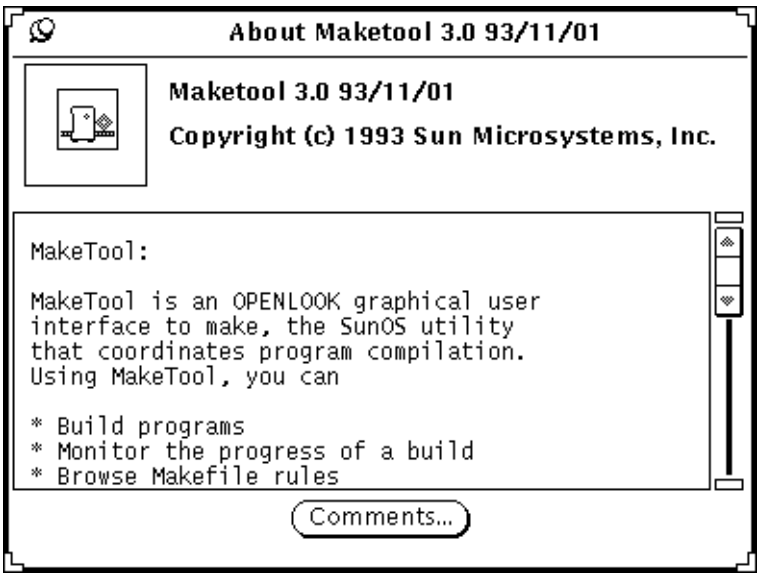


Figure 1-5 About Box

Drop Target

The Drop Target is located in the upper right corner of the MakeTool window (see Figure 1-2 on page 4). You can use it as a quick way to load a makefile. Select a makefile filename from a Command Tool or its icon from the File Manager and drag it over to the Drop Target. If you drop an invalid makefile onto the Drop Target, an error message is displayed in the message area of the window.

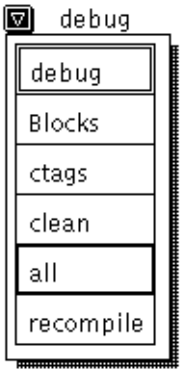
Makefile Menu and Text Entry Field

The Makefile menu lists all the makefiles that have been loaded previously and allows you to switch between makefiles easily.

Initially the Makefile text entry field shows the fully qualified name of a makefile in the working directory. If no file named `Makefile` or `makefile` is in the working directory, the field is blank.

Use the Load button menu to change directories and load different makefiles, as explained in Section 2.2, “Loading Makefiles,” on page 2-14.

Target Menu



Whenever you load a makefile into MakeTool, it examines the makefile, identifies high-level targets, and puts the target names into the menu. MakeTool does not place secondary targets such as intermediate files in the menu because users rarely need to build secondary targets directly. If you need to build a secondary target, type the name of the target in place of the `${Target}` variable in the Start Make command line and click on the Start Make button.

Because MakeTool automatically finds high-level targets and displays them, there is usually no need for you to look at the text of the makefile in order to identify the targets you want to build, as is sometimes the case when you use `make` directly from the command line.

You can edit the length of the Target menu using the following `.Xdefaults` file entry:

```
swMaketool.targetChoice.initialLength: 25
```

The default for the initial length of the target menu is 20 targets.

After you have updated the `.Xdefaults` file, you must execute the following command to make the changes effective:

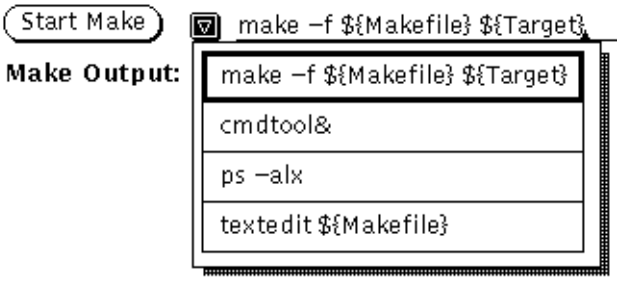
```
xrdb -load ~/.Xdefaults
```

Start Make Button

The Start Make button executes the command shown in the Start Make command line. Usually, the command is a `make` command. Once the Start Make button is activated, it toggles to Stop Make. If you want to stop the build, click on Stop Make. The build aborts and the button toggles back to Start Make.

Start Make Menu and Command Line

The Start Make menu displays the list of commands that you have previously executed, including the default `make` command. To execute a command, select a command from the menu and click on the Start Make button or press Return.



A `make` command template appears in the command line and is provided in the menu. MakeTool provides the variables `${Makefile}` and `${Target}` to the startup command, which correspond to the current makefile and selected target respectively. To build a target that doesn't appear in the target list, simply type the target name in place of the `${Target}` parameter. You can edit this template to include any options to `make` that you need.

MakeTool maintains a history of the commands you enter in the text entry field during a session and keeps them in the Start Make menu. Thereafter, you can quickly reissue a command by choosing from the menu.

To add a command to the menu, type the command (including arguments) in the command line and click on the Start Make button. This action executes the command and adds it to the menu (if the command is new). MakeTool uses the exit status of the command to set the state of the icon, so be sure that any command you issue with the Start Make command line preserves the state of the exit code.

If a subsequent build requires the same `make` arguments, you do not need to retype the command—simply select it from the menu and click on the Start Make button.

