sun®
microsystems

Sun SourceBrowser Reference Manual

Part No: 800-3464-10
Revision A of 16 March 1990

Contents

Figures

Preface

This manual explains how to use Sun SourceBrowser, a software tool for software developers.

Audience

This manual is written for programmers who want to use SourceBrowser on Sun™ workstations running Sun C, FORTRAN, or Pascal.

SourceBrowser can be used in either a window or command-line environment. If you want to use SourceBrowser in the window environment, you should be familiar with the SunView1 window environment. For information about SunView™, see the *SunView1 Beginner's Guide*.

Organization

This book includes five chapters and two appendices:

Chapter 1, "Introduction," provides an overview of the capabilities of SourceBrowser.

Chapter 2, "Using Sun SourceBrowser," provides step-by-step instructions for using SourceBrowser in a window environment.

Chapter 3, "Browsing Large Programs," explains the procedures you will follow to use SourceBrowser in conjunction with large software projects.

Chapter 4, "Window Command Reference," provides a concise description of each SourceBrowser window command.

Chapter 5, "Command-Line Reference," describes how to use SourceBrowser in the command-line environment.

Appendix A, "Browsing FORTRAN Programs," provides a brief explanation of browsing FORTRAN programs.

Appendix B, "Browsing Pascal Programs," provides a brief explanation of browsing Pascal programs.

Typographical Conventions

This book follows these typographical conventions:

Times Roman font, Italic face indicates:

- A variable name
- A command argument

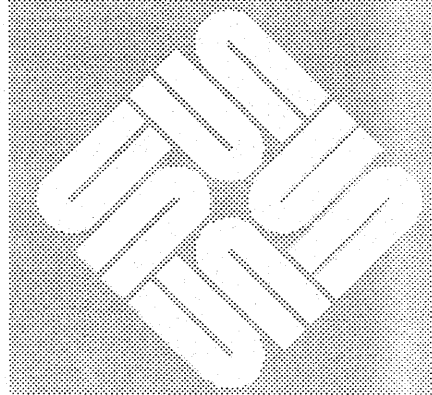
`Courier` font indicates:

- A listing
- A command name
- A menu name
- A program name

Courier font, Bold face indicates what the user types.

1

Introduction



1

Introduction

Sun SourceBrowser is an interactive tool to aid programmers in the development and maintenance of software systems, particularly large ones.

During the course of a programming project, new programmers often join the programming team to enhance, maintain, and port code. But before becoming productive team members, these individuals need to fully understand the code that they will modify. SourceBrowser is a powerful tool for helping programmers quickly grasp large programs. Specifically, SourceBrowser assists programmers by finding *all* occurrences of any symbol of their choice, including those found in header files.

SourceBrowser uses a "what you see is what you browse" paradigm. In other words, the source code you manipulate is the same source code SourceBrowser uses in its searches.

SourceBrowser is an extensible, open system that has been designed to be used with multiple languages. Currently, SourceBrowser can be used with Sun C, FORTRAN, and Pascal.

1.1 Window and Command Line Environments

You can use SourceBrowser in either the SunView1 window environment or a command-line environment. The SunView1 environment, in particular, is extremely easy to use. To issue a query, for example, you simply type the symbol you want SourceBrowser to find; then press Return.

While the SourceBrowser window environment can be accessed only from Sun workstation, the command-line environment can be used from character terminals and from workstations emulating terminals.

1.2 How it Works

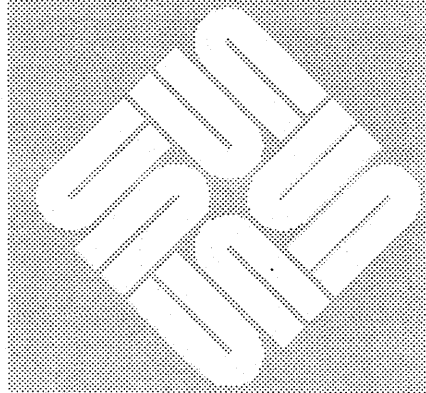
You use SourceBrowser by issuing *queries* which instruct SourceBrowser to find all occurrences of the identifier, string constant, or search pattern that you have specified. You then view *matches* (occurrences of the symbol you requested SourceBrowser to find) with their surrounding source code. SourceBrowser maintains a list of all queries you have conducted during a SourceBrowser session. This makes it easy to issue a query, view the matches associated with that query, conduct another query and then return to your original query — all with a minimum number of commands.

In addition to basic commands for issuing, adding, deleting, and moving between queries, SourceBrowser includes many powerful features. The `Filter` command, for example, lets you search for symbols based on the way in which they are used in a program, so you can issue very specific queries, such as "Show all occurrences of the variable *age* when *age* is used as a structure field name." To limit your search to specific items of certain classes of code, such as files or functions, you can use SourceBrowser's `Focus` command. You also can edit code from within SourceBrowser.

SourceBrowser maintain a high level of efficiency regardless of the size of the code you are browsing. That's because, once SourceBrowser has built the database it uses to respond to queries, the size of the code you are browsing has only minimal impact on SourceBrowser's speed. (See Section 3.8, "Performance Considerations," for details.)
SourceBrowser also includes several features specially designed for use with large programming projects. Chapter 3, "Browsing Large Programs," describes those features. Note that, because SourceBrowser uses the selection service, it is integrated with other software development tools such as dbxtool.

2

Using Sun SourceBrowser



2

Using Sun SourceBrowser

This chapter gives you the information you need to use Sun SourceBrowser in a window environment. If you want to use SourceBrowser in the command-line environment, see Chapter 5, "Command-Line Reference."

All of the examples in this chapter are written in C. If you want to use SourceBrowser with FORTRAN programs, see Appendix A. If you want to use SourceBrowser with Pascal programs, see Appendix B.

This chapter tells you how to

- start SourceBrowser
- change the current working directory
- specify when SourceBrowser should build the index it uses to locate information in the SourceBrowser database
- use SourceBrowser's window environment
- display the current match
- view matches
- remove matches
- move between queries
- remove queries
- search for string constants
- use wildcards in queries
- use the `Filter` command to search for identifiers and string constants based on how they are used in a program

- use the `Focus` command to restrict a query to classes of code, such as files and libraries
- edit code from SourceBrowser
- use the property sheet to customize SourceBrowser
- use SourceBrowser with dbxtool

2.1 A Sample Program

Throughout this chapter, the following sample program is used as the basis for all examples.

Figure 2-1 *Sample Program*

```
textedit - sample.c, dir: /home/examples/browser
Scratch window
#ident "Source Browser Demo"

#include <stdio.h>

#define SECONDARY_DEMO times

extern  void    hello();
extern  void    world();

void
main(argc, argv)
        int     argc;
        char    *argv[];
{
        int     times = 1;

        if (argc > 1) {
                if (sscanf(argv[1], "%d", &times) != 1) {
                        times = 1;
                }
        }

        for (; times > 0; times--) {
                hello(stdout);
                world(stdout);
        }
}

static void
hello(file)
        FILE    *file;
{
        (void)fprintf(file, "Hello ");
}

static void
world(file)
        FILE    *file;
{
        (void)fprintf(file, "world\n");
}
```

2.2 Compiling with the Source-Browser Option

SourceBrowser responds to queries by searching in a specialized database that contains pertinent information about the files you are browsing. SourceBrowser's default is to build that database in the current working directory. If you want SourceBrowser to construct a database in a directory that is not the current working directory, see Chapter 3, "Browsing Large Programs."

Before using SourceBrowser, you must recompile each of the source files you want to browse.

To turn on SourceBrowser if you are using `make`:

- Add `-sb` to CFLAGS in the makefile for C programs
- Add `-sb` to FFLAGS in the makefile for FORTRAN 77 programs
- Add `-sb` to PFLAGS in the makefile for Pascal programs

The `-sb` option causes the compiler to add information to the SourceBrowser database. The information that is included depends on the particular compiler. In general, only identifiers and strings are included; that is, reserved words are *not* included.

To turn on SourceBrowser if you are not using `make`:

- add the `-sb` option to the compiler command line

Figure 2-2 *Enabling Browsing from C*

```
venus% cc -sb hello.c -c
```

Figure 2-3 *Enabling Browsing from FORTRAN 77*

```
venus% f77 -sb hello.f -c
```

Figure 2-4 *Enabling Browsing from Pascal*

```
venus% pc -sb hello.p -c
```

NOTES: To generate correct SourceBrowser information, you must issue a compile run that does not generate any warnings or error messages.

Each source file compiled with the `-sb` option must have the same absolute path regardless of the machine from which it is referenced. If this is not the case, see Section 3-2, "Browsing From Multiple Machines."

You may find it useful to use SourceBrowser to browse declarations in Lint library specification files. To do that, compile the appropriate library specification file with the `-sb` option, as shown in Figure 2-5.

Figure 2-5 Browsing Lint Library Specification Files

```
venus% cc -c -sb /usr/lib/lint/llib-lc=.c -o /temp/junk.o
```

Depending on which libraries you use, you may want to compile other files in /usr/lib/lint in the same fashion.

Note that the =.*c* portion of the command is necessary to inform the Compiler that you are compiling a C file, even though the filename doesn't end with .*c* . The `-o` portion of the command is included to indicate that the result of the compilation is useless information. See Lint (1) for more about Lint files.

2.3 Starting SourceBrowser

To start SourceBrowser:

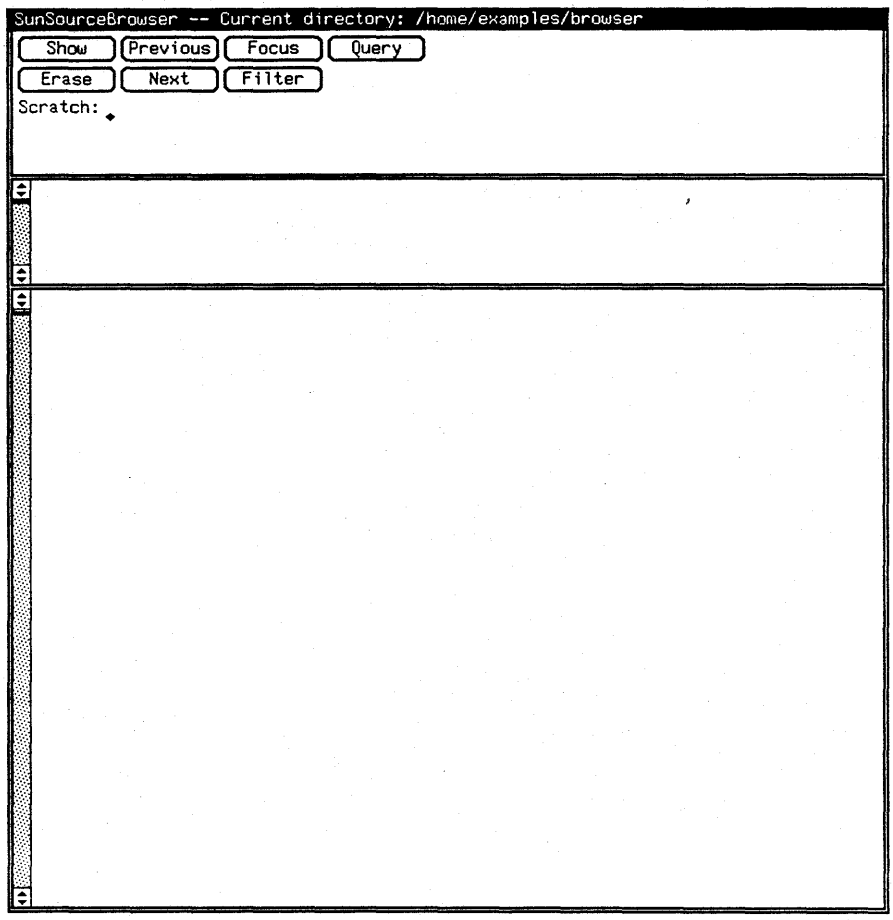
- type `sbrowser&` at the command prompt

Figure 2-6 *Starting SourceBrowser*

```
venus% sbrowser&
```

SourceBrowser appears as a window similar to the following.

Figure 2-7 *SourceBrowser Window at Start-Up*



The content of the SourceBrowser window is explained in Section 2.8, "The SourceBrowser Window."

NOTE: To bypass additional introductory information and learn how to issue a query, see section 2.7, "Issuing Queries."

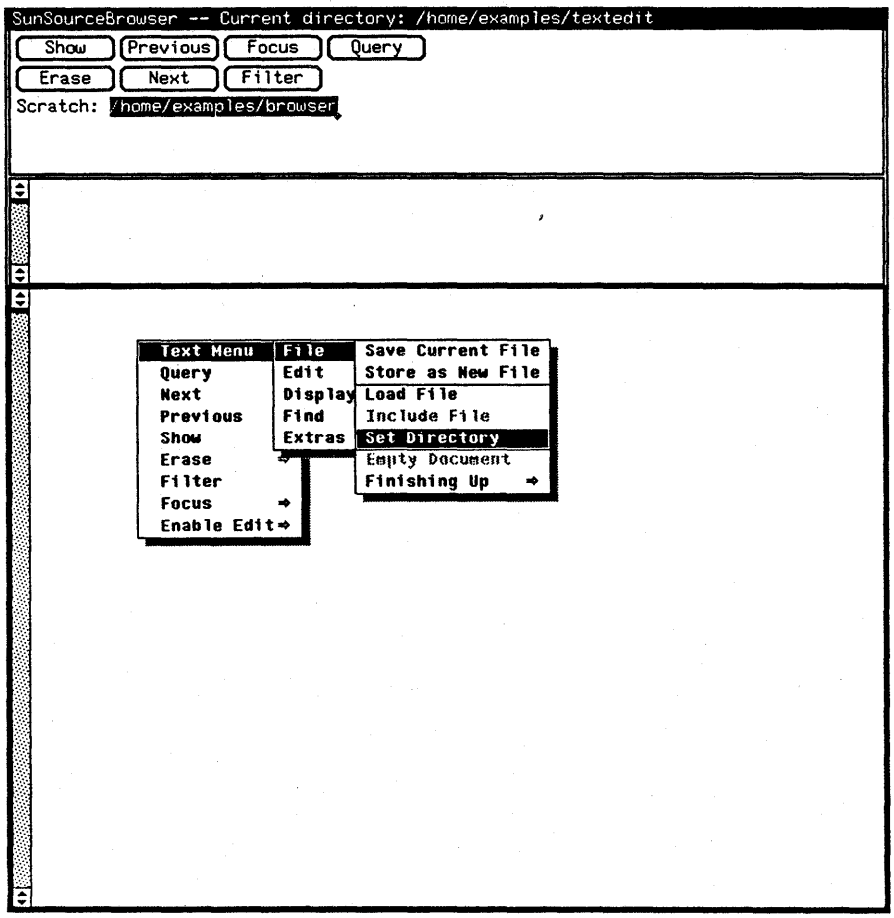
2.4 Changing the Current Working Directory

SourceBrowser's default is to browse all source files described by the database in the current working directory. The current working directory is displayed in the frame header, the dark stripe that runs across the top of the SourceBrowser window.

To change the current working directory from within SourceBrowser:

1. Select the path to the desired directory. See the *SunView 1 Beginner's Guide* for information about making selections.
2. Position the pointer in the match subwindow (the middle subwindow) or the source subwindow (the bottom subwindow) and hold down the right mouse button to select `Set Directory` from the `File` pull-right menu of the `Text Menu`.

Figure 2-8 *Changing the Current Working Directory*



2.5 Building the SourceBrowser Database Index

When you issue a query, SourceBrowser searches in a specialized database containing data about your source code. When you issue your initial query following a compilation or recompilation, SourceBrowser builds an index for this databaseand then processes your query.

You can, however, instruct SourceBrowser to conduct a query each time you run `make`. That way, the index will already be built when you issue your first query, so the query will be performed faster.

To automatically rebuild the database index every time you run `make`, add the following lines to your makefile.

```
.DONE: query

query:

   -sbquery symbol_not_in_prog
```

where *symbol_not_in_prog* is a symbol that does not appear in the code you are compiling.

Another option is to instruct SourceBrowser to only update its database when you explicitly instruct it to do so. You do this by changing the `Update Database for Each Query` option in the property sheet to `No`. See Section 4.2, "The SourceBrowser Property Sheet," for details.

2.6 Querying Options

As stated in the Introduction, a *query* is an instruction to SourceBrowser to find a specified symbol. To issue a query, you first need to determine the symbol you want SourceBrowser to search for. SourceBrowser can search for

- identifiers
- string constants
- search patterns that contain wildcards

SourceBrowser's default is to search for identifiers. See Section 2.14, "Searching for String Constants," and Section 2.15, "Using Wildcards in Queries," for instructions on searching for string constants and using wildcards in queries.

2.7 Issuing Queries

To issue a query to SourceBrowser, simply select the identifier you want SourceBrowser to search for and click left on the `Query` button.

The identifier you select can appear in any open window.

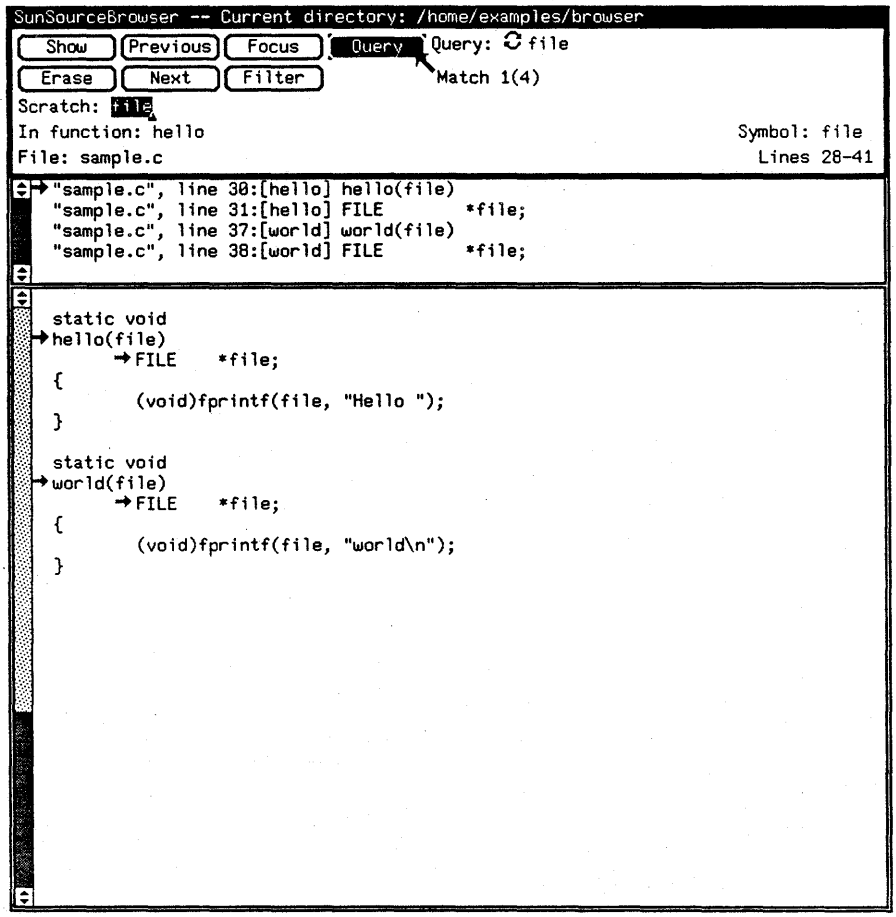
The Initial Query

To issue the first query of a session (or to issue a query when no other query is displayed):

1. Select the identifier you want to search for. The identifier can be:
   - an identifier you have typed in the Scratch field
   - an identifier in another window
2. Click left on the `Query` button.

For example, in Figure 2-9, the user instructed SourceBrowser to find all instances of *file* by typing *file* in the Scratch field; selecting *file*; then clicking left on the `Query` button.

Figure 2-9 *Issuing a Query*



See Section 2.8, "The SourceBrowser Window," for a description of the content of each SourceBrowser subwindow.

The following is important additional information about issuing a query.

- When using the Scratch field to issue a query, you can simply type the query in the Scratch field and press Return.
- You can reduce the time it takes to respond to a query by half by not including the source code in the match subwindow. You do this by setting the `Match Window Contents` option in the property sheet to `Filename and Linenumber Only`. See Section 4.2, "SourceBrowser Property Sheet," for details. See Section 3.9, "Performance Considerations," for more about SourceBrowser performance.
- To abort a query, press the Stop or L1 function key.
- If you receive a message starting "Request for xxx bytes of memory failed" you have run out of swap space. Use the `mkfile` (8) command to create more swap space or the `swapon` (8) command to abort existing processes (windows) and free up existing swapspace. To determine which processes occupy significant swap space, use the `ps uagx` command and look in the SZ column. To determine how much swap space you have, use the `pstat -s` command.
- If SourceBrowser displays messages indicating that it is having difficulty conducting a query (for example, if it says it is unable to find `.bd`--browser data--files), remove the .sb directory (that is, the directory containing the SourceBrowser database) and recompile everything with the `-sb` option. If you receive a message stating that the database is locked, you can either issue the `-break_lock` option or remove the .sb subdirectory and recompile. See Section 5.4, "Command-Line Options," for a description of the `-break_lock` option.
- SourceBrowser uses standard C notation to display non-printing characters included in queries.

Subsequent Queries

Issuing subsequent queries is similar to issuing the initial query. The only difference is that you now have the option of choosing as your search symbol text that is displayed in the match subwindow or source subwindow.

To issue a subsequent query:

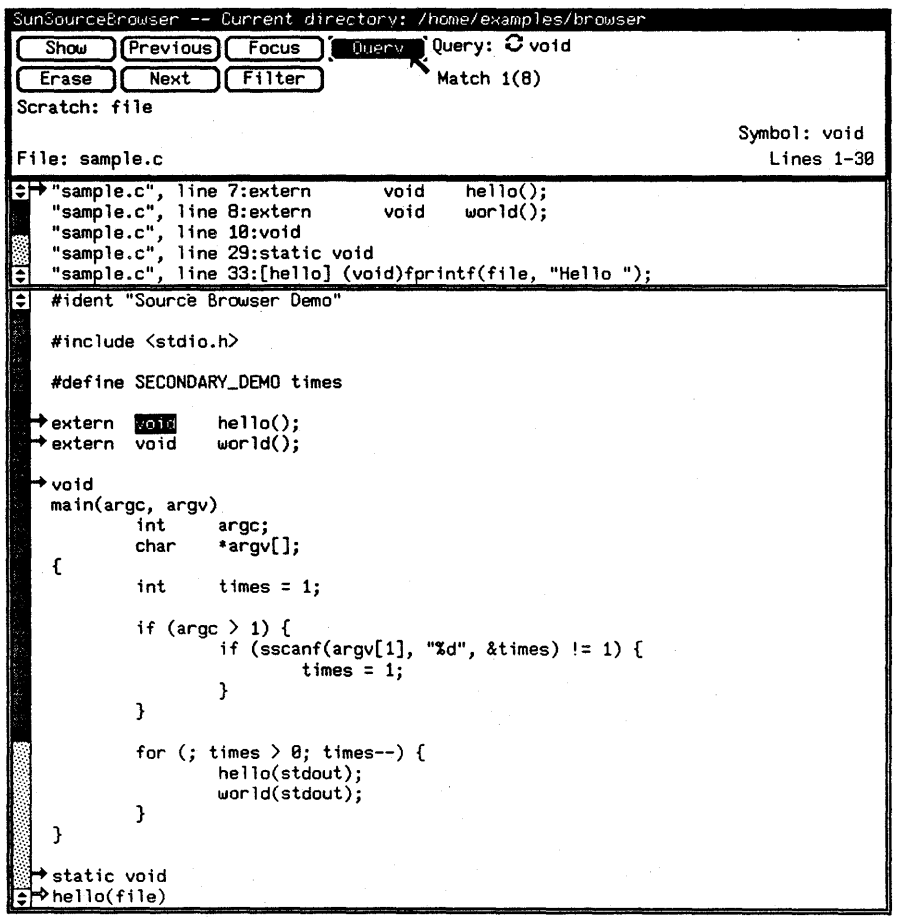
1. Select the identifier you want to search for. The selected identifier can be an identifier that you have typed in the Scratch field or an

identifier that appears in a SourceBrowser subwindow or in another window.

2. Click left on the `Query` button.

For example, in Figure 2-10, the user has instructed SourceBrowser to find all instances of *void* by selecting *void* in the source subwindow and then clicking left on the `Query` button.

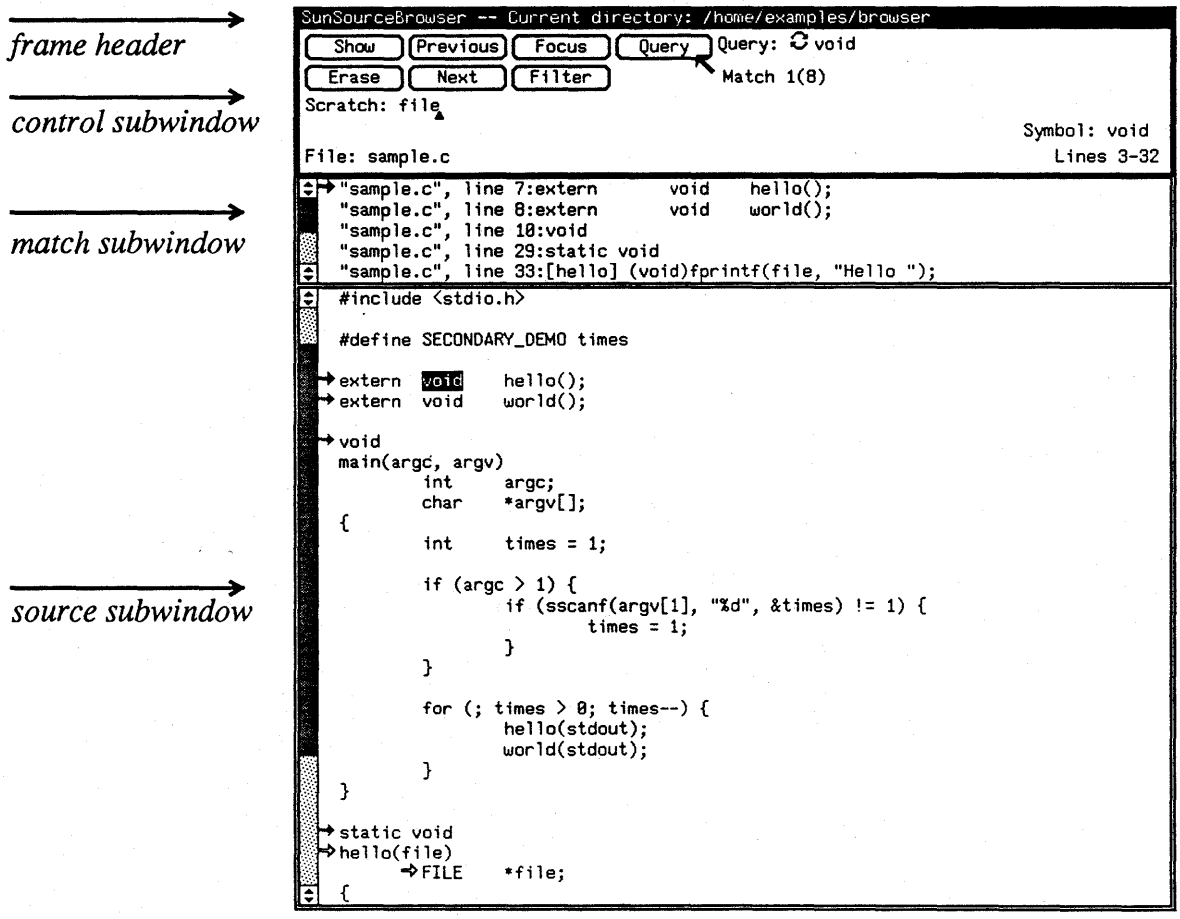
Figure 2-10 *Issuing a Subsequent Query*



2.8 The SourceBrowser Window

The SourceBrowser window consists of a frame header and three subwindows: the control subwindow, the match subwindow, and the source subwindow.

Figure 2-11 *The SourceBrowser Window*



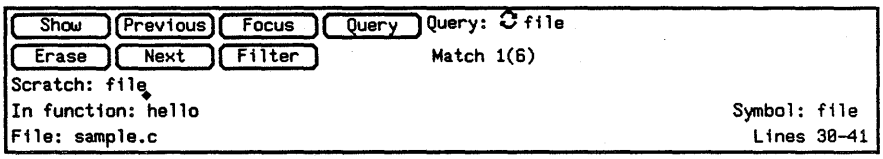
The Frame Header

The SourceBrowser header is the stripe that runs across the top of the SourceBrowser window. When SourceBrowser is invoked, the header consists of *SourceBrowser* followed by the current working directory. The frame header also tells you whether filtering and/or focusing is turned on.

The Control Subwindow

The control subwindow contains the controls needed to manipulate SourceBrowser. Specifically, the buttons and underlying menus in the control subwindow let you issue queries, move between matches (occurrences of the symbol specified by the query), move between queries, erase matches and queries, and narrow your search.

Figure 2-12 *The Control Subwindow*



Buttons and Menus

You use standard SunView conventions to manipulate buttons in the control subwindows.

- To activate a button, position the pointer on the button and click the left mouse button
- To display a menu under a button, position the pointer on the button and hold down the right mouse button

Clicking left on a button is equivalent to choosing the first option in the menu behind the button.

Text Fields and Other Items

The control subwindow contains one text field, called *Scratch*, that contains strings that you type. To issue a query you can simply type the identifier you want to search for in the Scratch field and press Return.

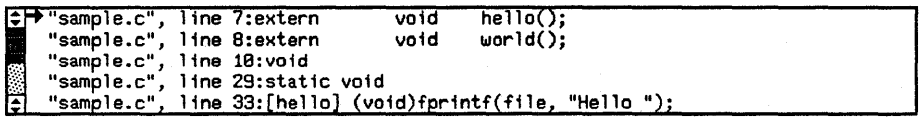
After you have issued a query, the control subwindow also contains the following items:

- *In function* specifies the function containing the current match (if applicable)
- *File* specifies the file containing the current match
- *Query* shows the current query
- *Match* displays the total number of matches found during the current query and the number of the current match. For example, in Figure 2-13, the current query has resulted in six matches, the first of which is the current match. The Match field also displays the number of secondary matches. See Section 2.16, "Using the Filter Command to Narrow a Search," for details.
- *Symbol* displays the identifier or string constant for the current match
- *Lines* displays the line numbers of the source code displayed in the source subwindow

The Match Subwindow

The match subwindow is a subwindow that displays all matches found by the current query. When you issue a query, the first match SourceBrowser finds is defined as the *current match.* The current match is identified by a black arrow in the match subwindow. Matches containing declarations appear first in the match subwindow, followed by all other matches.

Figure 2-13 *The Match Subwindow*



Information about each match listed in the match subwindow includes

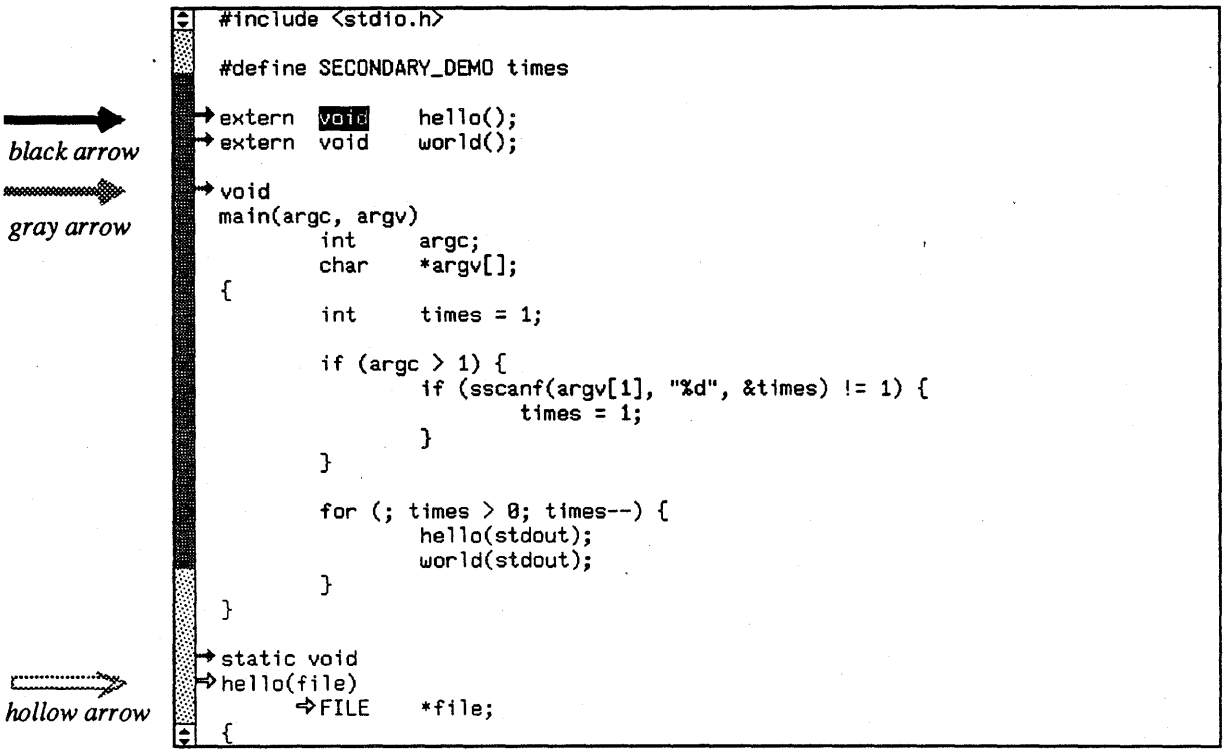
- the name of the file in which the match appears, without the preceding path
- the line number in the file on which the match appears
- if applicable, the function in which the match appears
- the line of code containing the match with preceding white space removed (optional)

NOTE: SourceBrowser's default is to display lines of code in the match subwindow and to truncate lines of code that don't completely fit in the subwindow. If you want to eliminate source code from the match subwindow altogether, or wrap lines of code in the match subwindow, you can modify SourceBrowser's property sheet. See Section 4.2, "The SourceBrowser Property Sheet," for details.

The Source Subwindow

The source subwindow displays the portion of source code that contains the current match. The source subwindow may also contain markers in the form of arrows that identify matches found during the current and other active queries.

Figure 2-14 *The Source Subwindow*



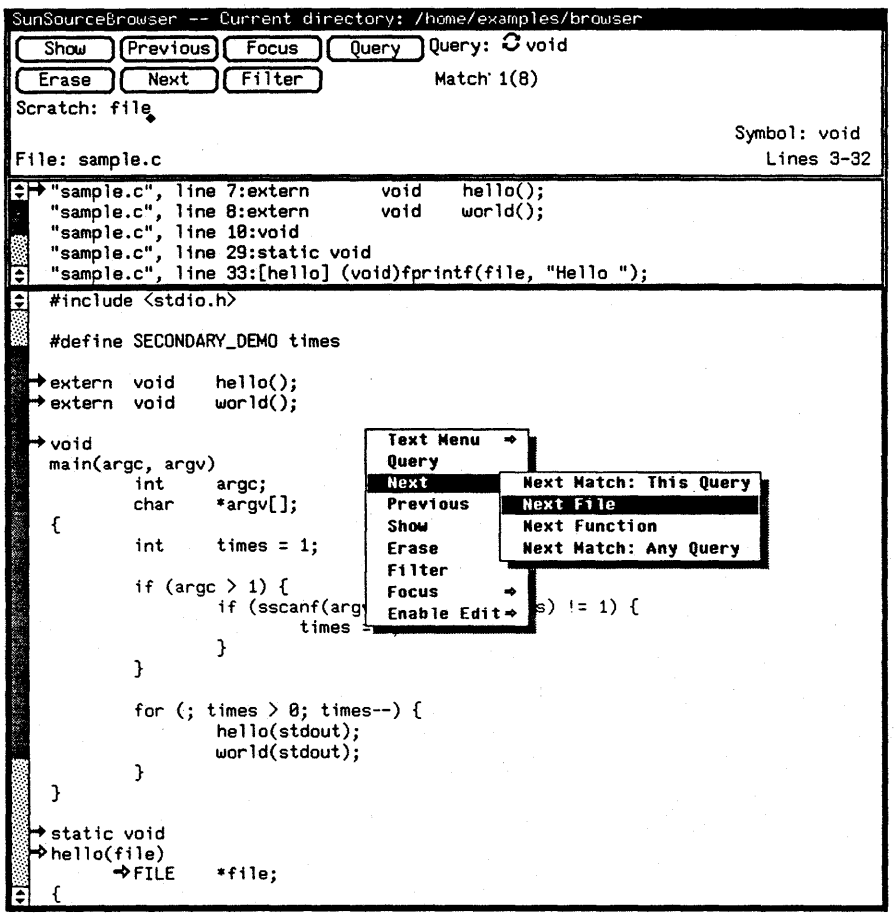
- A solid black arrow indicates the current match
- A gray arrow indicates a match other than the current match that has been found by the current query
- A hollow arrow indicates a match found by a query other than the current query
- A solid arrow with a line through it indicates a secondary match, that is, an identifier inside a macro definition. This marker only appears when you are using the `Filter` command. See Section 16, "Using the Filter Command to Narrow a Search," for details.

NOTE: You can use the property sheet to instruct SourceBrowser not to display any of these markers in the source subwindow or to indent the markers. See Section 4.2, "The SourceBrowser Property Sheet," for details.

Displaying SourceBrowser Menus

You can display SourceBrowser menus in either the match subwindow or the source subwindow by positioning the pointer in the subwindow in which you want the menu to appear and holding down the right mouse button.

Figure 2-15 *Displaying the SourceBrowser Menu*



Scrolling in SourceBrowser Windows

You can scroll through the match subwindow and the source subwindow using the scroll bars and can move and resize these subwindows by dragging their borders. For information about scrolling and resizing windows, see the *SunView 1 Beginner's Guide*.

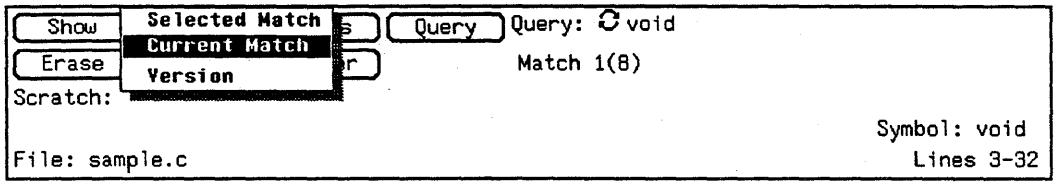
2.9 Redisplaying the Current Match

Often, when using SourceBrowser, you'll scroll through several screenfuls of source code in the source subwindow, so that the current match is no longer displayed.

To redisplay the current match in the source subwindow:

- Hold down the right mouse button and choose `Current Match` from the `Show` menu

Figure 2-16 *The `Show Current Match` Command*



2.10 Viewing Matches

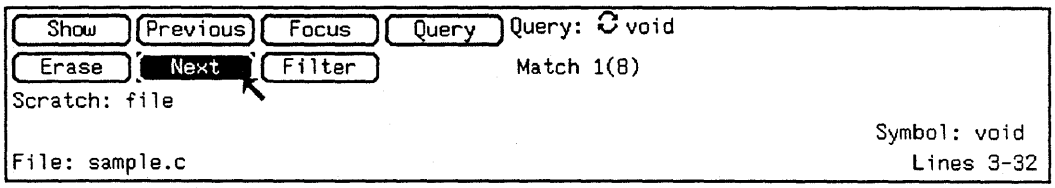
SourceBrowser provides several ways to specify a different match as the current match.

Using the `Next` Command to View Matches

To make the next match in the match subwindow the current match and display the source code surrounding that match in the source subwindow:

- Click left on the `Next` button. This is equivalent to choosing `Next Match: This Query` from the `Next` menu.

Figure 2-17 *The `Next Match: This Query` Command*

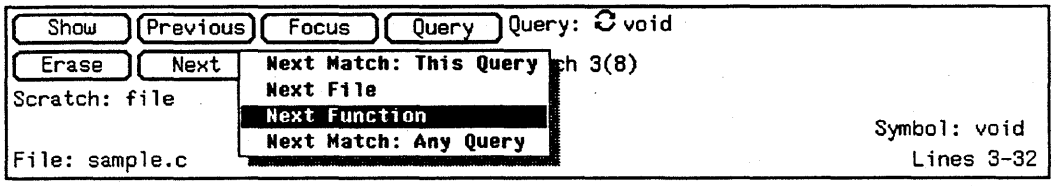


You can also make the first match that occurs in the next function or file that contains a match the current match. When you do, SourceBrowser displays the source code surrounding that match in the source subwindow.