You can edit the length of the Start Make menu using the following `.Xdefaults` file entries:

```
swMaketool.make.Cmd: make -f ${Makefile} ${Target}
swMaketool.commandMenu.maxLength: 20
swMaketool.commandMenu.maxItemLength: 40
```

The command `swMaketool.make.Cmd` is the only command that is initially part of the command list. The default is

```
make -f ${Makefile} ${Target}
```

where `${Makefile}` expands to the name of the currently loaded makefile and `${Target}` expands to the currently selected target.

The default for the maximum number of commands for the Start Make menu is 25. The default for the length of a command is the initial width of the command typed in.

After you have updated the `.Xdefaults` file, you must execute the following command to make the changes effective:

```
xrdb -load ~/.Xdefaults
```

Make Output Pane

To execute a command, click on the Start Make button. MakeTool responds by starting a subshell, changing directories to the makefile's working directory (if necessary) and issuing the command in the subshell. All commands MakeTool issues are echoed in the Make Output pane in the MakeTool window.

1

Building Programs 2

This chapter is organized into the following sections:

2.1 Starting MakeTool

You can start Maketool in one of three ways:

- From the SPARCworks Manager
- From the Debugger
- From a command line

MakeTool tries to load a makefile automatically at start-up, or you can load a makefile after MakeTool starts.

From the SPARCworks Manager

You can start Maketool from the Manager by doing one of the following:

- **Double-click on the MakeTool icon.**
- **Drag the MakeTool icon onto the workspace.**

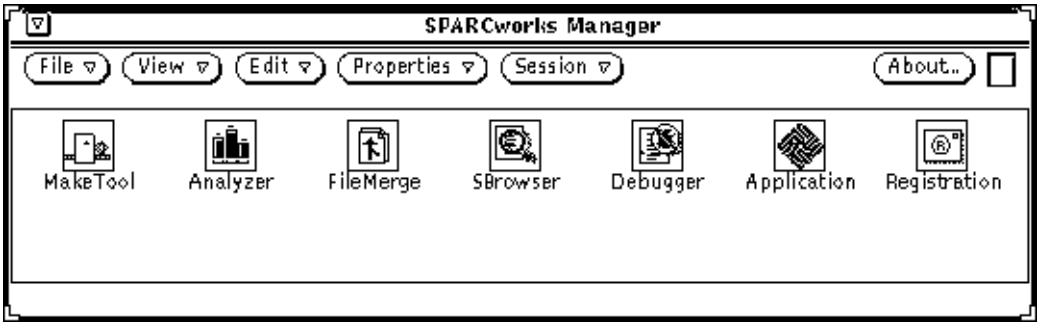


Figure 2-1 SPARCworks Manager Window

From the Command Line

To start MakeTool from the command line:

♦ **Type the following command at a prompt:** `maketool &`

From the Debugger

You can start MakeTool from the Debugger and issue `make` commands from there. You can set the properties to issue the `make` directly or to pass environment and directory information to MakeTool and run the `make` under control of MakeTool.

To start MakeTool from the Debugger window:

♦ **Choose Program ➤ Make ➤ Invoke MakeTool**

2.2 Loading Makefiles

MakeTool attempts to load a makefile automatically upon startup. If no file named `makefile` or `Makefile` exists in the working directory, MakeTool will not load a file. You can load a makefile with a different name from within MakeTool whether or not a makefile has already been loaded.

There are three ways to load a makefile into MakeTool:

From the Load button

When you click on the Load button, the Makefile Load window pops up (Figure 2-2). Double click on a file's icon (or select the icon and then click on the Load button).

If you prefer, you can type the name of a makefile directly into the Name text entry field at the bottom of the window. This file will be loaded when you click on the Load button or press Return after you type in the name. The makefile is added to the list in the Makefile menu.

The window uses the same conventions as the OpenWindows File Manager and provides a way to navigate through the file system.

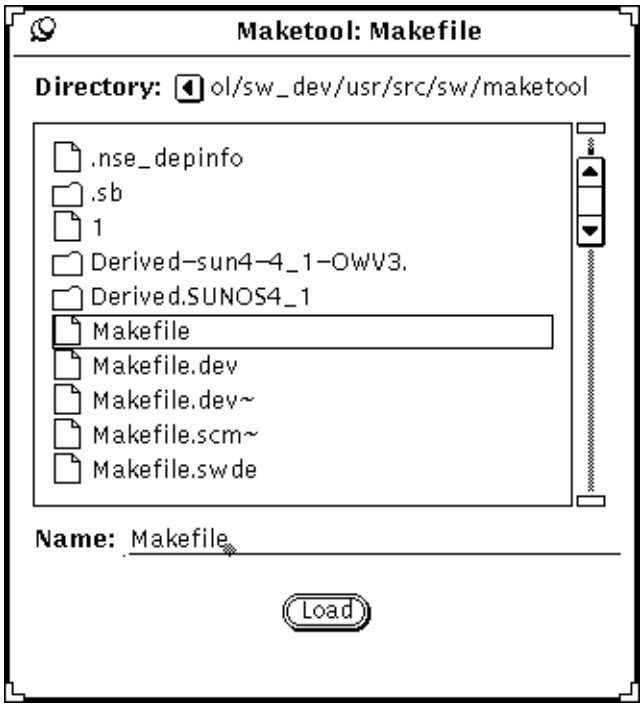


Figure 2-2 MakeTool Load Makefile Window

To descend into a directory, double-click on the directory's icon (the image of a file folder).

To ascend to a parent directory, double click on the first entry in the scrollable pane (the ". ." entry).

From the Drop Target

Select a makefile filename from a Command Tool or its icon from the File Manager and drag it onto the Drop Target. The makefile is loaded and added to the Makefile menu.

From the Makefile text entry field

Type the path name of a makefile in the Makefile text entry field and press Return.

2.3 Executing Commands in MakeTool

To execute a `make` command in MakeTool:

1. **Load the makefile (see "Loading Makefiles" on page 2-14), or select a previously loaded makefile from the Makefile menu.**
2. **Choose the target to build from the Target menu.**
3. **Choose the `make` command template from the Start Make menu.**
   If you want to use a different target or add `make` options, edit the Start Make command line.
4. **Click on Start Make.**
   The results of the command are displayed in the Make Output pane. If you close the MakeTool window, the animated icon shows the state of the `make` command.

To execute any other command in MakeTool:

1. **Type the command into the Start Make command line.**
   You can choose a previously executed command from the Start Make menu.
2. **Click on Start Make.**
   The results of the command are displayed in the Make Output pane. If you close the MakeTool window, the animated icon shows the state of the command.

2.4 Executing Commands in Response to `make` Events

The commands you enter in the pop-up window complement the MakeTool icons and messages, which give visual feedback on the progress of a build. One use of this feature might be to send a mail message to yourself or to another developer when your build has completed successfully (the `mail`(1) command can be used for this purpose). Another use could be to provide audio as well as visual feedback on the state of a `make` build.

1. **Click on the Properties button.**
   The MakeTool Properties window pops up in response.

2. **Enter the command you want to execute for each `make` event in the corresponding text entry field.**
   For example, Figure 2-3 shows the pop-up window with `mail` commands for each of the three events. The arguments to the `mail` commands specify who will receive the messages (in this case, the user wants to be notified when a build starts or fails, and wants other developers (in the alias `dev_group`) to be notified when the build is successful. The files that are input to the `mail` commands contain the messages to be mailed.

3. **Click on Apply.**
   This puts the properties into effect for the rest of the MakeTool session.

2.4.1 Saving MakeTool Customizations

In order for MakeTool to execute these commands as defaults when the tool is started, you must edit the file where properties are stored:

The search order for properties is

1. `$PWD/.maketool-init`
2. `$HOME/.maketool-init`
3. `$HOME/.Xdefaults`

Note – MakeTool executes shell commands with sh (Bourne shell), not csh (C shell). Because sh does not recognize the tilde (~) shorthand for your home directory, you must fully qualify path names in the MakeTool Properties window.

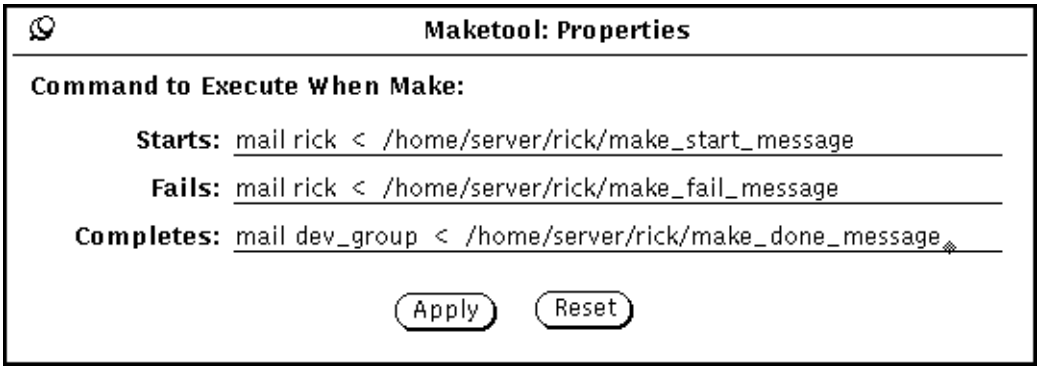


Figure 2-3 MakeTool Properties Window With `mail` Commands

To have MakeTool always execute certain commands in response to `make` events, perform the following steps. (This example uses the `.Xdefaults` file.)

1. **Edit the `.Xdefaults` file in your home directory to include resources required by the commands you specified.**
   For example, the following `.Xdefaults` file entries provide the necessary resources for the `mail` commands specified in Step 2.

```
Maketool.Make.StartedCmd: mail rick < /home/server/rick/make_start_message
Maketool.Make.FailedCmd: mail rick < /home/server/rick/make_fail_message
Maketool.Make.CompletedCmd: mail dev_group </home/server/rick/make_done_message
```

2. **After you have updated the `.Xdefaults` file, you must execute the following command to make the changes effective:**

   ```
   xrdb -load ~/.Xdefaults
   ```

In order to provide audio as well as visual feedback on the state of a `make` build, you can use the `audioplay`(1) command. This command is part of the Demos software installation option if you have the SUNWaudio and SUNWaudmo packages added to your system. For information on how to install optional software, refer to *Installing SunPro Software on Solaris.*

Figure 2-4 shows the MakeTool Properties window with `audioplay` commands for each of the three events. The arguments to the `audioplay` commands access a different sound for each event, including a crash sound for a failed build.