To make the first match that occurs in the next function or file the current match:

- Hold down the right mouse button and choose `Next File` or `Next Function` from the `Next` menu

Figure 2-18 *The `Next Function` Command*

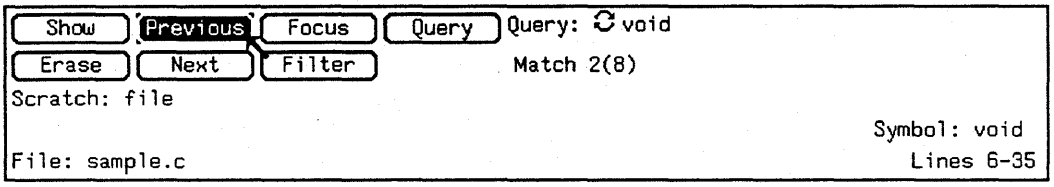


Using the `Previous` Command to View Matches

To make the previous match in the match subwindow the current match and display the source code surrounding that match in the source subwindow:

- Click left on the `Previous` button. This is equivalent to choosing `Previous Match: This Query` from the `Previous` menu.

Figure 2-19 *The `Previous Match: This Query` Command*

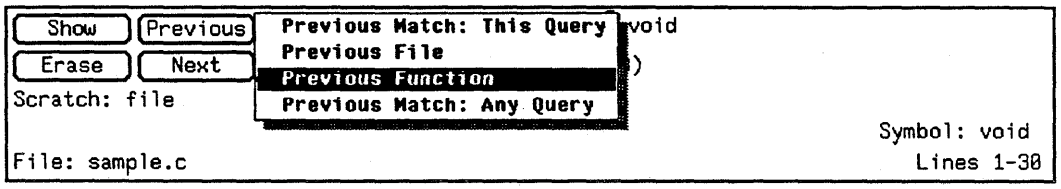


You can also make the last match that occurs in the previous function or file that contains a match the current match. When you do, SourceBrowser displays the source code surrounding that match in the source subwindow.

To make the last match that occurs in the previous function or file the current match:

- Hold down the right mouse button and choose `Previous File` or `Previous Function` from the `Previous` menu

Figure 2-20 *The `Previous Function` Command*



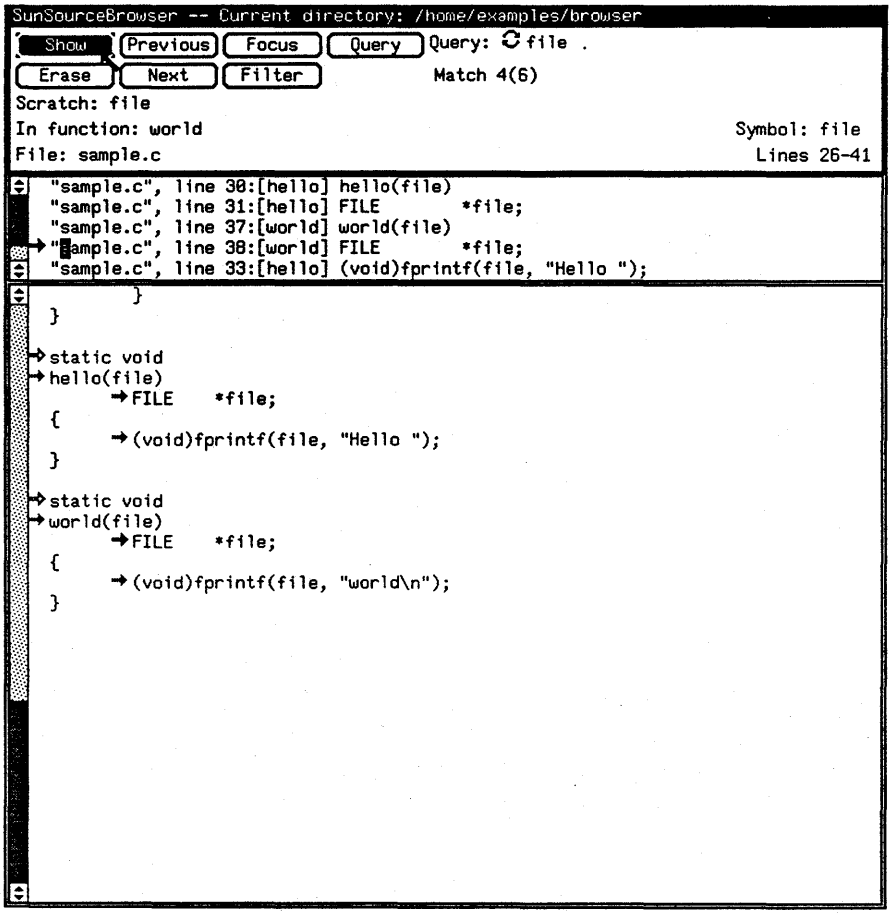
Selecting a Different Current Match

It's often convenient to scroll through the match subwindow until you find a match that you want to examine more closely.

To make a match in the match subwindow the current match:

1. Use the scroll bar in the match subwindow to scroll until the match you want to select is displayed.
2. Select the match by clicking left anywhere on the line containing the match.
3. Click left on the `Show` button. This is equivalent to choosing `Show Selected Match` from the `Show` menu.

Figure 2-21 *The `Show Selected Match` Command*



For example, in Figure 2-21, the user selected the fourth line in the match subwindow and then clicked left on the `Show` button. The black arrow moved to the selected match in the match subwindow and the surrounding source code was displayed in the source subwindow.

2.11 Erasing Matches

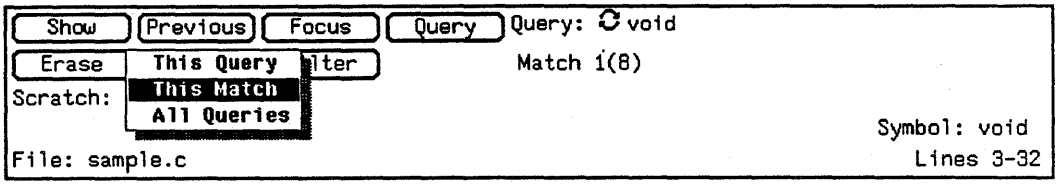
When viewing matches, you may want to erase matches that are no longer of interest.

To erase a match:

1. Make the match you want to erase the current match.
2. Hold down the right mouse button and choose `This Match` from the `Erase` menu.

The number of matches shown by the Match item in the control subwindow is reduced by one.

Figure 2-22 *The `Erase This Match` Command*



2.12 Moving Between Queries

During a SourceBrowser session, you often will issue several queries. SourceBrowser maintains a list of these queries, called *active queries*, so you can easily move between them.

When you move between queries, SourceBrowser retains the current match for each query. Thus, you can easily move between queries without losing your place in the code you are browsing.

SourceBrowser provides multiple ways to move between queries.

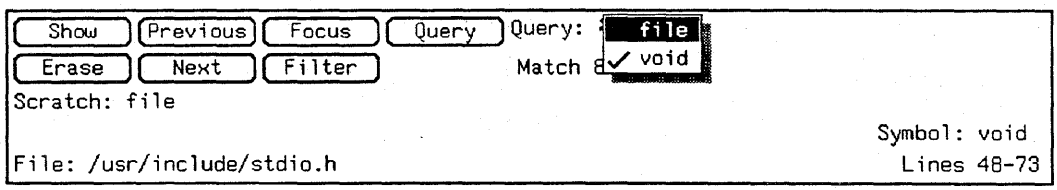
- You can choose a current query from the menu of active queries
- You can cycle through the active queries
- You can make the next match marked by a gray or hollow arrow the current match. (That is, you can make the next match appearing in the source code after the current match the current match even if the match was obtained by another query.) The query used to obtain that match then becomes the current query.
- You can make the previous match marked by a gray or hollow arrow the current match (that is, you can make the previous match appearing in the source code before the current match the current match even if the match was obtained by another query.) The query used to obtain that match then becomes the current query.

Selecting Queries from the `Query` Menu

To choose a new current query from the menu of active queries:

- Click right on the Query item and choose a query from the menu of active queries

Figure 2-23 *Displaying the `Query` Menu*



Cycling Through Queries

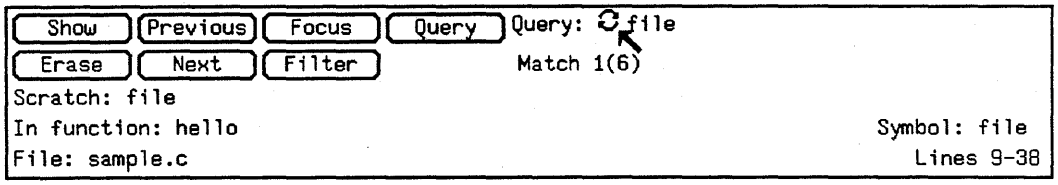
The cycle icon provides a quick way to view active queries.

To cycle forward through the active queries:

- Position the pointer on the cycle icon and click the left mouse button

The Query item changes accordingly.

Figure 2-24 *The Cycle Icon*



To cycle backward through the active queries:

- Position the pointer on the cycle icon, hold down the Shift key and click the left mouse button

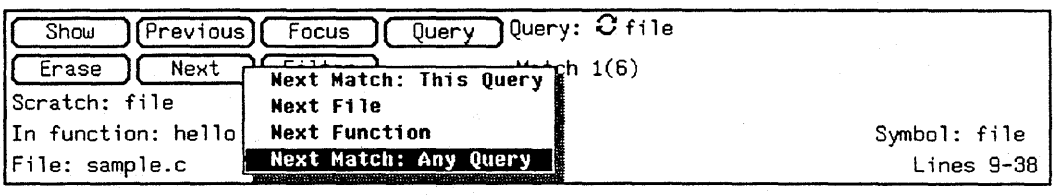
Using the `Any Match` Command

As you scroll through your source code, you may want to make the next match or previous match in the source code the current match. If the new current match was found by another query, that query becomes the current query and the Query item in the control subwindow changes accordingly.

To make the next match the current match:

- Hold down the right mouse button and choose `Next Match: Any Query` from the `Next` menu

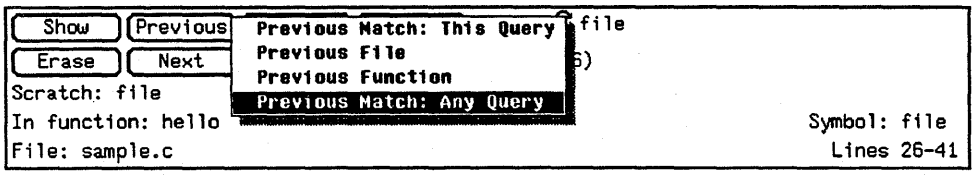
Figure 2-25 *The `Next Match: Any Query` Command*



To make the previous match the current match:

- Hold down the right mouse button and choose `Previous Match: Any Query` from the `Previous` menu

Figure 2-26 *The `Previous Match: Any Query` Command*



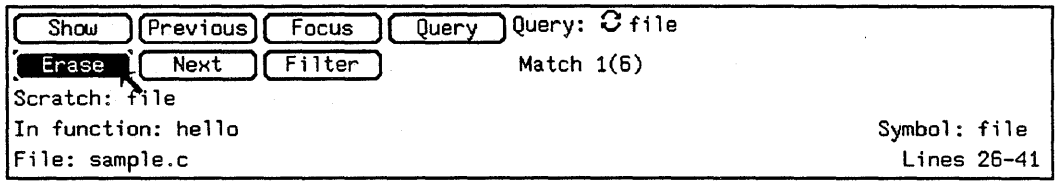
2.13 Removing Queries

When a query is no longer of interest, you can remove it from the list of active queries.

To remove the current query from the list of active queries:

- Click left on the `Erase` button. This is equivalent to choosing `Erase This Query` from the `Erase` menu.

Figure 2-27 *The `Erase This Query` Command*



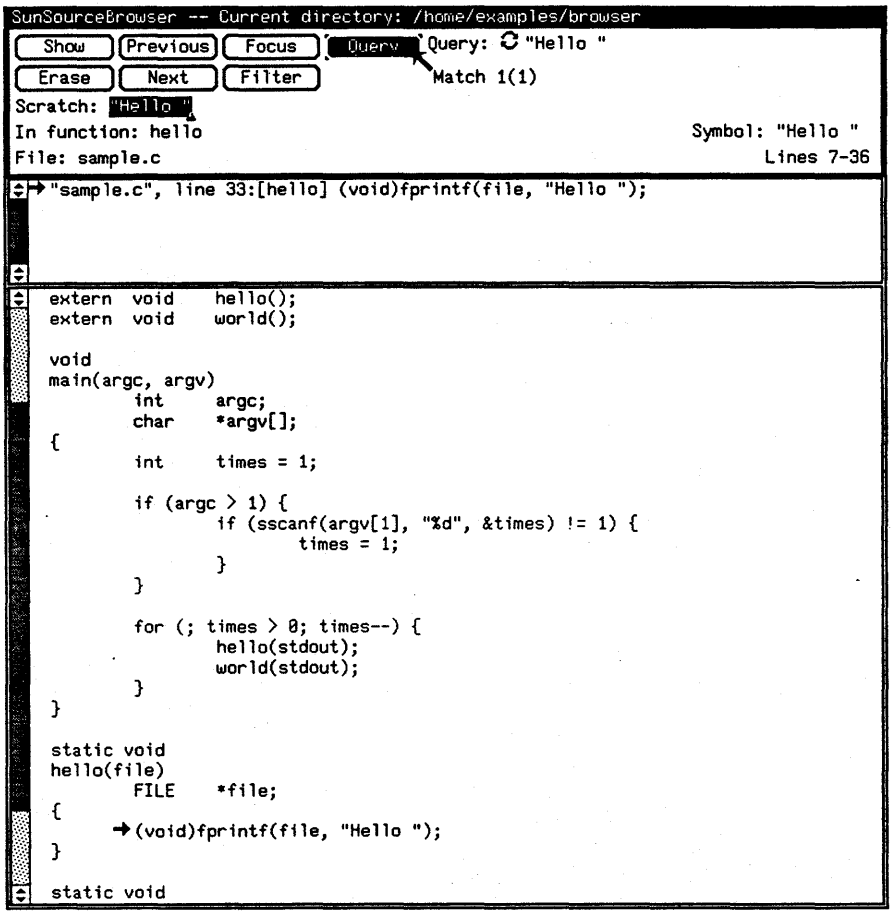
2.14 Searching for String Constants

SourceBrowser's default is to search for identifiers. To instruct SourceBrowser to search for string constants:

1. Select the string constant you want to search for. You can type the string constant in the Scratch field, enclosed in quotation marks, or you can select a string constant in the source subwindow or in another window.
2. Click left on the `Query` button.

For example, in Figure 2-28, the user instructed SourceBrowser to find all occurrences of "Hello".

Figure 2-28 *Searching for String Constants*



2.15 Using Wildcards in Queries

When issuing a query, you can use wildcards to specify a search pattern. When using wildcards, SourceBrowser gives you two options:

- shell-style patterns
- regular expressions

See `sh`(1) for information about shell-style pattern matching.

A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. In the following specification for regular expressions *character* means any character except NULL.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus "\", "[", " ." and sometimes "^", " *", and "$".
2. A ". " matches any character.

3. A "\" followed by any character except a digit or "()" matches that character.
4. A nonempty string s bracketed [s] (or [^s]) matches any character in (or not in) s. In s, "\" has no special meaning, and "]" may only appear as the first letter. A substring a-b, with a and b in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of the form shown in items 1 through 4 followed by "*" matches a sequence of 0 or more matches of the regular expression.
6. A regular expression, x, as in items 1 through 8, bracketed \(x) matches what x matches.
7. A "\" followed by a digit n matches a copy of the string that the bracketed regular expression beginning with the nth "\(" matched.
8. A regular expression of the form shown in items 1 through 8, x, followed by a regular expression of the form shown in item 7, y, matches a match for x followed by a match for y, with the x match being as long as possible while still permitting a y match.
9. A regular expression of the form shown in items 1 through 8 preceded by "^" (or followed by "$") is constrained to matches that begin at the left (or end at the right) end of a line.
10. A regular expression of the form shown in items 1 through 9 picks out the longest among the leftmost matches in a line.

SourceBrowser's default is shell-style expressions. If you want to use regular expressions, you need to change the SourceBrowser property sheet. See Section 4.2, "The SourceBrowser Property Sheet," for details.

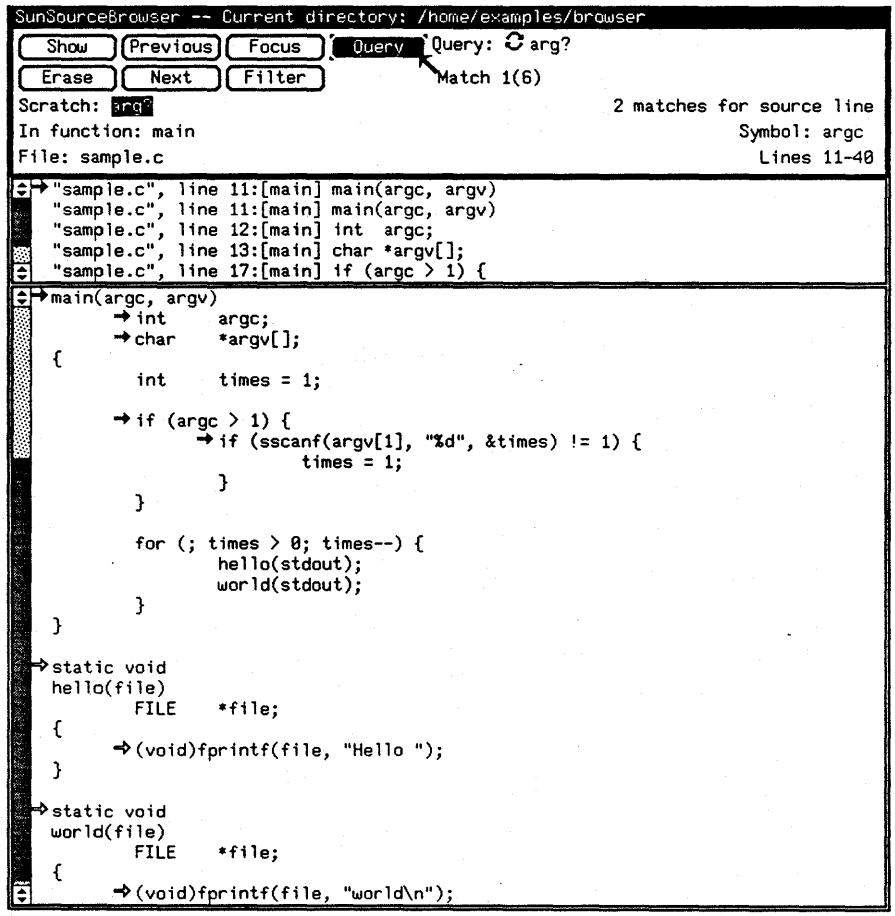
NOTE: *To limit the number of matches you receive when including wildcards in a query, it is often useful to use the* `Filter` *command to search for items based on how they are used in a program. See Section 2.16, "Using the Filter Command to Narrow a Search," for details.*

To include wildcards in a query:

1. Select the search pattern you want to search for. You can type the search pattern in the Scratch field or you can select a search pattern in another window.
2. Click left on the `Query` button. Or, if you typed the pattern in the Scratch field, press Return.

For example, in Figure 2-29, the user instructed SourceBrowser to find all instances of arg?

Figure 2-29 *Including Wildcards in a Query*



The Symbol item in the control subwindow displays the current identifier or string constant that matches the search pattern you have specified.

2.16 Using the `Filter` Command to Narrow a Search

The `Filter` command lets you search for symbols based on how they are used in a program.

You might, for example, want to limit your search to declarations of variables.

If you issue a filtered query that is identical to another active query, SourceBrowser will ignore your request and redisplay the existing query. Thus, if you want to issue a filtered query that uses the same symbol as an existing query, you must first delete the existing query using the `Erase Current Query` command.

You can have only one Filter setting for any query.

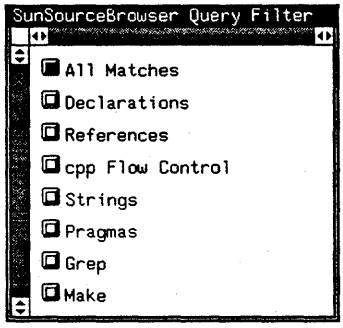
NOTE: When you choose a different current query, the settings in the Filter panel change to reflect the settings you used when issuing that query. Thus, if you are moving between queries with different filters, you will avoid confusion by leaving the Filter panel in view.

To use the `Filter` command to filter a query:

1. Click left on the `Filter` button.

The Filter panel is displayed. The content of the Filter panel is determined by the language(s) of the code you are browsing.