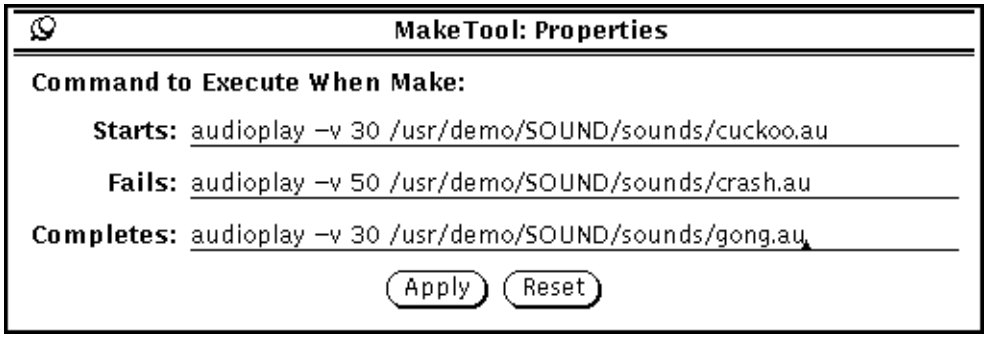


Figure 2-4 MakeTool Properties Window With `audioplay` Commands

The following `.Xdefaults` file entries provide the necessary resources for the `audioplay` commands specified in Figure 2-4.

```
Maketool.Make.StartedCmd: audioplay -v 30 /usr/demo/SOUND/sounds/cuckoo.au
Maketool.Make.FailedCmd: audioplay -v 50 /usr/demo/SOUND/sounds/crash.au
Maketool.Make.CompletedCmd: audioplay -v 30 /usr/demo/SOUND/sounds/gong.au
```

Browsing Makefiles 3

This chapter is organized into the following sections:

To browse a loaded makefile, click on the Browser button in the MakeTool window's control area. In response, the Makefile Browser window pops up, as shown in Figure 3-1.

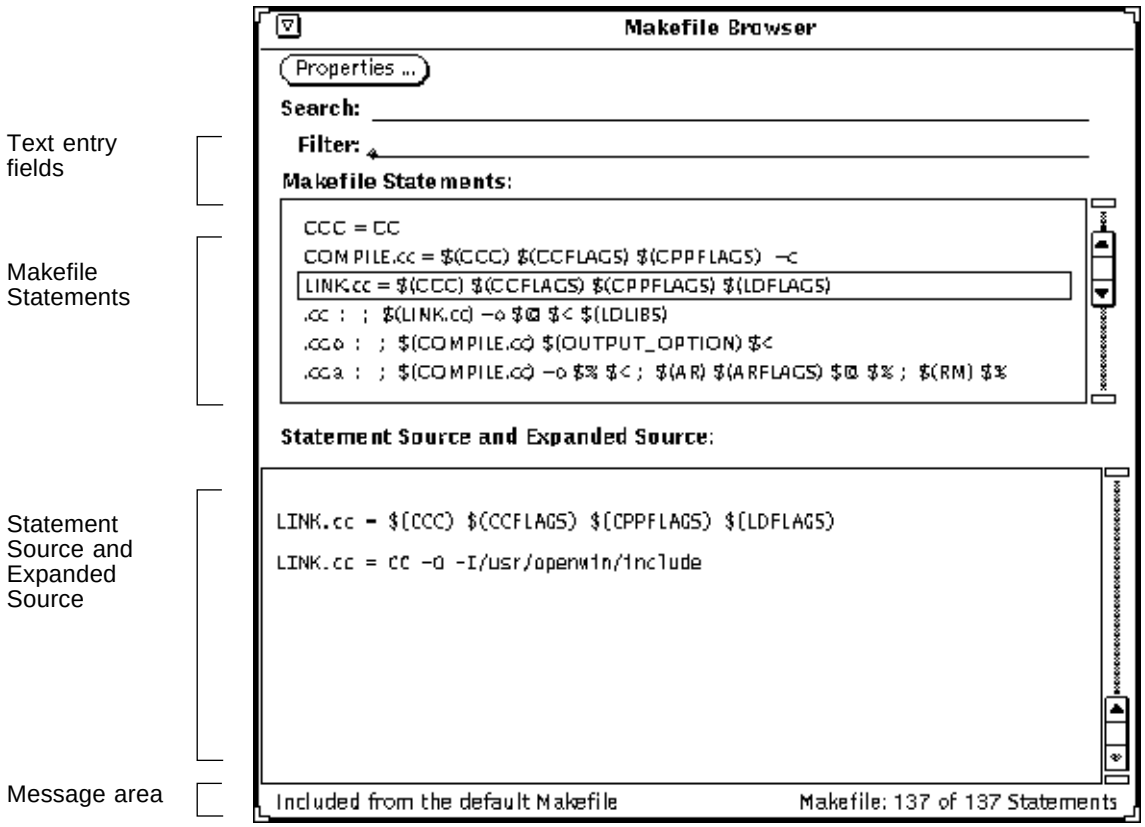


Figure 3-1 Makefile Browser Window

3.1 Browser Window

As shown in Figure 3-1, the Makefile Browser window contains

- Properties button
- Two text entry fields for searching and filtering
- Two read-only text panes that display makefile statements
- Message area

The window elements are described in detail in the sections that follow.

Properties Button

The Properties button opens the Makefile Browser Properties pop-up window (shown in Figure 3-2) which lets you control the display of makefile statements in the Browser window.

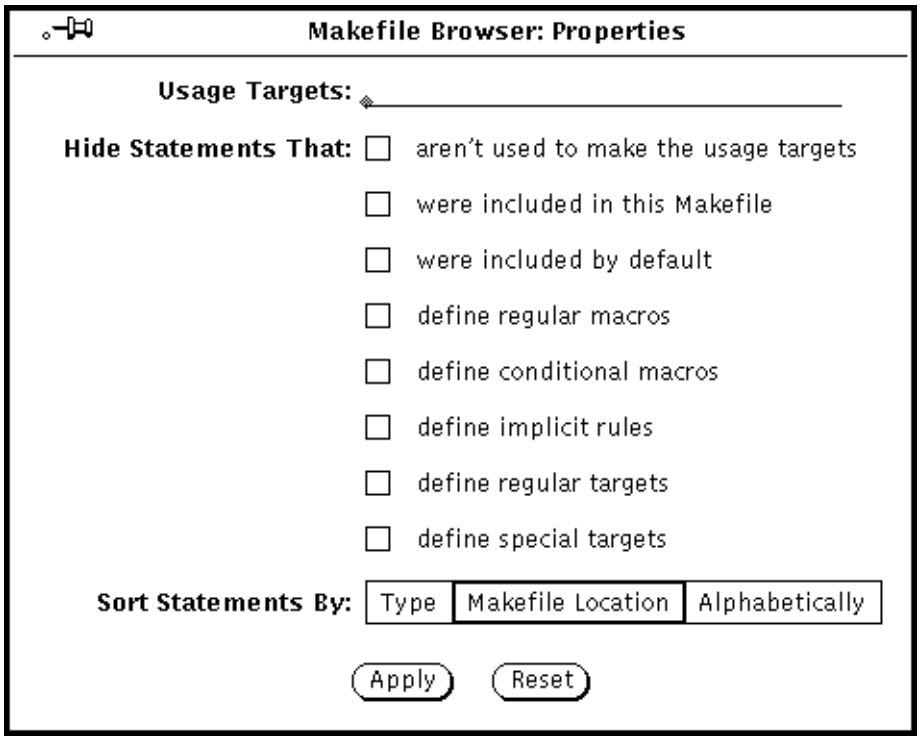


Figure 3-2 Makefile Browser Properties Window

The window presents two types of properties: those that hide statements and those that sort them. By default, Makefile Browser shows all the rules in your makefile and all the rules that are included from other makefiles.

Usage Targets

The Usage Targets text entry field lets you enter the names of targets on which you want to focus attention. You can then choose to display only statements that are used to build the usage targets by selecting Hide

Statements That aren't used to make the usage targets. By isolating only those statements that pertain to a few usage targets, you can more easily understand them.

Hide Statements That

These settings specify classes of statements that will be hidden in the Browser display panes. Hiding classes of statements allows you to more effectively focus your attention on statements of interest—macros that are defined in the loaded makefile, for example.

Indicate your choices by clicking in the check boxes next to the description of each class of statements. The options are defined below.

aren't used to make the usage targets
This option displays only those statements that are used to make the usage targets whose names you enter in the Usage Targets text entry field.

were included in this Makefile
Makefiles can include statements from other makefiles. This option hides such statements, displaying only statements that literally appear in the loaded makefile.

were included by default
Makefiles can include statements from a file of default make rules, named `make.rules` in SunOS 5.x and `default.mk` in earlier SunOS releases. Additional statements result from setting certain environment variables such as `.KEEP_STATE`, which implicitly includes a `.KEEP_STATE` special target in makefiles whenever `make` runs. This option hides such statements, displaying only statements that are *not* included from the default rules list and that are *not* the result of environment variables.

define regular macros
A *regular macro* is defined by the "equal sign" relational operator (`=`), as explained in the section titled "Macro" on page 1-3. This option hides regular macro definitions, displaying only statements that do *not* define regular macros.

define conditional macros
A *conditional macro* is defined differently for different targets, and is defined with a "colon equal-sign" relational operator (`:=`). This option hides conditional macro definitions, displaying only statements that do *not* define conditional macros.

define implicit rules
An *implicit rule* is a suffix rule or pattern-matching rule. Implicit rules define, for example, how to build object files (and append them with a .o suffix) from source files (with .c, .cc, .p, or .f suffixes). This option hides implicit rule definitions, displaying only statements that do *not* define implicit rules.

define regular targets
A *regular target* is any target that is not a special target (see below). This option hides regular target definitions, displaying only statements that do *not* define regular targets.

define special targets
A *special target* controls make behavior (for example, the way make tracks hidden dependencies). This option hides special target definitions, displaying only statements that do *not* define special targets.

The following table lists the special targets that can be hidden by MakeTool:

Special Makefile Targets	
.DEFAULT	.NO_PARALLEL
.DONE	.PARALLEL
.FAILED	.PRECIOUS
.IGNORE	.SCCS_GET
.INIT	.SILENT
.KEEP_STATE	.SUFFIXES
.MAKE_VERSION	.WAIT

Sort Statements By

The Browser sorts statements in one of three ways, as explained below. Indicate your choice by clicking on one of the three exclusive settings in the MakeTool Properties window.

Type
This setting sorts statements into rules, which are listed first, and macros, which are listed second. Within these two categories, statements are sorted into the following subcategories:

- Special rules first, then regular rules, then implicit rules

- All macros that are not conditional macros first, then conditional macros

Within each subcategory, statements are sorted alphabetically.

Makefile Location
This setting sorts statements according to the order of their appearance in the makefile. This is the default.

Alphabetically
This setting sorts statements alphabetically without regard to type.

Apply Button

Click on the Apply button to transmit your properties settings to the Makefile Browser.

Reset Button

Click on the Reset button to return all properties settings to the last applied state. To transmit these settings to the Makefile Browser, click on the Apply button.