Figure 2-30 *The Filter Panel*

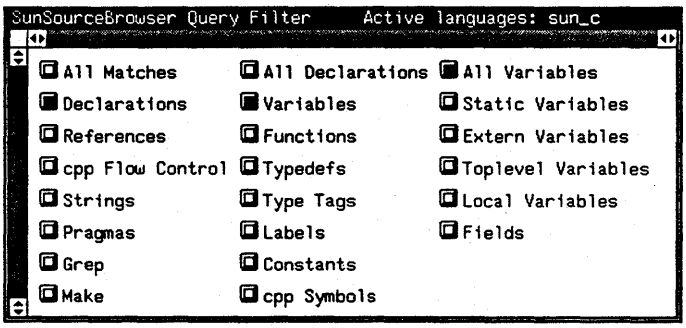


2. Click left on the menu item that describes the filter you want to use in your search.

Depending on the item you choose, the Filter panel may expand to include additional menu items which you can use to further narrow your search. In Figure 2-31, for example, the user instructed SourceBrowser to search only for symbols that are used as declarations of variables.

Note that after you have specified a filter, the SourceBrowser frame header displays a message telling you that filtering is turned on.

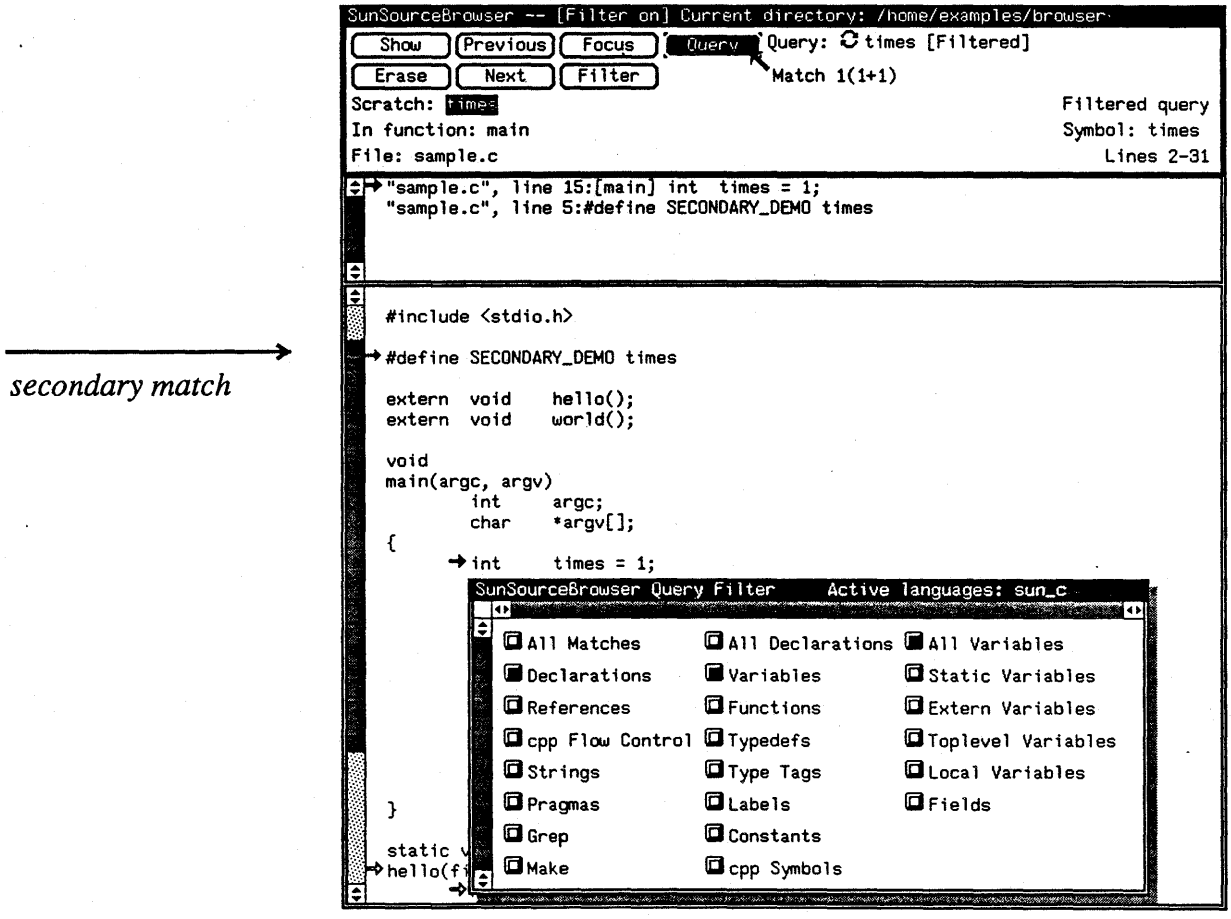
Figure 2-31 *The Filter Panel after Specifying a Filter*



3. If you want to close the Filter window, choose `Done` from the frame menu.
4. Activate the SourceBrowser window and issue a query.

In Figure 2-32, for example, the user instructed SourceBrowser to find all instances in which *times* is used as a variable declaration.

Figure 2-32 *Issuing a Filtered Query*



secondary match

Notice that the current match is marked as a secondary match, that is, as a solid arrow with a white line through it. See the next section for an explanation of secondary matches.

A Special Case: Filtering Functions in Macro Definitions

When SourceBrowser browses through source code, it is unable to determine how identifiers inside macro definitions are used. For example, in this code fragment from the sample program used in this chapter

```
#define SECONDARY_DEMO times

..

     int   times = 1
```

SourceBrowser does not know how *times* in the macro definition will be used. Consequently, when you use the `Filter` command to search for identifiers, as in Figure 2-32, SourceBrowser displays identifiers inside

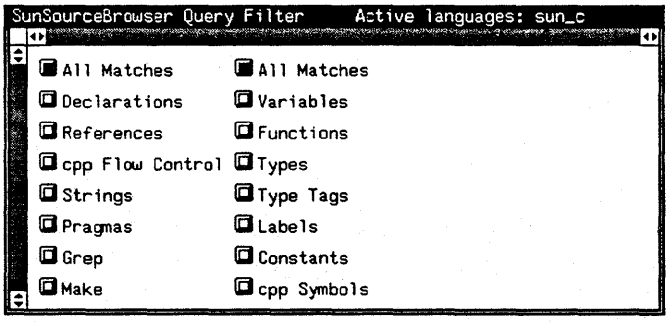
macro definitions as *secondary matches*. When it is the current match, a secondary match is marked in the match subwindow as a solid arrow with a line through it. Whenever SourceBrowser finds a secondary match, the total number of secondary matches is displayed in the Match item after the plus (+) sign. Thus in Figure 2-32, SourceBrowser has found one primary match and one secondary match.

Removing the Filter

To disable the filter and instruct SourceBrowser to find all occurrences of the symbol identified in your query:

1. Activate the Filter panel.
2. Select the `All Matches` item in the left column.

Figure 2-33 *Removing the Filter*



3. If you want to close the Filter window, choose Done from the frame menu.
4. Activate the SourceBrowser window and issue a query.

2.17 Using the `Focus` Command to Narrow a Search

The `Focus` command lets you focus your search on instances of specific classes of code, such as programs, functions, or libraries. For example, you might want SourceBrowser to limit its search to specified functions. SourceBrowser then searches for occurrences of the symbol in any of the functions you have activated.

Because SourceBrowser searches all source files described by the database in the current working directory, the `Focus` command is especially useful when you want to limit your query to selected files in a directory.

You can use the `Focus` command to focus your query on more than one class of code. For example, you could conduct a query that was limited to certain files and libraries.

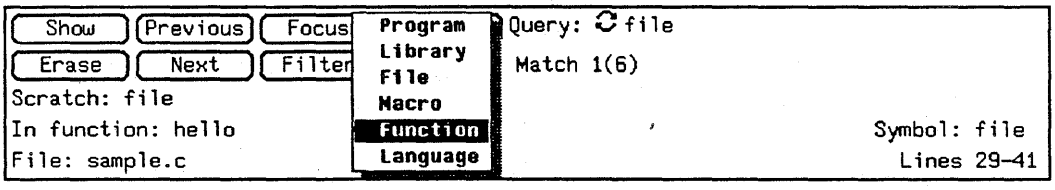
Once you use the `Focus` command to specify a focus, the focus remains in effect for all subsequent queries until you change the settings in each Focus panel.

If you issue a focused query that is identical to another active query, SourceBrowser will ignore your request and redisplay the existing query. Thus, if you want to issue a focused query that is the same as an existing query, you must first delete the existing query using the `Erase Current Query` command.

To issue a focused query:

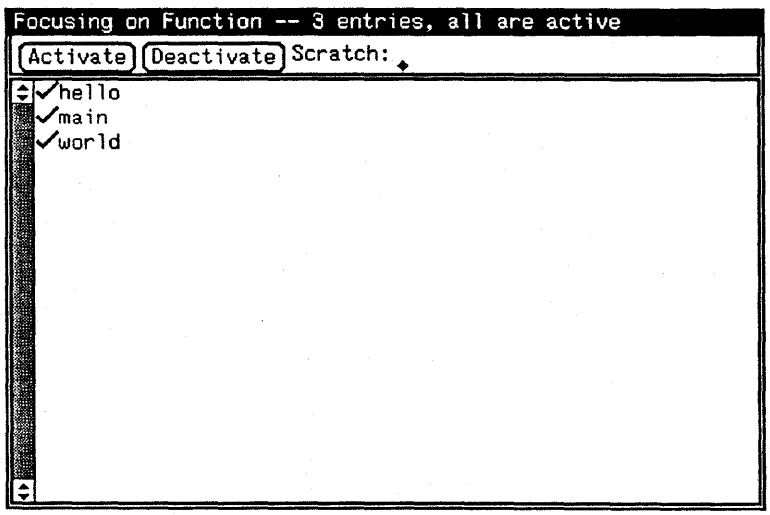
1. Hold down the right mouse button and choose the class of code you want to focus on from the `Focus` menu. The content of the `Focus` menu changes depending on the language(s) of the source code you are browsing.

Figure 2-34 *The `Focus` Menu*



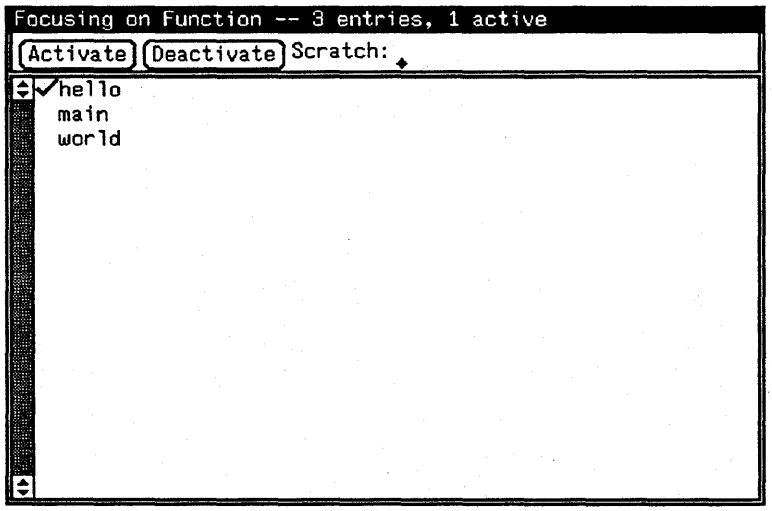
After a menu item is chosen, all of the available items for the selected class of code are displayed.

Figure 2-35 *The Focus Panel*



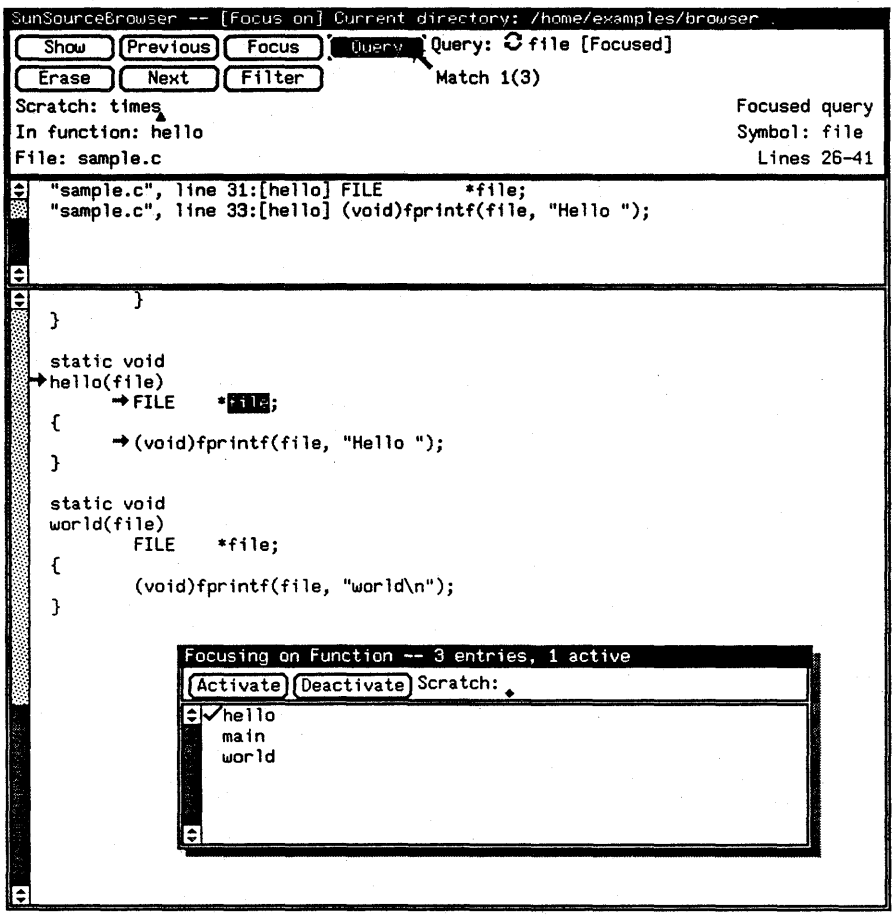
Activate or deactivate the items you want to use in your search using commands found on the `Activate` and `Deactivate` menus. For example, in Figure 2-36, the user limited the search to identifiers found in the *hello* function by choosing `Deactivate All Lines` from the `Deactivate` menu, selecting *hello;* then choosing `Activate Selected Lines` from the `Activate` menu. Note that after you have specified a Focus item, the SourceBrowser frame header displays a message telling you that focusing is turned on.

Figure 2-36 *The Focus Panel with One Function Activated*



2. If you want to close the Focus window, choose `Done` from the frame menu.
3. Activate the SourceBrowser window and issue a query. Note that SourceBrowser tells you when you are conducting a focused query.

Figure 2-37 *Issuing a Focused Query*



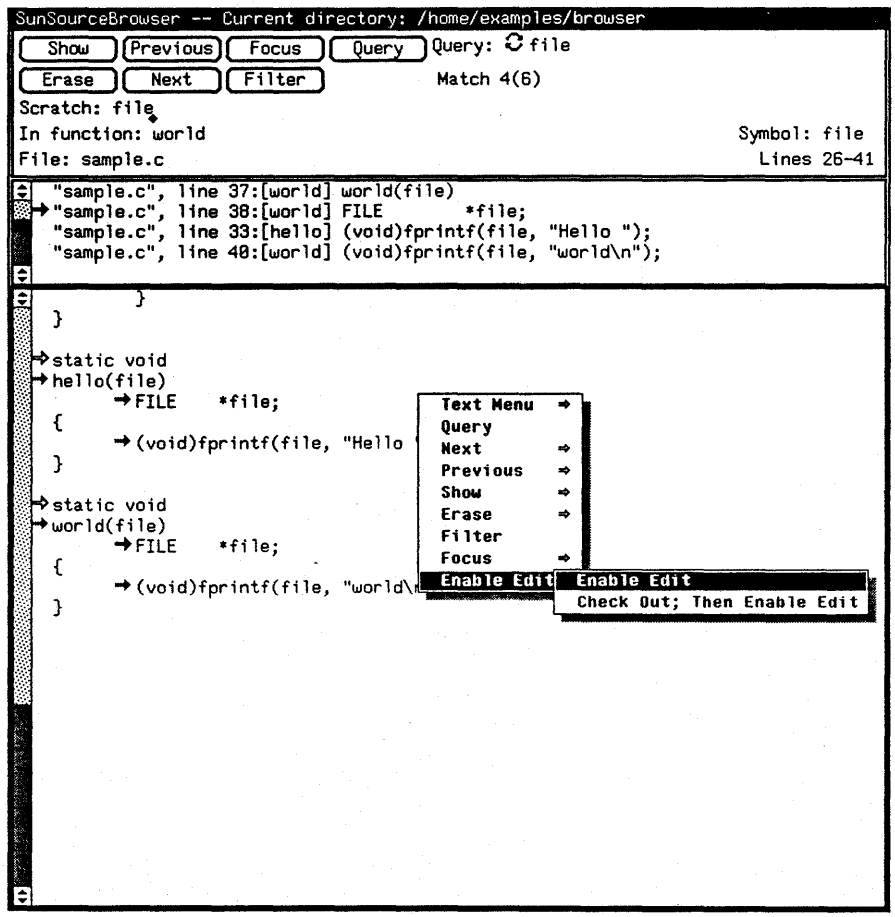
2.18 Editing Code from SourceBrowser

You can edit code without leaving SourceBrowser using SunView Text Edit's editing commands. For information about these commands, see the *SunView I Beginner's Guide.*

To edit code from SourceBrowser:

1. Position the pointer in the source subwindow and hold down the right mouse button to display the SourceBrowser menu.
   - If your code is not part of the SCCS (or VCS for NSE™) file control system, choose the `Enable Edit` item
   - If the code is part of a file control system, choose `Check Out; Then Enable` from the `Enable Edit` pull-right menu in the `SourceBrowser` menu

Figure 2-38 *The `Enable Edit` Menu*



The pointer changes to a pencil.

2. Edit the code.
3. Disable editing.
   - To save your changes, choose `Disable Edits` from the `SourceBrowser` menu
   - If you don't want to save your edits, choose `Ignore Changes` from the `Disable Edits` pull-right menu item in the `SourceBrowser` menu

NOTE: You cannot check files back in to SCCS or VCS from SourceBrowser. You must check them in manually.

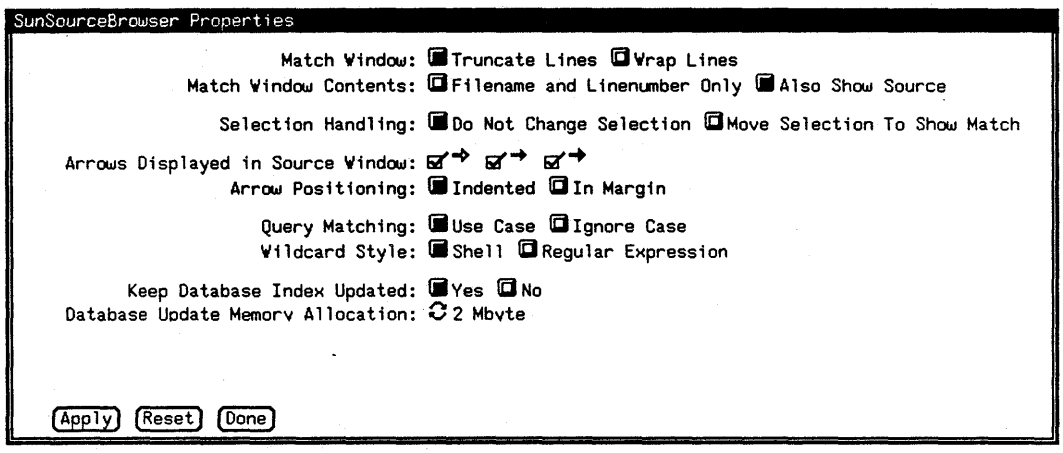
2.19 Customizing SourceBrowser

The SourceBrowser property sheet lets you customize SourceBrowser to suit your specific needs.

To change SourceBrowser properties:

1. Choose `Props` from the frame menu or press the L3 or Props key.

Figure 2-39 *The SourceBrowser Property Sheet*



2. Change the appropriate options.
3. Click left on the `Apply` button.
4. If you want to close the property sheet window, choose `Done` from the frame menu, or click the `Done` button.

For a complete description of each item in this panel, see Section 4.2, "The SourceBrowser Property Sheet.."

2.20 Using SourceBrowser with dbxtool

You may find it convenient to use SourceBrowser in conjunction with dbxtool. Specifically, you can use SourceBrowser to search for a variable; then, make the appropriate occurrence of that variable a breakpoint in dbxtool.

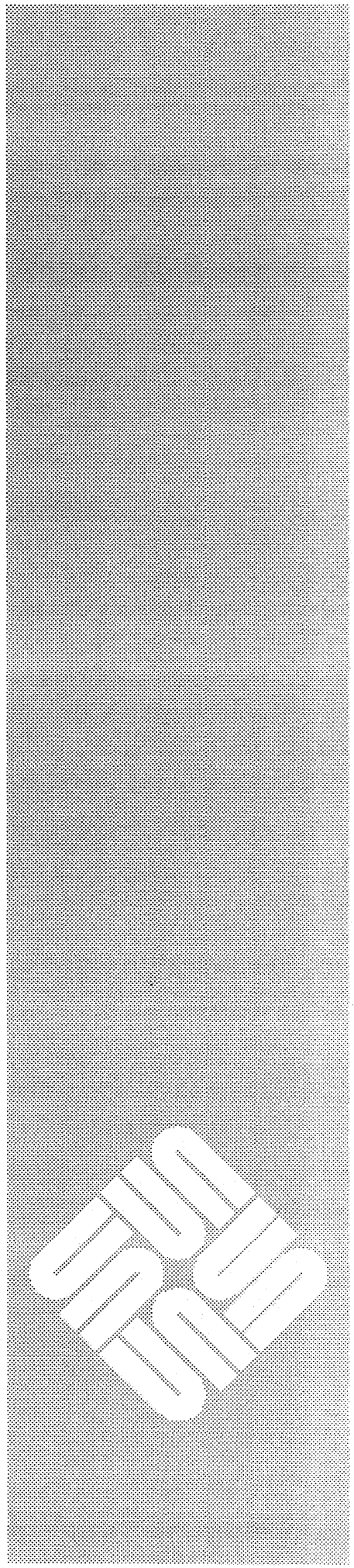
When using SourceBrowser with dbxtool, set the *Selection Handling* option in the SourceBrowser property sheet to *Move Selection to Show Match.* That way, each time you move to another match, that match will automatically be selected.

Begin by issuing a query that instructs SourceBrowser to find a variable that appears on the line at which you want to break. Then, use SourceBrowser commands to move between matches until the line of code at which you want to set a breakpoint is selected.

Finally, click the `stop at` button in dbxtool. The breakpoint will be set at the selected line of source code in the SourceBrowser window. For more information about dbxtool, see dbxtool (1).

3

Browsing Large Programs



3

Browsing Large Programs

Sun SourceBrowser responds to queries by searching in a specialized database that contains pertinent information about the files you are browsing. Knowledge of the layout of that database is especially important when you want to browse large programs whose source files are located in more than one directory.

This chapter introduces you to the SourceBrowser database and the commands you used to manipulate it. Specifically, this chapter describes:

- components of the SourceBrowser database
- using SourceBrowser from multiple machines
- creating and updating the SourceBrowser database
- information about .bd (browser data) files
- procedures for using symbolic links to organize databases
- options for browsing databases contained in more than one directory
- the .sbinit file, a textfile used by SourceBrowser to obtain information about the database structure and to improve SourceBrowser's performance when browsing large programs
- information about database file protection
- performance considerations

NOTE: This chapter contains very general information that is likely to change in the future. Note too that the description of the structure of the .sb (SourceBrowser) subdirectory explained later in this chapter has been simplified. In reality, the structure of the subdirectory is more complex than the description included here.

3.1 Overview: The SourceBrowser Database

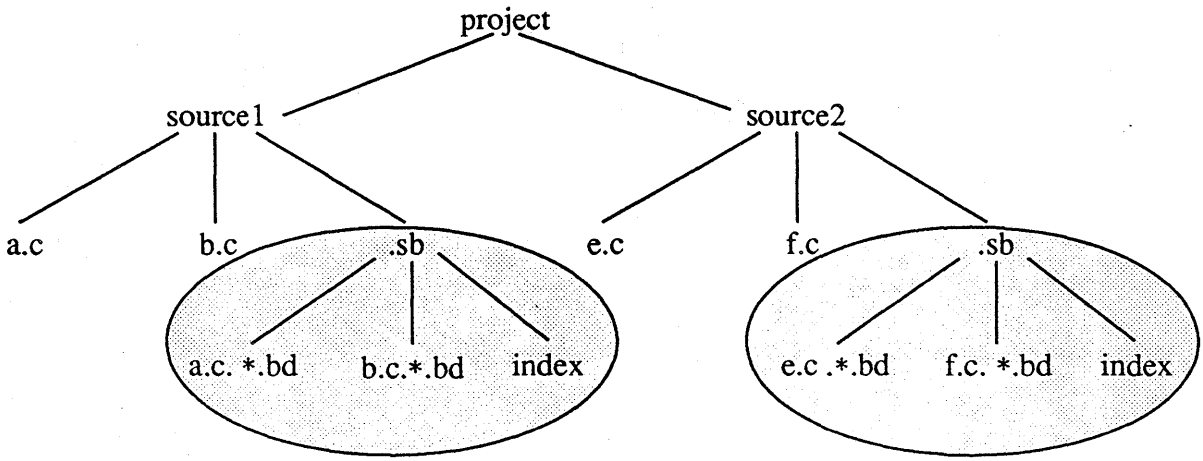
The SourceBrowser database consists of two parts:

- one file with an extension of .bd (browser data) for each source file for which you have generated SourceBrowser information. Browser data files are created by the compiler when you use the `-sb` option.
- an index file created by SourceBrowser which SourceBrowser uses to locate information in the .bd files

The .bd files and the index file are stored in the .sb (SourceBrowser) subdirectory. Unless you specify otherwise, the .sb subdirectory is created in the current working directory from which the compiler is executed.

For example, in Figure 3-1, SourceBrowser databases are created in the .sb subdirectories in the source1 and source2 directories.

Figure 3-1 *The SourceBrowser Database*



NOTE: The asterisk (*) in each .bd file is a place holder for a random string that is a hash value computed from the contents of the file.

NOTE: The programatic interface to the SourceBrowser database is available to third party compiler writers.

NOTE: The SourceBrowser database is architecture independent. Thus, you can build a database on a Sun-3™ machine and then browse it on a Sun-4 machine™.

3.2 Browsing From Multiple Machines

Each source file compiled with the `-sb` option must have the same absolute path regardless of the machine from which it is referenced. That's because the .bd (browser data) files that are created when you run the compiler contain the absolute path for each source file. This information is used to display the source file when a query is issued. If the absolute path is not uniform across machines, SourceBrowser won't be able to display the source files when it responds to a query.

If SourceBrowser users are working on different machines, make sure that all source files are mounted at the same mount point.

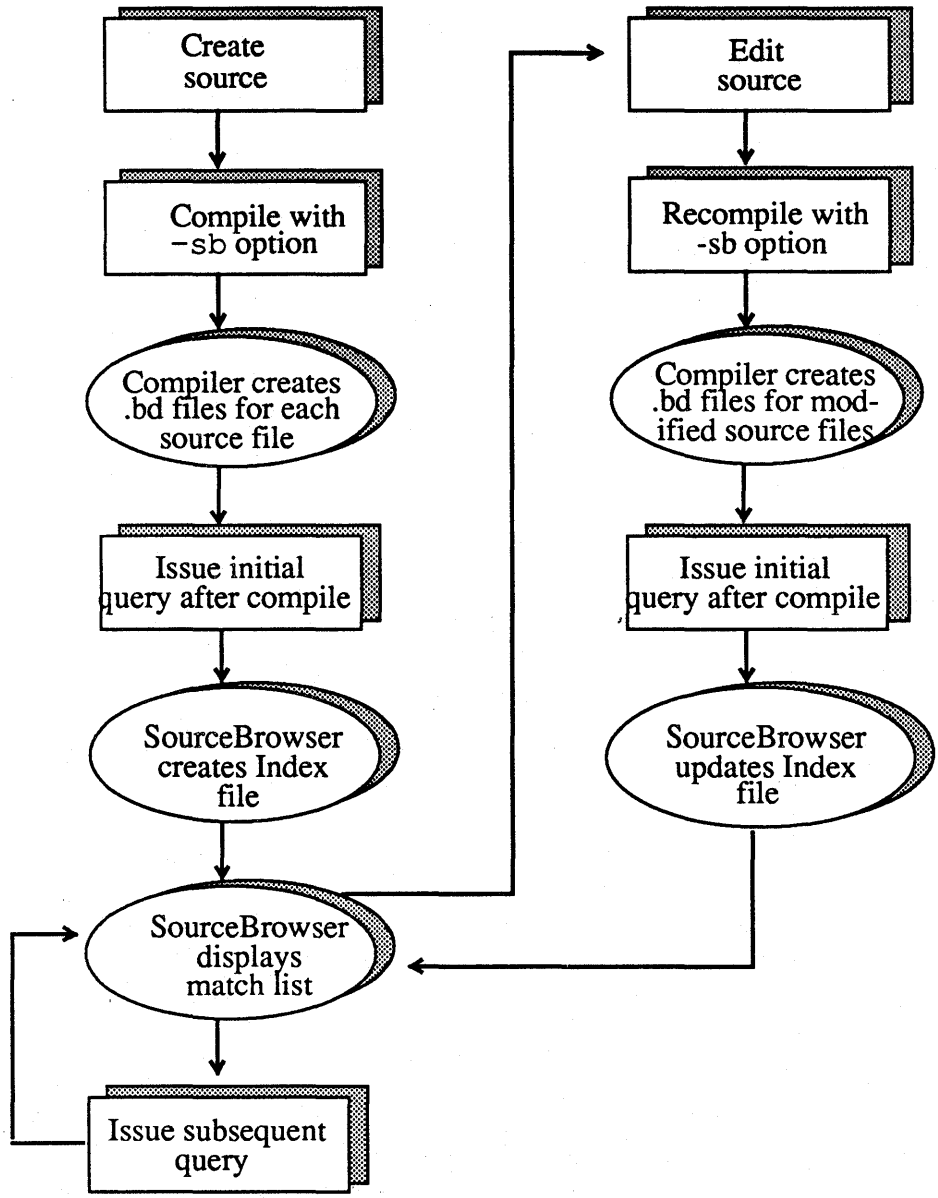
3.3 Creating and Updating the Database

The first time you compile a source file with the `-sb` option, the compiler creates a .bd file for each file that you compiled. Then, prior to responding to your initial query following a compilation, SourceBrowser creates an index file which it uses to locate information in the .bd files.

Each subsequent time that you compile a source file with the `-sb` option, new .bd files are created for source files that have been modified since the last compilation. When you issue your initial query following the compilation, SourceBrowser updates the index file before it responds to your query.

Figure 3-2 illustrates the process.

Figure 3-2 *Creating and Updating the Database*



3.4 More about .bd Files

This section describes the process of creating .bd files when you compile, link, or build a library. Note that this section presents very general information.

Compiling and Browser Data Files

When you compile with the -sb option, browser data (.bd) files are created for each source file. They are stored in the current working directory from which the compiler was executed.

If it is the first time the source files have been compiled with the `-sb` option, the compiler creates a .sb subdirectory as well as a .bd file for each source file. For example, in Figure 3-3, the C compiler creates a c.c.*.bd file while the FORTRAN 77 compiler creates an f.f.*.bd file.

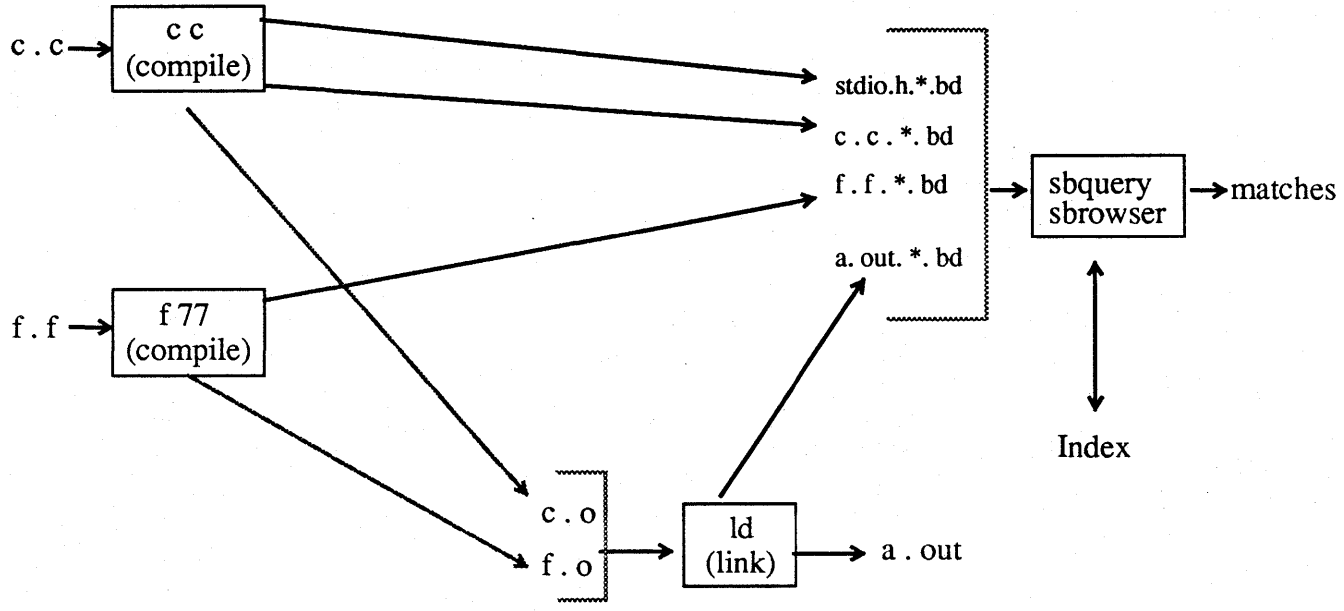
If it is a recompile, .bd files are created for those files that have been changed since the last compile.

NOTE: To generate correct .bd information, you must issue a compile run that does not generate any warnings or error messages.

Linking and Browser Data Files

When you link files that have been compiled with the `-sb` option, an additional .bd file is created by the linker that lists the object files used to construct the program. For example, in Figure 3-3, the linker creates an a.out.*.bd file.

Figure 3-3 *Compiling with the -sb Option*



Libraries and Browser Data Files

When you build a library with files that were compiled with the -sb option, a .bd file is created that lists the object files contained in the library.

3.5 Working with Multiple Directories

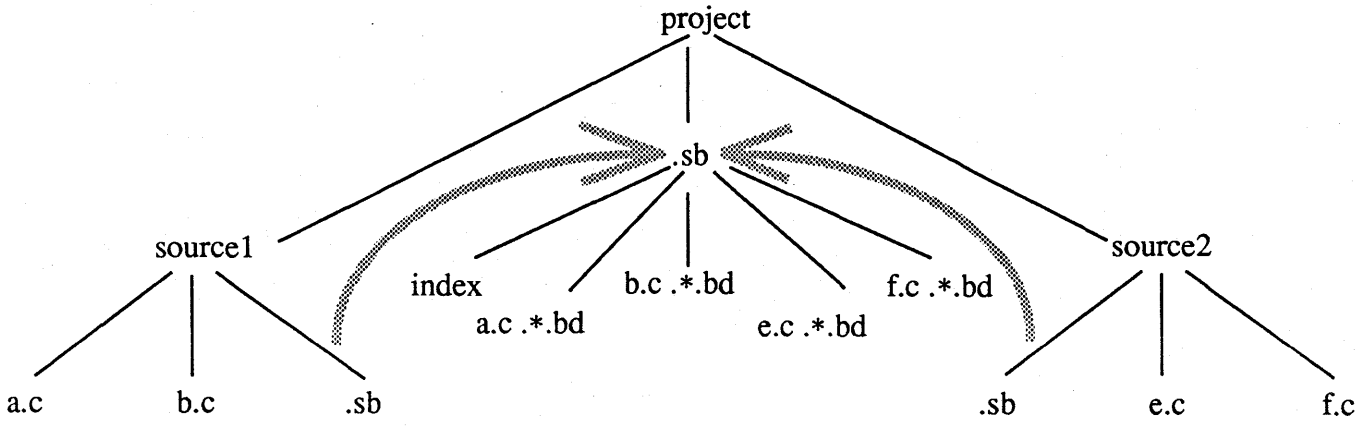
SourceBrowser's default is to build its database in the current working directory, then search in that database when you issue a query. If you want to work with source files whose database information is stored in multiple directories, you have the following options:

- You can create a common database that is used by all directories from which the compiler will be executed. To do this you must install a symbolic link to the common directory. See Section 3.6, "Installing a Symbolic Link," for details.
- You can browse multiple databases. To do this, you must include the `import` command in a text file called .sbinit. See Section 3-7, "Using .sbinit," for details.
- You can store SourceBrowser databases in directories other than the current working directory. To do this, you must include the `export` command in a text file called .sbinit. See Section 3-7, "Using .sbinit," for details.

3.6 Installing a Symbolic Link

To browse databases stored in multiple directories, you can install a symbolic link to a common .sb directory from all directories in which the compiler will be executed. Figure 3-4 shows a conceptual view of installing a symbolic link to a common subdirectory.

Figure 3-4 *Installing a Symbolic Link*



Use one of the following options to create a symbolic link to a common .sb subdirectory:

- Execute this command:

```
ln -s path/.sb
```

- Include the following code fragment in your makefile:

```
.INIT: .sb

.sb:
        -ln -s path/.sb
```

Note that *path* can be relative or absolute.

It often is useful to consolidate several databases into a single database. Suppose you wanted to move the databases in the .sb subdirectories in the source directories in Figure 3-1 to the project directory, as in Figure 3-4. Figure 3-5 shows the steps you would follow to complete that process.

Figure 3-5 *Consolidating Databases*

```
venus% cd project
venus% (cd source1 ; tar cf - .sb) | tar xfB -
venus% (cd source2 ; tar cf - .sb) | tar xfB -
venus% rm -rf source1/.sb source2/.sb
venus% ln -s ../.sb source1/.sb
venus% ln -s ../.sb source2/.sb
venus% rm -f .sb/Index
```

The second and third lines of code use the `tar` command to move the .sb subdirectories in source1 and source2 up one level. The fourth line of code removes the .sb subdirectories in source1 and source2. The fifth and sixth lines of code create symbolic links to the .sb subdirectory in the project directory. The final line removes the Index file in the .sb subdirectory. Then, when you issue a query, a new Index file will be created in the .sb subdirectory in the project directory.

3.7 Using .sbinit

The .sbinit file is a text file used by SourceBrowser to obtain information about the database structure. The .sbinit file is limited to these commands:

- The `import` command: reads databases in directories other than the current working directory used by SourceBrowser.

- The `export` command: writes databases associated with specified source files to directories other than the current working directory used by SourceBrowser and the compiler. When you include the `export` command in the .sbinit file, there is an implied `import` command.
- The `split` command: speeds up SourceBrowser's performance by only updating relevant portions of the database used by SourceBrowser when the database is updated following a recompilation.

These commands are explained in subsequent sections of this chapter.

NOTE: SourceBrowser's default is to look in the current working directory for the .sbinit file. If the .sbinit file is in another directory, or if you want to give the .sbinit file a different name, set the environment variable SUN_SOURCE_BROWSER_INIT_FILE to the appropriate pathname. Setting SUN_SOURCE_BROWSER_INIT_FILE also causes compilers to turn on the `-sb` *option.*

The `import` Command

The `import` command tells SourceBrowser which databases to read each time it performs a query.

If you don't include the `import` command or the `export` command in the .sbinit file, SourceBrowser only browses the database in the current working directory. (The `export` command is explained in the next section.)

In contrast to installing a symbolic link, which creates a common database, the `import` command enables you to retain separate databases. For example, you may want to set up administrative boundaries so that programmers working on Project A cannot write into directories for Project B and vice versa. In that case, Project A and Project B will each need to maintain its own SourceBrowser database, both of which can then be read but not written by programmers working on the other project.

The `import` command has the form:

`import` *<path>*

where *path* is the path to the directory that contains the .sb subdirectory containing the database you want to import.