Search Text Entry Field

Provides a way for you to enter search strings. As you enter text here, the Browser searches for the first match in the Makefile Statements pane. The search is incremental, meaning that it automatically finds the first match in the file each time you type a letter; it does not require that you press Return or give any other instruction to start searching after you enter text.

To display the second and succeeding matches for the text you enter, press the down-arrow key on the keyboard. At the end of the listing, the search wraps back to the top.

Filter Text Entry Field

Controls the statements that are displayed in the Makefile Statements pane. To display only statements that contain a specific text string, enter the string in the Filter field and press the Return key.

Makefile Statements Text Pane

Displays the makefile's rules and macros, sorted according to the choice you make in the Makefile Browser Properties window (see "Properties Button" on page 3-23). Each makefile statement is displayed on a single line, which does not wrap to window width. To view the full text of long statements, select the line by clicking SELECT on it. The source is then displayed in the Statement Source and Expanded Source text pane.

Statement Source and Expanded Source Text Pane

Displays two versions of statement source:

- The entire source line you have selected in the Makefile Statements window. The line, shown at the top of the pane, is reformatted, if necessary, for clarity: semicolons are replaced by line feeds, and spaces are added around operators.
- An expanded version of the statement. The expansion substitutes values for any macros that appear in the statement.

If the statement you select is very long or results in a large expanded version, you may need to scroll the pane up in order to view all the information.

Message Area

Indicates where the statement came from, the name of the makefile being browsed, the number of statements displayed in the Makefile Statements pane, and the total number of statements in the makefile (including statements from default and included makefiles). The "displayed" and "total" numbers may differ because filter and properties settings can reduce the number of statements displayed.

3.2 Examples

In Figure 3-1 on page 22, the statement beginning with `LINK.CC` was selected in the Makefile Statements pane. The statement contains macros, which can be recognized by the `$(MACRO)` syntax, shown in the Statement Source and Expanded Source pane. Below the statement source, the statement is displayed with these macros expanded so that their values are visible. At the bottom of

the window, the message area indicates that this statement was included from the default makefile, that the makefile is named `Makefile`, and that it contains a total of 138 statements, all of which are displayed in the Makefile Statements pane.

3.2.1 Searching for Text

In Figure 3-3, the string `CPPFLAGS` has been typed into the search text entry field. The first line that contains the word is selected in the Makefile Statements pane. To select other lines that contain `CPPFLAGS`, just press the down-arrow key. The search wraps through the makefile, so repeated pressing of the down-arrow key would eventually return to the first line that contains `CPPFLAGS`.