For example, in Figure 3-6, if your current working directory is /project/source1, and you want to instruct SourceBrowser to read the SourceBrowser database in source2, the `import` command in source1's .sbinit file would be

```
import /project/source2
```

or

```
import ../source2
```

Similarly, if your current working directory is /project /source2, and you want to instruct SourceBrowser to read the SourceBrowser database in source1, the `import` command in source2's .sbinit file would be

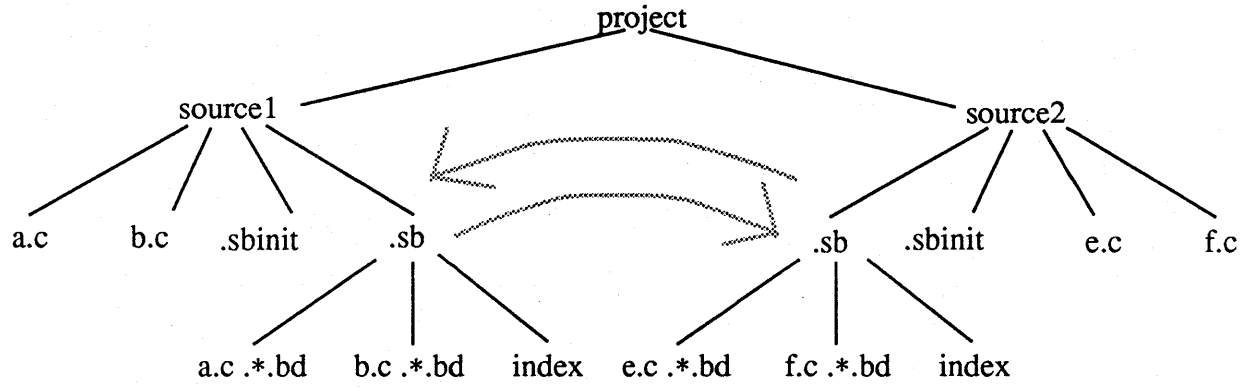
```
import /project/source1
```

or

```
import ../source1
```

When you recompile and issue a subsequent query, the databases for both source1 and source2 are updated as appropriate if they can be written.

Figure 3-6 *Using the* `import` *Command*



The `export` Command

You use the `export` command to cause the compiler to store specified browser data (.bd) files in a directory other than the current working directory. This enables you to save disk space by placing .bd files associated with identical files, such as #include files from /usr/include, in a single SourceBrowser database while still retaining distinct databases for individual projects. If you don't include a `export` command in the .sbinit file, the compiler writes all .bd files in the current working directory.

The `export` command has this form:

`export <prefix> into <path>`

Whenever the compiler processes a source file whose absolute path starts with *prefix*, the resulting browser data (.bd) file will be stored in `<path>/.sb` .

Note that the `export` command provides instruction for *writing* .bd files while the `import` command provides instructions for *reading* .bd and index files.

In Figure 3-7, to place the .bd file and index file created for files from /usr/include in a .sb subdirectory in the sys subdirectory, the `export` command in the .sbinit file for source1 would be

```
export /usr/include into /project/sys
```

If your configuration had included a source2 directory with a .sbinit file containing the same `export` command shown above, you would save disk space because, instead of creating two identical stdio.h.*.bd files, the compiler would create a single stdio.h.*.bd file in the sys subdirectory.

Note that the .sbinit file contains an implied

```
export / into .
```

that instructs the compiler to put .sb files created by source files not explicitly mentioned by a `export` command in the current working directory. Thus, in Figure 3-7, the .bd files associated with a.c and b.c are placed in the .sb subdirectory in the source1 directory.

When you include the `export` command in the .sbinit file, there is an implied `import` command which causes SourceBrowser to read each database that you have instructed the compiler to create. Thus, based on the configuration in Figure 3-7, SourceBrowser will search in the database in the sys subdirectory, as well as in the database in the source1 directory, each time you issue a query.

Figure 3-7 *Using the* `export` *Command for Shared System Files*

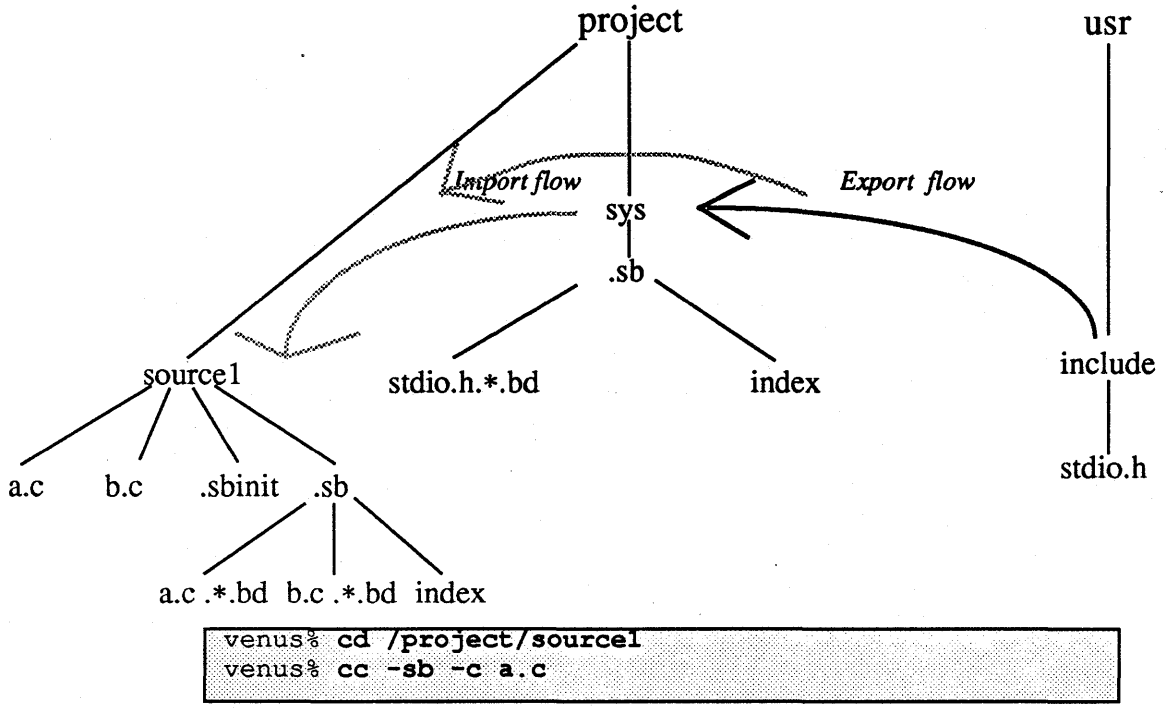


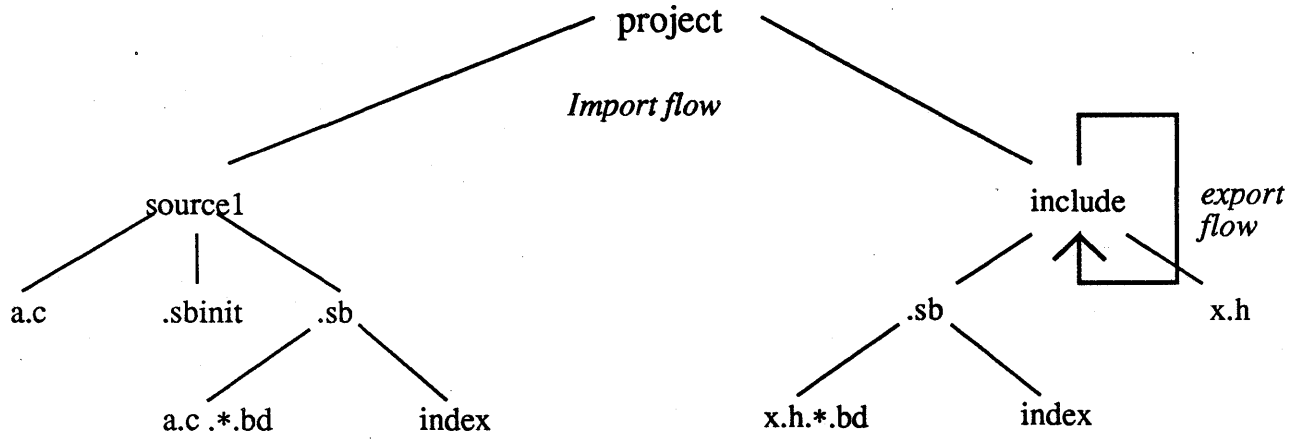
Figure 3-8 illustrates how to use the `export` command to store .bd files in a common subdirectory when the include file is located in a non-standard location. In this example, to place the .bd file and index file created for files from /project/include in a .sb subdirectory in the /project/include subdirectory, the `export` command in the .sbinit file for source1 would be

```
export /project/include into /project/include
```

Here again, using the `export` command will save disk space when your program includes multiple references from many different directories to the same include file.

Because the `export` command includes an implied `import` command, SourceBrowser will read the database in the include subdirectory as well as the database in the source1 subdirectory each time you issue a query.

Figure 3-8 *Using the* `export` *Command for Shared Project Include Files*




```
venus% cd /project/source1
venus% cc -sb -I/project/include -c a.c
```

NOTE: If a .sbinit files includes multiple `export` *commands, they are scanned in the same order that they are encountered in the .sbinit file. Thus,* `export` *commands should be arranged from most specific to least specific.*

The `split` Command

The purpose of the `split` command is to improve performance when working with large numbers of programs.

The `split` command, which must be included in the .sbinit file, has this form:

`split <`*size*`>`

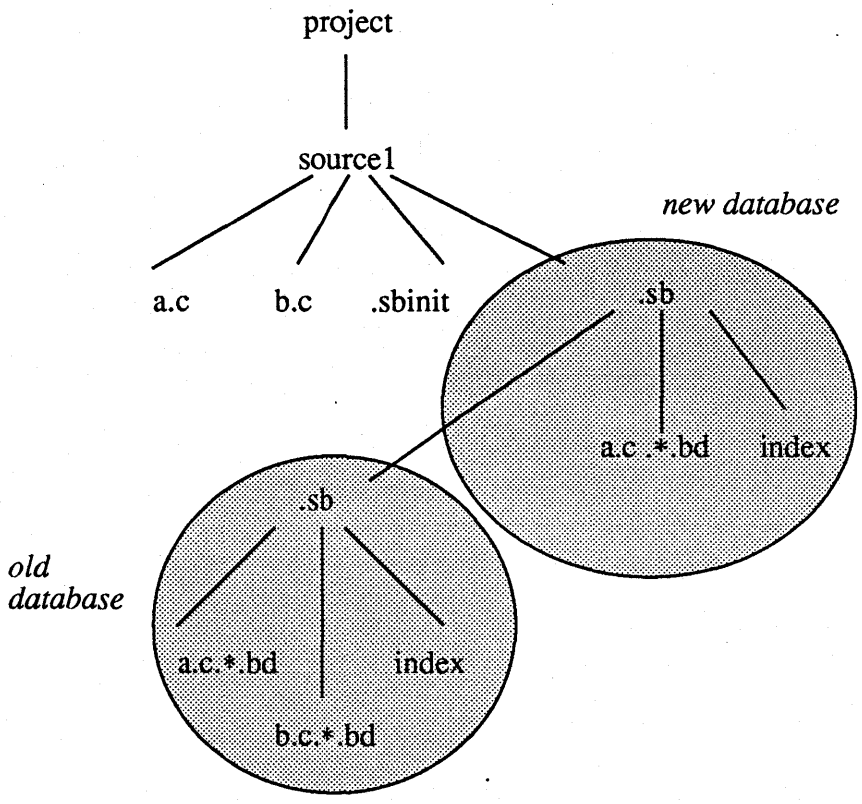
where <*size*> is the size in bytes of the database index when the database will be split by SourceBrowser.

The `split` command causes the SourceBrowser database to be split into a "new" and an "old" database whenever the database's index file exceeds a specified number of bytes. Then, when you recompile, SourceBrowser places those .bd files that have changed since the last compilation into the new database, leaving the old database unchanged.

When you issue the initial query following compilation, a new index file is created in the new database, while the index file in the old database remains unchanged. This greatly speeds up SourceBrowser's performance, since the time it takes to build the index file is proportional to the number of .bd files that require indexing. Consequently, it takes far less time to build the small index file in the new database than to build one large index file for the entire database.

For example, in Figure 3-9, the database has exceeded the number of bytes specified by the `split` command in the .sbinit file in source1; consequently, the old database has moved down a subdirectory level and a new database is created in its place. In this example, the only source file that has been modified since the split is a.c; thus, the new database contains a.c.*.bd and an index file.

Figure 3-9 *Using the `split` Command*



If the new database reaches the number of bytes specified by the size parameter of the `split` command, the old and new database files are merged, then split again.

3.8 File Protection

UNIX® file protection modes control whether the SourceBrowser database will be updated when you recompile. If you have Read/Write access, the database will be updated. If you have Read Only access, the database will not be updated.

3.9 Performance Considerations

This section describes those factors that determine the speed with which SourceBrowser builds the index file and conducts queries. Note that this is very general information.

Building the Index File

The time it takes SourceBrowser to build the index file the first time you query is proportional to the following:

- The number of .bd files
- The number of symbols in the .bd files
- The size of the resulting index file

The time it takes SourceBrowser to rebuild the index file following recompilation is proportional to the following:

- the number of new .bd files
- the sum of the number of symbols in all .bd files
- the size of the index file

Conducting a Query

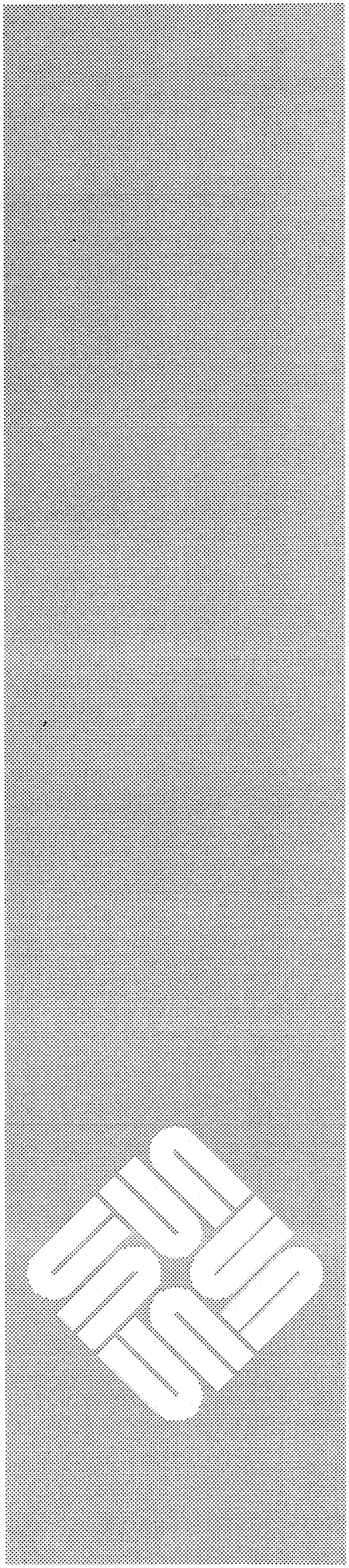
The time required to conduct a query is determined by the number of .bd files that contain matches and the number of source files that contain matches.

If the `Match Window Contents` option in the property sheet is set to `Also Show Source`, the speed of the query is also affected by the number of source files that contain matches. You can halve the time required to respond to a query by setting this option to `Filename and Linenumber Only`.

The time required to conduct a query that includes wildcards is based on the time it takes to conduct a query without wildcards as well as the size of the index file.

4

Window Command Reference



4

Window Command Reference

This chapter provides reference information about Sun SourceBrowser window commands. Specifically, it includes

- a description of each SourceBrowser menu command with its command line equivalent
- a description of each option on the SourceBrowser property sheet, with its command-line equivalent

4.1 SourceBrowser Menu Commands

SourceBrowser commands can be activated either by clicking the appropriate button in the control subwindow or choosing the appropriate menu item in the control subwindow, or by accessing the SourceBrowser menu from the match subwindow or the source subwindow.

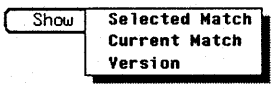
Clicking left on a button in the control subwindow is equivalent to choosing the first option in the menu behind the button.

NOTE: *The only SourceBrowser-specific menu commands that are not accessible through menus in the control subwindow are the* `Enable Edit` *and* `Disable Edit` *commands. These commands must be activated by displaying the SourceBrowser menu in the source subwindow. See Section 2.18, "Editing Code from SourceBrowser," for details.*

The following sections describe each command as it appears in the control subwindow.

The Show Menu

Holding down the right mouse button when the pointer is positioned on the `Show` button activates a menu with the following items.



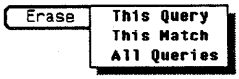
`Selected Match`: (This is the default.) Makes the selected match in the match subwindow the current match and displays the appropriate line of code in the source subwindow. This command is useful when you

want to scroll through the match subwindow until you find a match that you want to make the current match.

`Current Match`: Displays the current match in the source subwindow. This command is useful when you have scrolled through the source subwindow and want to redisplay the current match in that subwindow.

`Version`: Displays SourceBrowser's current version number. This command is equivalent to the `-version` command line option.

The `Erase` Menu



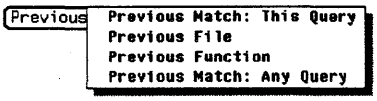
Holding down the right mouse button when the pointer is positioned on the `Erase` button activates a menu with the following items.

`This Query`: (This is the default.) Erases the current query. Once a query is erased, it is removed from the Query menu. Using this command reclaims memory.

`This Match`: Erases the current match. This command is useful when you are reviewing matches and only want to retain those matches that are of interest to you.

`All Queries`: Erases all active queries. Using this command reclaims memory.

The `Previous` Menu



Holding down the right mouse button when the pointer is positioned on the `Previous` button activates a menu with the following items:

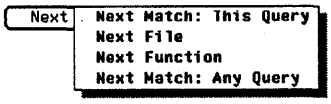
`Previous Match:This Query`: (This is the default.) Makes the previous match found by the active query the current match and displays it in the match subwindow and the source subwindow.

`Previous File`: For the current query, makes the last match that occurs in the previous file that contains a match the current match and displays that match in the match subwindow and the source subwindow.

`Previous Function`: For the current query, makes the last match that occurs in the previous function that contains a match the current match and displays that match in the match subwindow and the source subwindow.

`Previous Match: Any Query`: Makes the previous match in the source subwindow the current match. If the new current match was found by another query, that query becomes the active query and the Query item in the control subwindow changes accordingly.

The Next Menu



Holding down the right mouse button when the pointer is positioned on the `Next` button activates a menu with the following items:

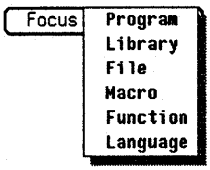
`Next Match: This Query`: (This is the default.) Makes the next match found by the active query the current match and displays it in the match subwindow and the source subwindow.

`Next File`: For the current query, makes the first match that occurs in the next file that contains a match the current match and displays that match in the match subwindow and the source subwindow.

`Next Function`: For the current query, makes the first match that occurs in the next function that contains a match the current match and displays that match in the match subwindow and the source subwindow.

`Next Match: Any Query`: Makes the next match in the source subwindow the current match. If the new current match was found by another query, that query becomes the active query and the Query item in the control subwindow changes accordingly.

The Focus Menu

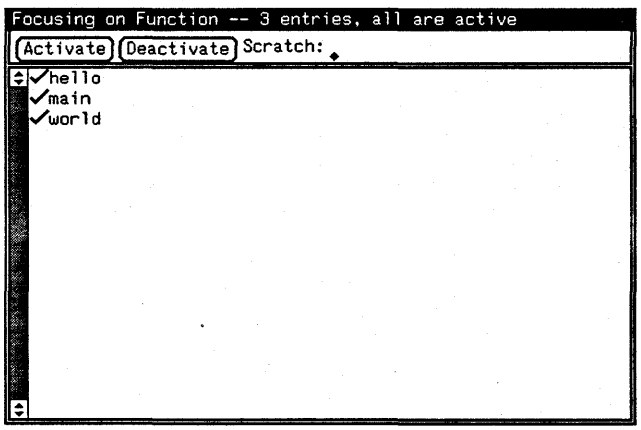


Holding down the right mouse button when the pointer is positioned on the `Focus` button activates a menu with items representing classes of code.

The content of this menu changes depending on the language(s) used to write the source files you are browsing.

After you choose an item from the Focus menu, all of the available items from the class of code you have selected are displayed in a Focus window. Use the `Activate` and `Deactivate` menus described below to activate and deactivate the items you want to use in your focused query. Note that when you first open this window, all items are activated.

Figure 4-1 *The Focus Window*

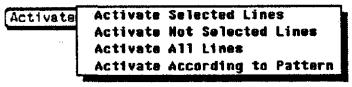


See Section 2.17, "Using the Focus Command to Narrow a Search," for more about the `Focus` command.

The Focus window contains two menus: `Activate` and `Deactivate`.

The `Activate` Menu

The `Activate` menu in the Focus window is used to activate specific code items. The `Activate` menu contains these items:



`Activate Selected Lines` (This is the default.) Activates selected items.

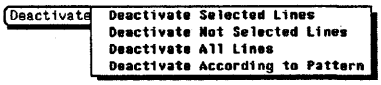
`Activate Not Selected Lines` activates all items that are not selected.

`Activate All Lines` activates all items.

`Activate According to Pattern` activates items based on the search pattern you have selected.

The `Deactivate` Menu

The `Deactivate` menu in the Focus window is used to deactivate specific code items. The `Deactivate` menu contains these items:



`Deactivate Selected Lines` (This is the default.) Deactivates selected items.

`Deactivate Not Selected Lines` deactivates all items that are not selected

Command-Line Equivalent

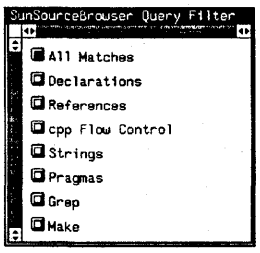
`Deactivate All Lines` deactivates all items.

`Deactivate According to Pattern` deactivates items based on the search pattern you have selected.

This item is equivalent to using the `-in_*` command-line option to specify a focusing option. To receive a list of command-line focusing options, use the `-help_focus` option. See `-help_focus` in Section 5.4, "Command-Line Options," for details.

The `Filter` Command

Clicking left on the `Filter` button displays the Filter panel.



You use the Filter panel to conduct a search based on how symbols are used in a program.

The content of the Filter panel changes depending on the language(s) used to write the files you are browsing. See Section 2.16, "Using the Filter Command to Narrow Your Search" for more about the `Filter` command.

Command-Line Equivalent

To receive a list of filtering options from the command line, use the `-help_filter` option followed by the language of the source code you are browsing. See `-help_filter` in Section 5.4, "Command-Line Options," for details.

The `Query` Button

Clicking left on the `Query` button causes SourceBrowser to search for the selected symbol.

You also can issue a query by typing the symbol in the Scratch field and then pressing Return.

The `Update` Button

Clicking left on the `Update` button causes SourceBrowser to update the Index file.

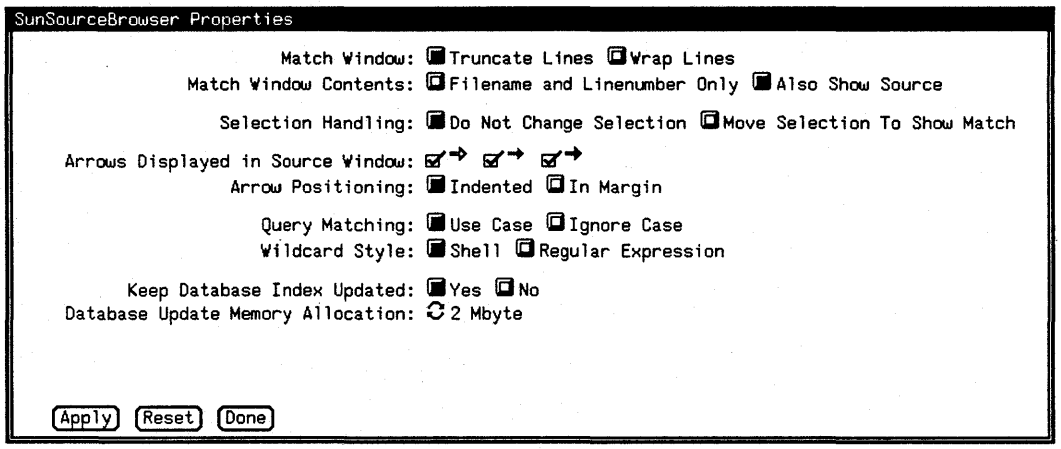
This button only appears if you have set the `Keep Database Index Updated` item in the SourceBrowser property sheet to `no`. If that is the case, you need to click the `Update` button whenever you want SourceBrowser to update the database.

If you have set this item to `yes`, SourceBrowser updates the database, then processes your query when you issue your initial query following a compilation or recompilation.

4.2 SourceBrowser Property Sheet

To display the SourceBrowser property sheet, choose Props from the frame menu or press the F3 or Props button.

Figure 4-2 *The SourceBrowser Property Sheet*



To hide the Property sheet, but retain the current settings, click `Done`.

The items in the SourceBrowser property sheet are explained in the following sections.

Match Window

`Truncate Lines` truncates lines of code in the match subwindow. This is the default. `Wrap Lines` wraps lines of code so that the complete line of source code is displayed.

Match Window Contents

`Filename and Line Number Only` does not display the source code line in the match subwindow. Choosing this option causes a fifty percent improvement in performance when conducting queries. `Also Show Source` displays the source code line, along with the filename and line number, in the match subwindow. This is the default.

Command-Line Equivalent

See `-no_source` in Section 5.4, "Command-Line Options," for information about the equivalent command-line option.

Selection Handling

`Do Not Change Selection` does not move the selection bar to the current match in the source subwindow when a new current match is displayed. This is the default. `Move Selection to Show Match` selects the current match in the source subwindow when a new current match is displayed.

NOTE: *This item does not take effect until you choose a different match as the current match.*

Arrows Displayed in Source Window

Determines which arrows are displayed in the source subwindow.

- A black arrow indicates the current match
- A gray arrow indicates a match other than the current match that has been found by the current query
- A hollow arrow indicates a match found by a query other than the current query

If you click the black arrow box, only black arrows will be displayed. If you click the gray arrow box, gray and black arrows will be displayed. If you click the hollow arrow box, all three kinds of arrows will be displayed.

Arrows Positioning

`Indented` indents arrows so that they line up with the first line of source code. This is the default. `In margin` places arrows in the left margin.

Query Matching

`Ignore Case` causes SourceBrowser not to look at case when searching for symbols in the source code that match the query symbol. `Use Case` causes SourceBrowser to only make a match when the case of the symbol matches the case of the symbol in the source code. This is the default.

Command-Line Equivalent

See `-no_case` in Section 5.4, "Command-Line Options," for information about the equivalent command-line option.

Wildcard Style

`Shell` informs SourceBrowser that you are using shell-style patterns when using wildcards in a query. This is the default. `Regular Expressions` informs SourceBrowser that you are using regular expressions when issuing a query that includes wildcards.

See Section 2.15, "Using Wildcards in Queries," for more about using wildcards in a query and about pattern matching using regular expressions. See sh(1) for information about shell-style pattern matching.

Command-Line Equivalent

See `-reg_expr` in Section 5.4, "Command-Line Options," for information about the equivalent command-line option.

Keep Database Index Updated

`Yes` instructs SourceBrowser to update the index file before the initial query after a compilation is processed. This is the default. `No` instructs SourceBrowser not to automatically update the database. When you choose `No`, a new button, `Update`, is added to the control subwindow. You click left on this button each time you want to update the index file.

Command-Line Equivalent

See `-no_update` in Section 5.4, "Command-Line Options," for information about the equivalent command-line option.

Database Update Memory Allocation

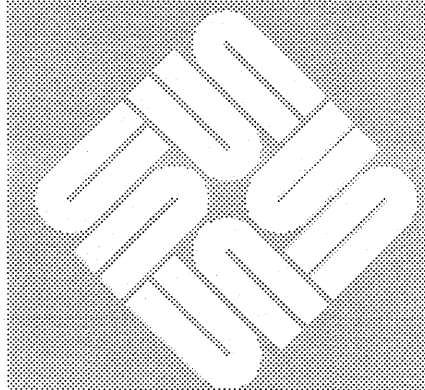
This is a cycle item that tells SourceBrowser the approximate amount of memory that should be allocated before SourceBrowser uses temporary files when building the index file.

Command-Line Equivalent

See `-max_memory` in Section 5.4, "Command-Line Options," for information about the equivalent command-line option.

5

Command-Line Reference



5

Command-Line Reference

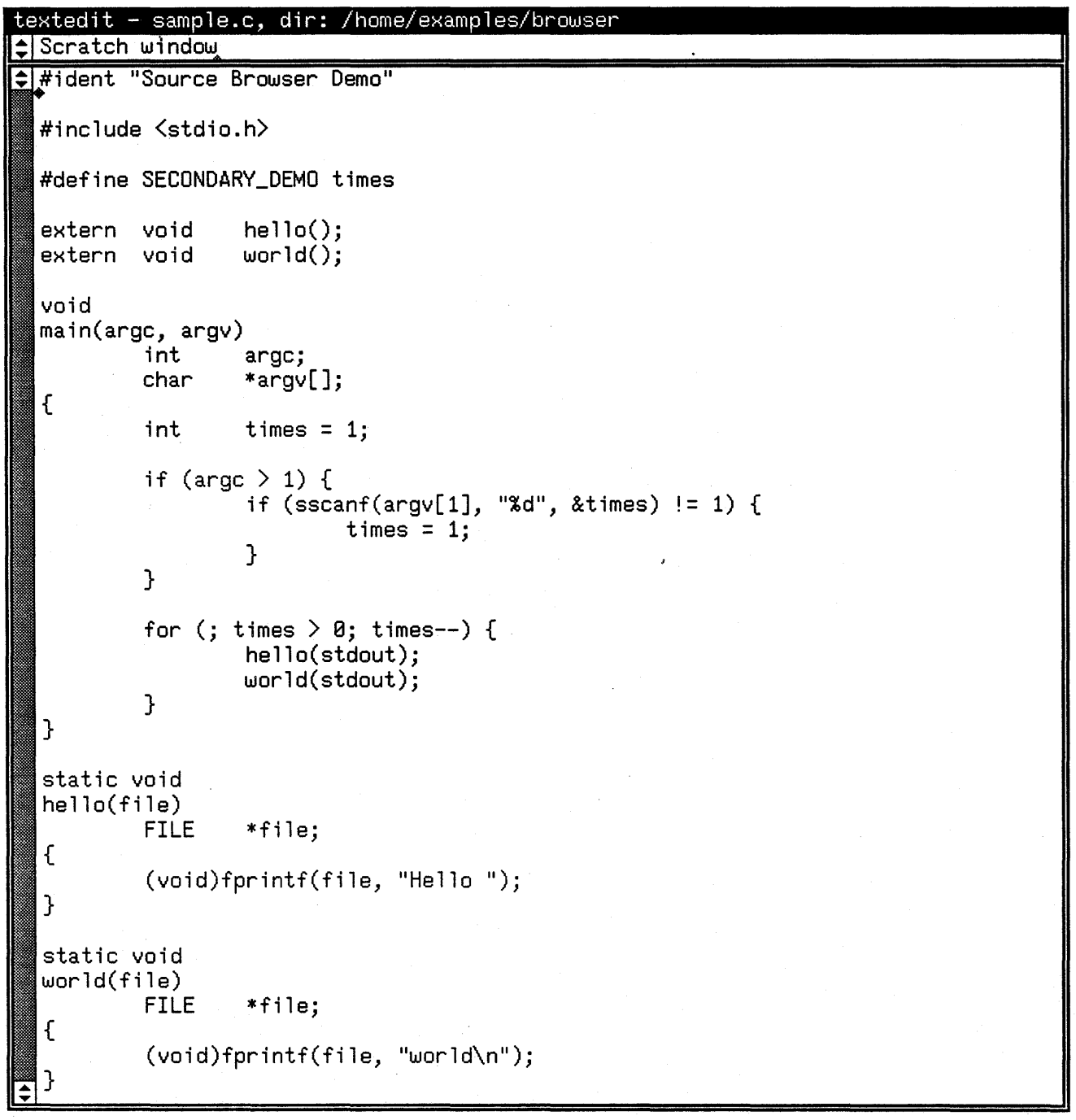
The Sun SourceBrowser command-line environment is a conventional command-style interface that can be accessed from Sun workstations, as well as from character terminals and from workstations emulating terminals.

This chapter explains how to issue a query from the command line and how to use SourceBrowser's command-line options.

5.1 Sample Program

Throughout this chapter, the following sample program is used as the basis for all examples. This program is written in C. For information about using SourceBrowser with Sun FORTRAN programs, see Appendix A. For information about using SourceBrowser with Sun Pascal programs, see Appendix B.

Figure 5-1 *Sample Program*



5.2 Viewing Command-Line Options

To obtain a list of the SourceBrowser's command-line options,

- type **sbquery** when the Unix shell prompt is displayed

Figure 5-2 *SourceBrowser Command-Line Options*

```
venus@breeze{5}% sbquery
sbquery [options] <symbols>
        All lines that contain symbols are displayed.
General options
-break_lock         Break the database lock.
-no_update          Do not update the index file.
-files_only         Only show the names of the files with matches.
-help_focus         Show focusing options.
-help_filter <language> Show filter options for <language>.
-max_memory <size>      Amount of memory to use when building index.
-no_case            Ignore case information during lookup
-no_source          Do not show source line containing match.
-o <file>           Send output to <file>.
-reg_expr           Use grep-style regular expressions.
-symbols_only       Display list of symbols that match search pattern.
-version            Display the current version number.
```

5.3 Issuing a Query

To issue a query from the command line:

- type **sbquery**, followed by any command-line options and their arguments, followed by the symbol you want to search for

For example, in Figure 5-3 the user instructed SourceBrowser to ignore all case information and display all the appropriate matches, that is, all lines of code in the current working directory that contain the symbol *times*.

SourceBrowser then displays a listing that includes the file in which the symbol appears, the line number on which the symbol appears, the function, if any, in which the symbol appears, and the line of source code containing the symbol.

Figure 5-3 *Issuing a Query from the Command Line*

```
venus@breeze{6}% sbquery -no_case times
"sample.c", line 15:[main] int   times = 1;
"sample.c", line 5:#define SECONDARY DEMO times
"sample.c", line 18:[main] if (sscanf(argv[1],"%d",&times) != 1) {
"sample.c", line 19: [main] times = 1;
"sample.c", line 23: [main] for (; times > 0; times--)  {
"sample.c", line 23: [main] for (; times > 0; times--)  {
```

NOTE: SourceBrowser's default is to search for symbols in the SourceBrowser database in the current working directory. If you want to browse a database stored in another directory, see Chapter 3, "Browsing Large Programs."

NOTE: SourceBrowser can search for identifiers, string constants, and search patterns that contain wildcards. SourceBrowser's default is to search for identifiers. For instructions on searching for a string constant, see Section 2.14, "Searching for String Constants." For instructions on using wildcards in a query, see Section 2.15, "Using Wildcards in Queries."

5.4 Command-Line Options

The following is a description of each SourceBrowser command-line option.

-break_lock

Breaks the lock on a locked database. If the update of the index file is aborted for some reason (e.g. a power failure), SourceBrowser will display a message telling you that the database is locked the next time you issue a query. When you use the `-break_lock` option to break the lock, your database may be in an inconsistent state. To ensure consistency, remove the .sb subdirectory and recompile.

-no_update

Causes SourceBrowser not to rebuild the index file when you issue a query following compilation. If you do not include this option and issue a query following compilation or recompilation, SourceBrowser updates the database; then processes your query.

For more information, see section 2.5, "Building the SourceBrowser Database."

Menu Command Equivalent

This option is equivalent to setting the `Keep Database Index Updated` item in the property sheet to `no`.

-files_only

Lists the files in which the symbol you are searching for appears. When you use this command with back quotes, you can take SourceBrowser output and use it as an argument to another command. For example, this command

```
edit `sbquery -files_only hello`
```

reads into the editor all files containing the symbol *hello*. For information about using back quotes to redirect output to another command see `sh`(1) or `csh`(1).

-help_focus

Displays a list of the classes of code that you can focus on. You then can issue a focused query, that is, a query that is limited to specific classes of code such as specific programs or functions. Because SourceBrowser automatically searches through all files described by the database in the current working directory, the `Focus` option is especially useful when you want to limit your search to certain programs in a directory.

Issuing the `-help_focus` option without any arguments displays a list of the classes of code that you can use to limit your search.

Figure 5-4 *Using -help_focus To Display Focusing Options*

```
venus@breeze{9}% sbquery -help_focus
-in_program <Program>   Focus query on Program units
-in_library <Library>   Focus query on Library units
-in_file <File>         Focus query on File units
-in_file <Macro>        Focus query on Macro units
-in_file <Function>     Focus query on Function units
-in_language <Language> Focus query on Language units
```

To limit your search to a specified class of code, type **sbquery** followed by the class of code you want to focus on, followed by the specific unit of code you want to search in, followed by the symbol you want to search for. Note that you can include multiple classes of code in a query.

Because SourceBrowser automatically searches through all files described by the database in the current working directory, the `Focus` option is especially useful when you want to limit your search to certain programs in a directory.

In Figure 5-5, the user instructed SourceBrowser to find all instances in which *times* is used in function `main`. Notice that SourceBrowser displays one fewer match than in Figure 5-3, which was not a focused query.

Figure 5-5 *Using -help_focus To Issue a Focused Query*

```
venus@breeze{10}% sbquery -in_function main times
"sample.c", line 15:[main] int   times = 1;
"sample.c", line 18:[main] if (sscanf(argv[1],"%d",%times) ! = 1) {
"sample.c", line 19:[main] times = 1;
"sample.c", line 23:[main] for (; times > 0; times--)  {
"sample.c", line 23:[main] for (; times > 0; times--)  {
```

Menu Command Equivalent

This option is equivalent to the `Focus` command in SourceBrowser's window environment.

-help_filter

Issuing this option without any arguments causes SourceBrowser to display a list of the languages for which filtering options are available. (You use filtering options to search for symbols based on how they are used in a program.)

Figure 5-6 *Using -help_filter To Display Supported Languages*

```
venus@breeze{11}% sbquery -help_filter
The '-help_filter' option must be given with one of the following arguments:
        -help_filter ansi_c
        -help_filter base
        -help_filter cpp
        -help_filter executable
        -help_filter extend
        -help_filter focus
        -help_filter library
        -help_filter sun_c
        -help_filter sun_cpp
        -help_filter sun_f77
        -help_filter sun_pascal
```

Issuing this option followed by a language supported by SourceBrowser causes SourceBrowser to display a list of filtering options for that language. Figure 5-7 shows a partial listing of these options for Sun C.

Figure 5-7 *Using -help_filter To Display Filtering Options*

```
venus@breeze{12}% sbquery -help_filter sun_c | more
--all_matches
          All mentions of the symbol
-all_matches-all_matches
          All mentions of the symbol
-all_matches-variables
          All uses as a variable
-all_matches-variables-all_variables
          All uses as a variable
-all_matches-variables-regular_variables
          All uses as a non-field variable
-all_matches-variables-field_variables
          All uses as a field
-all_matches-variables-field_variables-all_fields
          All uses as a field
-all_matches-variables-field_variables-struct_fields
          All uses as a struct field
...
```

Specifying the filter followed by the symbol you want to search for causes SourceBrowser to conduct a filtered query.

For example, in Figure 5-8 the user instructed SourceBrowser to find all instances in which *times* is used as a variable declaration.

Figure 5-8 *Issuing a Filtered Query*

```
venus@breeze{6}% sbquery -declaration-variables times
"sample.c", line 15:[main] int   times = 1;
"sample.c", line 5:?#define SECONDARY_DEMO times
```

The filter can be shortened between the dashes as long as it still is a unique filter. For example, *-declaration-variables* could be shortened to *-decl-var*.

Note that the question mark (?) that appears in the final line of output in Figure 5-8 represents a secondary match. Because SourceBrowser is unable to determine precisely how identifiers inside macro definitions are used in a program, SourceBrowser identifies all identifiers inside macro definitions as secondary matches. For more about secondary matches, see Section 2-16, *Using the `Filter` Command to Narrow a Search.*

Menu Command Equivalent

This option is equivalent to the `Filter` command in SourceBrowser's window environment.

-max_memory <size>

Tells SourceBrowser the approximate amount of memory, in Megabytes, that should be allocated before SourceBrowser uses temporary files when building the index file.

Menu Command Equivalent

This option is equivalent to specifying the `Database Update Memory Allocation` item in the property sheet.

-no_case

Instructs SourceBrowser not to look at case when searching for symbols in the source code that match the query symbol. If you do not use this option, SourceBrowser only makes a match when the case of the symbol matches the case of the symbol in the source code.

Menu Command Equivalent

This option is equivalent to setting the `Query Matching` item in the property sheet to `Ignore Case`.

-no_source

Causes SourceBrowser to display only the filename and line number associated with each match. If you do not include this option, SourceBrowser also displays the line of source code containing the match.

Menu Command Equivalent

This option is equivalent to setting the `Match Window Contents` option in the property sheet to `Filename and Line Number Only`.

-o <file>

Sends SourceBrowser output to a file instead of to standard out.

-reg_expr

Indicates that you are using regular expressions when issuing a query that includes wildcards. If you do not include this option, SourceBrowser assumes you are using shell-style patterns.

See Section 2.15, *Using Wildcards in Queries,* for more about using wildcards in queries and about regular expressions. See `sh(1)` for information about shell-style pattern matching.

Menu Command Equivalent

This option is equivalent to setting the `Wildcard Style` item in the property sheet to `Regular Expressions.`

-symbols_only

Displays a list of all symbols that match the pattern specified in your search pattern. This option is only useful when you use wildcards in a query.

-version

Displays SourceBrowser's current version number.