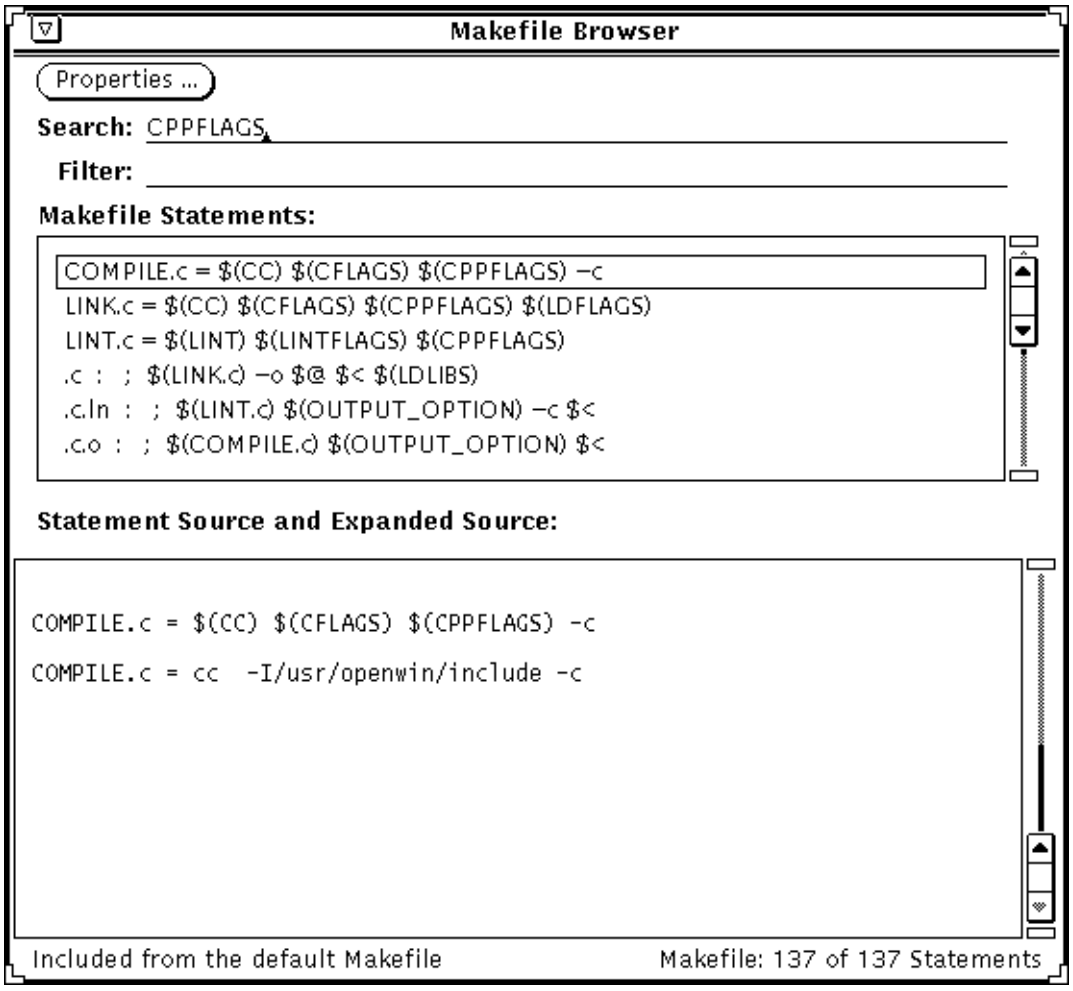


Figure 3-3 Makefile Browser Window with CPPFLAGS Search String

3.2.2 Filtering Statements

Figure 3-4 shows a Browser window in which the user has set a filter for the string LDFLAGS. The Makefile Statements pane shows only statements that contain the string LDFLAGS. The message area at the bottom of the window now shows that only 7 statements are displayed out of a total of 138 because only seven statements contain the string LDFLAGS.

.

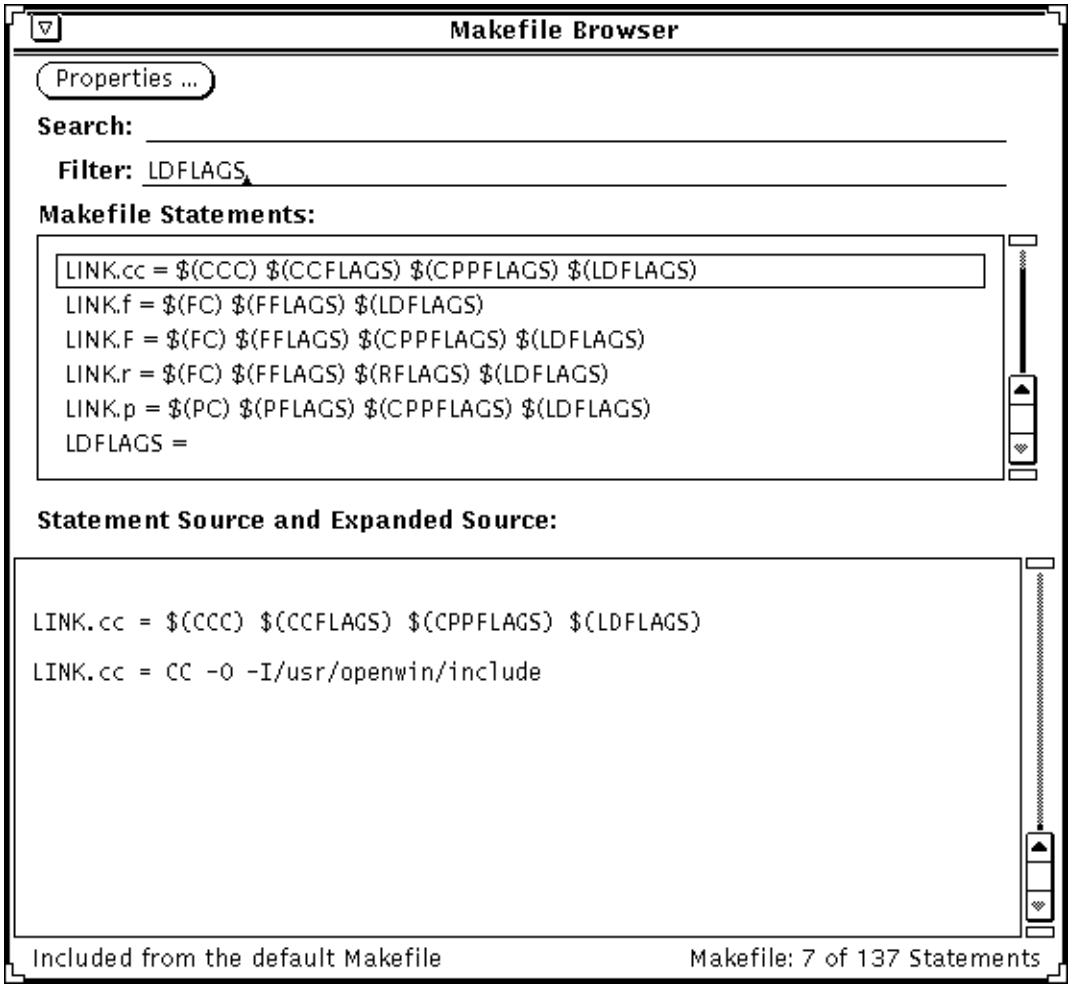


Figure 3-4 Browser Window with LDFLAGS Filter

3.2.3 Filtering with Properties

The Maketool Properties window provides another way to filter the statements shown in the Makefile Statements pane. In Figure 3-5, the user has instructed the Browser to hide all statements that define special targets. As a result, the statement that defines the target `.INIT` is no longer displayed.

The message at the bottom of the Makefile Browser window now shows that only seven statements (out of a total of 137) are displayed. The result is the set of all statements that do not define special targets and the set of all statements that contain the string `LDFLAGS`.