Menu Command Equivalent

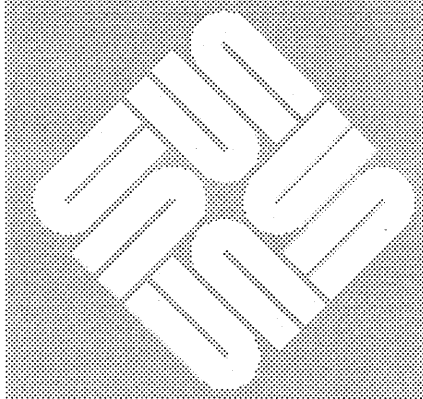
This option is equivalent to choosing `Version` from the `Show` menu.

5.5 Non-Standard Installation Procedure

SourceBrowser relies on finding the file sun_source_browser.ex in a certain location. If you install in a non-standard way such that the file is not found where expected, the system returns an error message when you run a SourceBrowser. To correct the error, set the environment variable SUN_SOURCE_BROWSER_EX_FILE to the absolute pathname of sun_source_browser.ex. The variable must be set each time you run SourceBrowser.

A

Browsing FORTRAN Programs



A

Browsing FORTRAN Programs

This appendix explains those features of Sun SourceBrowser that are unique to browsing FORTRAN files. Specifically, this appendix

- explains how to enable SourceBrowser from FORTRAN
- shows how to browse a FORTRAN program from the window environment
- shows how to browse a FORTRAN program from the command-line
- describes how to set the case option so it is suitable for browsing FORTRAN programs in the window and command line environments

A.1 Startup

To turn on SourceBrowser if you are using `make`, add `-sb` to FFLAGS in the makefile.

To turn on SourceBrowser if you are not using `make`, add the `-sb` option to the FORTRAN compiler command line.

Figure A-1 *Enabling Browsing from FORTRAN*

```
venus% f77 -sb hello.f -c
```

A.2 Sample Program

Throughout this appendix, the following sample FORTRAN program is used as the basis for the examples.

Figure A-2 *Sample FORTRAN Program*

textedit - example.f (edited), dir: /home/examples/browser/fortran

Scratch window

```
      program example
      integer argc
      integer times
      character*40 argv

      data times/1/

      argc = iargc()
      if (argc .le. 0) goto 10
      call getarg(1, argv)
      read (unit=argv, fmt=*, err=99) times
      goto 10
99    times = 1
10    do 20 count = 1, times
         call hello(6)
         call world(6)
20    continue

      end

      subroutine hello(ud)
      integer ud

      write (ud, *) 'Hello '
      end

      subroutine world(ud)
      integer ud

      write (ud, *) 'world'
      end
```

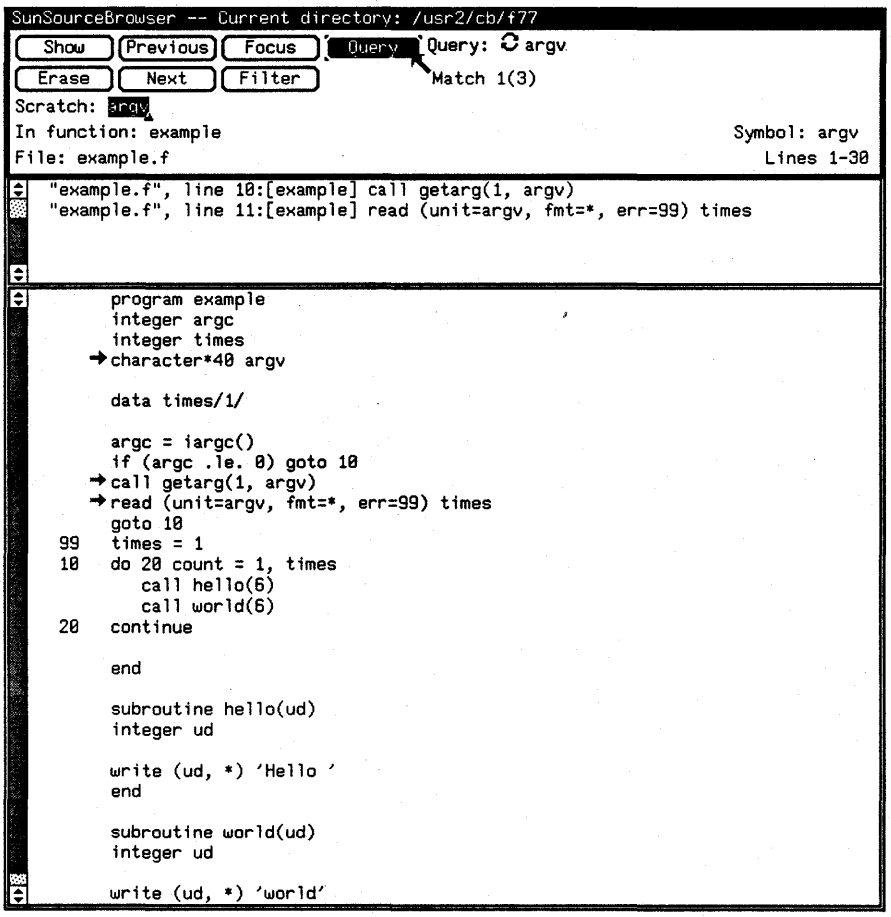
A.3 Browsing from the Window Environment

Chapter 2, *Using Sun SourceBrowser*, provides a complete description of the SourceBrowser window environment. This section provides two brief examples of browsing FORTRAN source code.

Issuing a Query

You use the same procedure to query a FORTRAN program that you use to query a C program. For example, in Figure A-3, the user instructed SourceBrowser to find all instances of *argv* by typing *argv* in the Scratch field selecting *argv*, and then clicking left on the `Query` button.

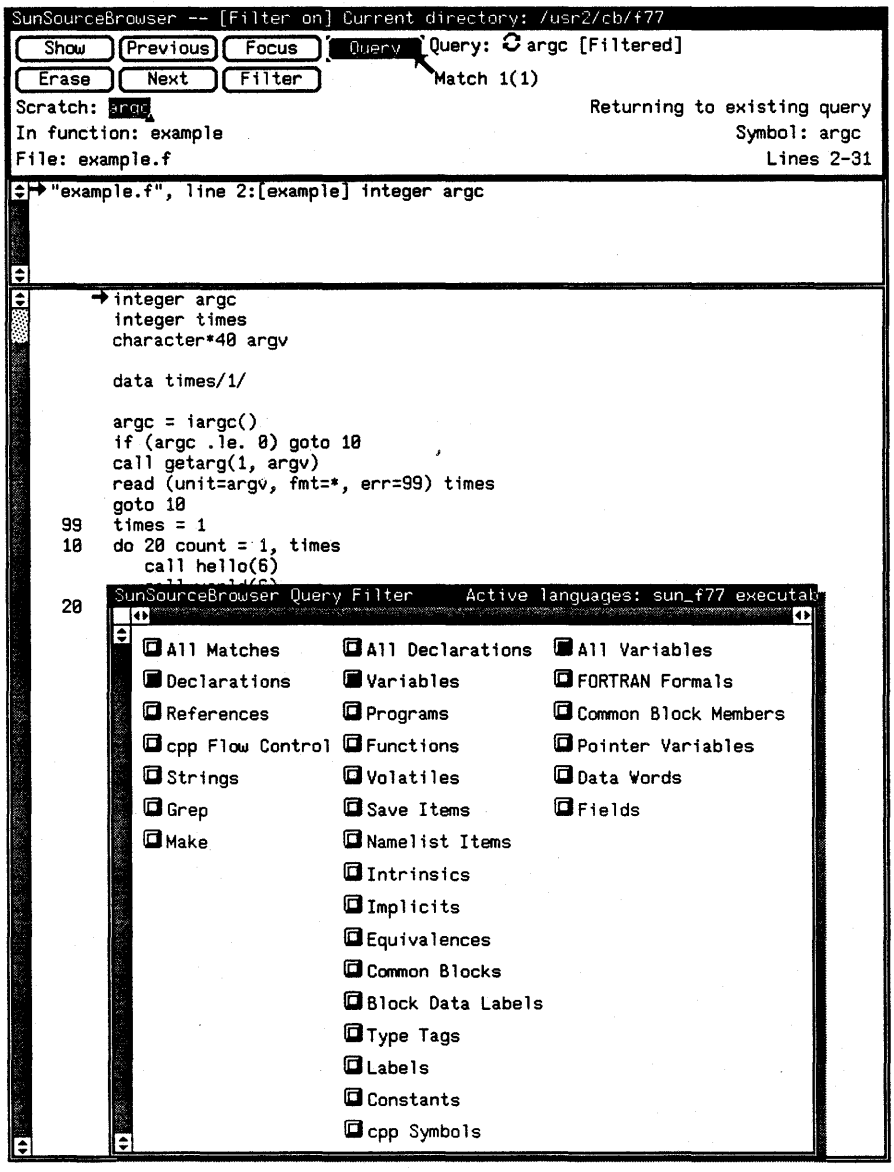
Figure A-3 *Issuing a Query*



Issuing a Filtered Query

SourceBrowser's filtering options are determined by the language of the source code you are browsing. Figure A-4 shows an example of the Filter panel that is displayed when you query a FORTRAN program. In that example, the user instructed SourceBrowser to find all instances in which *argc* is used as a variable declaration.

Figure A-4 *Issuing a Filtered Query*



A.4 Using the Command-Line Environment

Chapter 5, "Command-Line Reference," contains a complete description of the SourceBrowser command-line environment. This section contains a brief example of browsing a FORTRAN program from the command line.

To issue a query from the command line:

- type **sbquery**, followed by any command-line options and their arguments, followed by the symbol you want to search for

For example, in Figure A-5 the user has instructed SourceBrowser to ignore all case information and display all the appropriate matches, that is, all lines of code in the current working directory that contain the symbol *argv*.

SourceBrowser then displays a listing that includes the file in which the symbol appears, the line number on which the symbol appears, the function in which the symbol appears, and the line of source code containing the symbol.

Figure A-5 *Issuing a Query from the Command Line*

```
venus@breeze{10}% sbquery -no_case argv
"example.f", line 4:[example] character*40 argv
"example.f", line 10:[example] call getarg(1 argv)
"example.f", line 11:[example] read (unit=argv, fmt=*, err=99) times
```

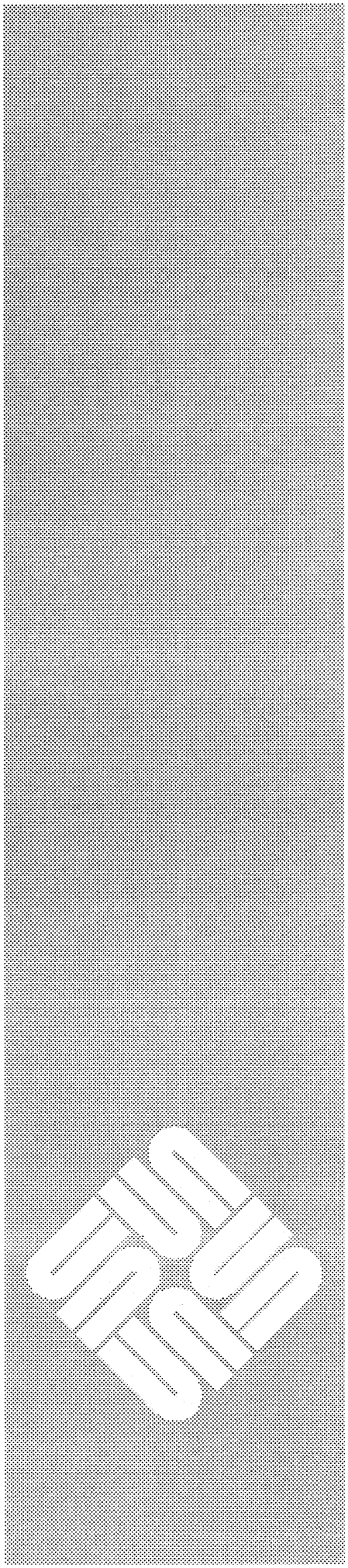
A.5 Turning Off Case

When browsing FORTRAN programs, SourceBrowser should ignore case. There are two ways to set the case option:

- If you are using SourceBrowser from the window environment, set the `Query Matching` option in the Property Sheet to `Ignore Case`. See Section 4.2, "SourceBrowser Property Sheet," for more information.
- If you are using SourceBrowser from the command line, include the `-no_case` option when issuing a query, as in Figure A-5.

B

Browsing Pascal Programs



B

Browsing Pascal Programs

This appendix explains those features of Sun SourceBrowser that are unique to browsing Pascal files. Specifically, this appendix

- explains how to enable SourceBrowser from Pascal
- shows examples of using SourceBrowser's window environment to query a Pascal program
- shows an example of using SourceBrowser's command-line environment to query a Pascal program

B.1 Startup

To turn on SourceBrowser if you are using `make`, add `-sb` to PFLAGS in the makefile.

To turn on SourceBrowser if you are not using `make`, add the `-sb` option to the Pascal compiler command line.

Figure B-1 *Enabling Browsing from* Pascal

```
venus% pc -sb hello.p -c
```

B.2 Sample Program

Throughout this appendix, the following sample Pascal program is used as the basis for all examples.

Figure B-2 *Sample Pascal Program*

textedit - example.p, dir: /home/examples/browser/pascal

Scratch window

```
program example(input, output);
type
        argv_type = packed array[1..100] of char;
var
        argc_cnt: integer;
        count: integer;
        argv_string: argv_type;
        format: argv_type;

        procedure sscanf(ptr, format: univ_ptr; var i: integer); external;

        procedure hello(var fd: text);
        begin
                write(fd, 'Hello ');
        end;

        procedure world(var fd: text);
        begin
                writeln(fd, 'world');
        end;

begin
        if argc > 1 then begin
                argv(1, argv_string);
                format := '%d';
                sscanf(addr(argv_string), addr(format), argc_cnt);
        end else begin
                argc_cnt := 1;
        end;
        for count := 1 to argc_cnt do begin
                hello(output);
                world(output);
        end;

end.
```

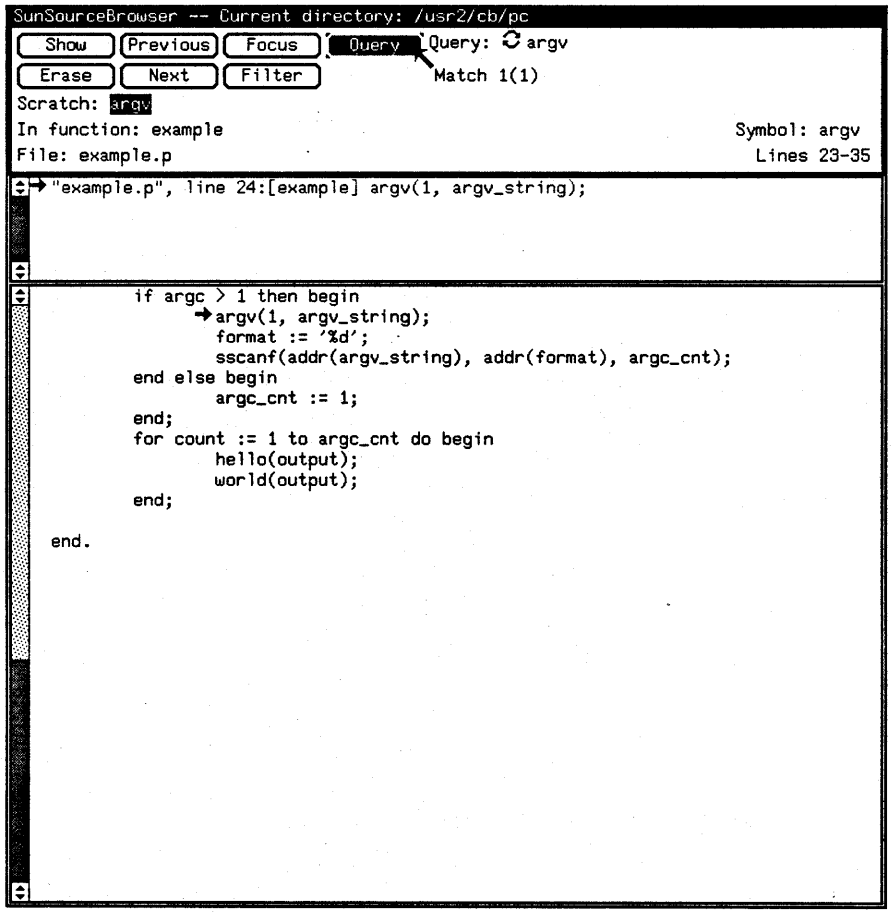
B.3 Browsing from the Window Environment

Chapter 2, "Using Sun SourceBrowser," provides a complete description of the SourceBrowser window environment. This section provides two brief examples of browsing Pascal source code.

Issuing a Query

You use the same procedure to query a Pascal program that you use to query a C program. For example, in Figure B-3, the user instructed SourceBrowser to find all instances of *argv* by typing *argv* in the Scratch field, selecting *argv*, and then clicking left on the `Query` button.

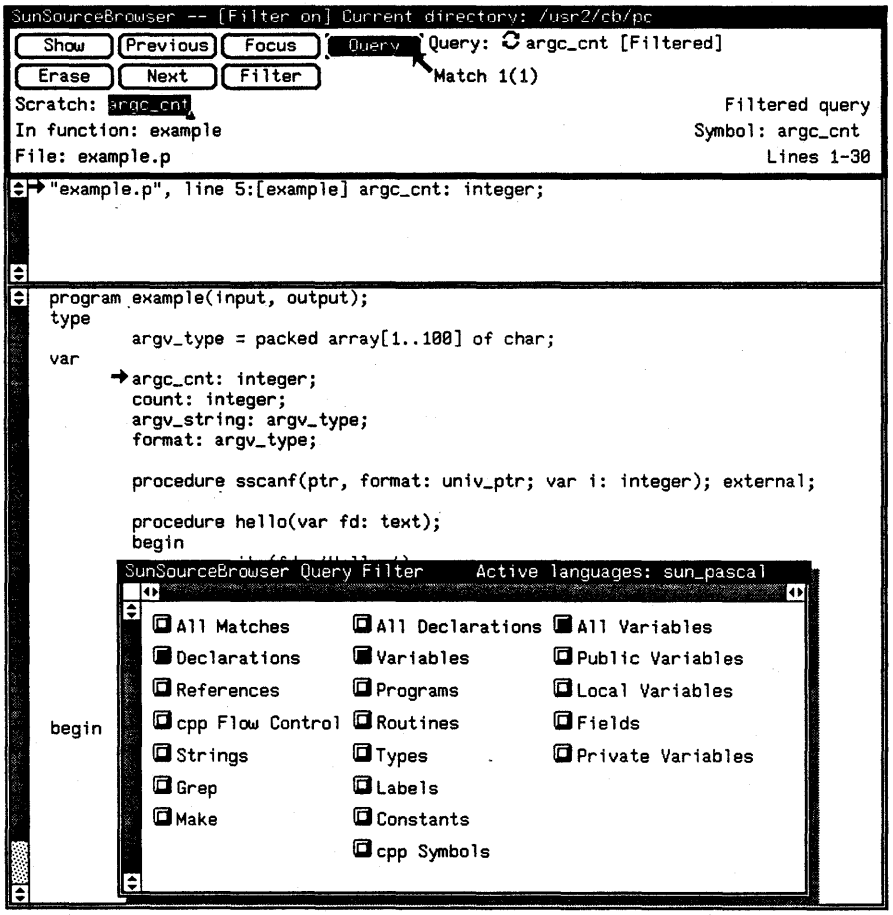
Figure B-3 *Issuing a Query*



Issuing a Filtered Query

SourceBrowser's filtering options are determined by the language of the source code you are browsing. Figure B-3 shows an example of a Filter panel that is displayed when you query a Pascal program. In that example, the user instructed SourceBrowser to find all instances in which *argc_cnt* is used as a variable declaration.

Figure B-4 *Issuing a Filtered Query*



B.4 Issuing a Query from the Command Line

To issue a query from the command line:

- type **sbquery**, followed by any command-line options and their arguments, followed by the symbol you want to search for

For example, in Figure B-5, the user instructed SourceBrowser to display all the appropriate matches, that is, all lines of code in the current working directory that contain the symbol *argv*.

SourceBrowser then displays a listing that includes the file in which the symbol appears, the line number on which the symbol appears, the function, if any, in which the symbol appears, and the line of source code containing the symbol.

Figure B-5 *Issuing a Query from the Command Line*

```
venus@breeze[10]% sbquery argv
"example.p", line 24:[example] argv(1, argv_string);
```

Index

D

E

F

H

I

Q

R

S

T

U

V

W

Revision History

Version	Date	Comments
A	16 March 1990	FCS of Sun SourceBrowser 1.0