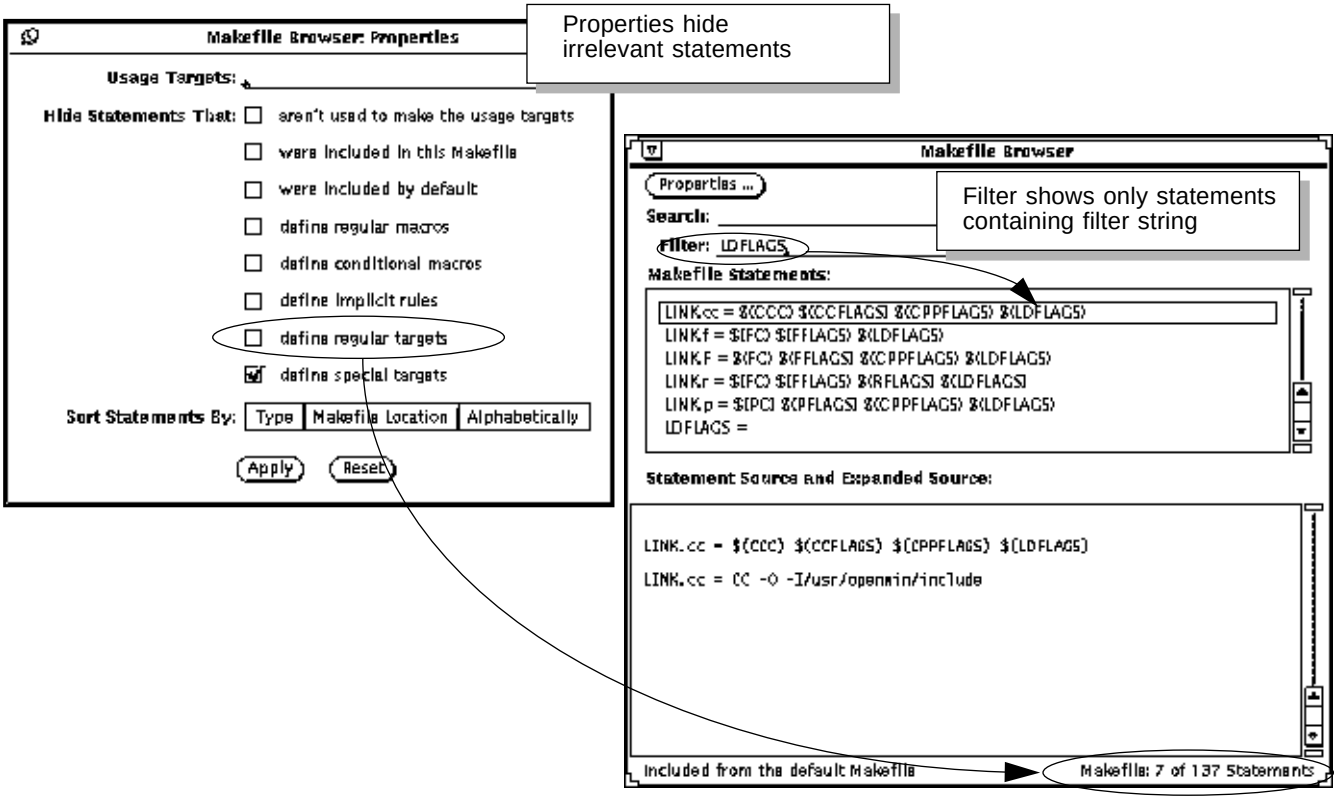


Figure 3-5 Setting Properties to Hide Statements

3

Troubleshooting A

This appendix describes how to overcome MakeTool problems. It is organized into the following sections:

A.1 Problem Checklist

If you are having problems using MakeTool, check for the following:

❑ That MakeTool is installed in the standard location. If you do not know where MakeTool should be installed, contact your system administrator.

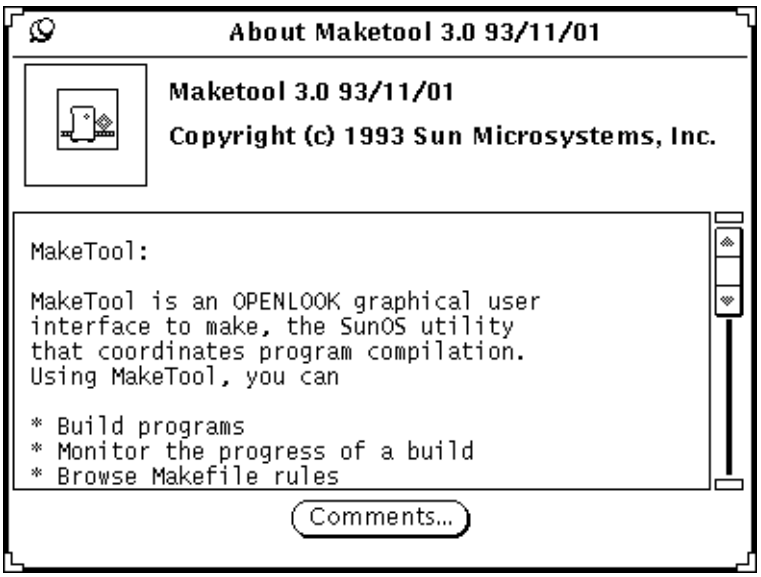
❑ That `maketool` can be found in your execution search path (as determined by the `PATH` environment variable).

A.2 Reporting Problems

If you have gone through the checklist and still have problems using MakeTool, call your local service office. Have MakeTool's version number ready to give to the dispatcher. Also, be ready to provide your operating system version number and your hardware system configuration.

To display MakeTool's version:

♦ **Click on the About button.**
The resulting pop-up window shows the MakeTool product release number.



A.3 MakeTool Error Messages

The error messages displayed by MakeTool are those of the `make` program. If a `make` build fails, examine the MakeTool window's output pane for error messages from `make`. If you need help interpreting the messages, consult your operating system documentation for the `make` utility.

Index